TASKING®

Getting Started with TASKING SmartCode

Table of Contents

Chapter 1. Product Overview

TASKING® SmartCode consists of several toolsets, which are listed below. This Getting Started manual and the User Guides describe all features available. In this manual, **TASKING SmartCode** and **SmartCode** are used as synonyms.

C/C++ compiler toolsets

A C/C++ compiler toolset is supported for the following core:

• TriCore™/AURIX™ TC4x.

C compiler toolsets

C compiler toolsets are supported for the following cores:

• Generic Timer Module (GTM) / Multi Channel Sequencer (MCS). In the product and manuals we use the term "MCS".

• Standby Controller (SCR). This controller is based on Infineon® XC800 microcontroller, which is an 8051 compatible 8-bit microcontroller. In the product and manuals we use the term "8051".

• Infineon Parallel Processing Unit (PPU).

All toolsets are accessible from the Eclipse™ Integrated Development Environment (IDE).

Chapter 2. Preparing for First Use

This chapter guides you through the installation process of TASKING® SmartCode. It also describes which documentation is available and how you best can use it.

In this manual, **TASKING SmartCode** and **SmartCode** are used as synonyms.

2.1. Installing the Software

This section describes the installation of the embedded software for Windows and Linux. It also describes how to license the software.

2.1.1. Installation for Windows

System Requirements

Before installing, make sure the following minimum system requirements are met:

• 64-bit version of Windows 7 or higher

• 4 GB memory

• 5 GB free hard disk space

• Screen resolution: 1024 x 768 or higher

Installation

1. If you received a download link, download the software and extract its contents.

    - or -

    If you received an USB flash drive, insert it into a free USB port on your computer.

2. Run the installation program (**setup.exe**).

    The TASKING Setup dialog box appears.

3. Select a product and click on the **Install** button. If there is only one product, you can directly click on the **Install** button.

4. Follow the instructions that appear on your screen. During the installation you need to enter a license key, this is described in Section 2.1.3, *Licensing*.

2.1.2. Installation for Linux

System Requirements

Before installing, make sure the following minimum system requirements are met:

* 64-bit version of Linux

* Linux distributions must support GLIBC version 2.11 or higher. You can check the version with the command `ldd --version`

* 4 GB memory

* 5 GB free hard disk space

* Screen resolution: 1024 x 768 or higher

Installation

TASKING VX-toolsets on Linux are delivered as GNU zipped tar files (e.g. SW260800-v10.3r1.tar.gz). The installation is basically just unpacking this file.

1. Goto the directory where you want to install the product. For example, `/opt`, and issue the following command:

   ```
   tar xzvf SW260800-v10.3r1.tar.gz -C /opt
   ```

2. License the software as described in Section 2.1.3, *Licensing*.

2.1.3. Licensing

TASKING products are protected with TASKING license management software (TLM). To use a TASKING product, you must install that product and install a license.

The following license types can be ordered from TASKING.

Node-locked license

A node-locked license locks the software to one specific computer so you can use the product on that particular computer only.

For information about installing a node-locked license see Section 2.1.3.3.2, *Installing Server Based Licenses (Floating or Node-Locked)* and Section 2.1.3.3.3, *Installing Client Based Licenses (Node-Locked)*.

Floating license

A floating license is a license located on a license server and can be used by multiple users on the network. Floating licenses allow you to share licenses among a group of users up to the number of users (seats) specified in the license.

For example, suppose 50 developers may use a client but only ten clients are running at any given time. In this scenario, you only require a ten seats floating license. When all ten licenses are in use, no other client instance can be used. Also a linger time is in place. This means that a license seat is locked for a period of time after a user has stopped using a client. The license seat is available again for other users when the linger time has finished.

For information about installing a floating license see Section 2.1.3.3.2, *Installing Server Based Licenses (Floating or Node-Locked)*.

Build server license

A build server license is a license on a central computer system, shared (used) by other systems, automated processes or users. A build server license is limited to a maximum number of parallel processes per user (seat). A build server license is usually installed as a floating license with a process limit.

When the maximum number of parallel processes is reached a new client process is set to waiting until the number of running client processes under the same license seat is below the process limit.

License service types

The license service type specifies the process used to validate the license. The following types are possible:

- **Client based** (also known as 'standalone'). The license is serviced by the client. All information necessary to service the license is available on the computer that executes the TASKING product. This license service type is available for node-locked licenses only.

- **Server based** (also known as 'network based'). The license is serviced by a separate license server program that runs either on your companies' network or runs in the cloud. This license service type is available for both node-locked licenses and floating licenses.

  Licenses can be serviced by a cloud based license server called "**TASKING Remote License Server**". This is a license server that is operated by TASKING. Alternatively, you can install a license server program on your local network. Such a server is called a "**TASKING Local License Server**". You have to configure such a license server yourself. The installation of a TASKING local license server is not part of this manual. You can order it as a separate product (SW000089).

  The benefit of using the TASKING Remote License Server is that product installation and configuration is simplified.

  Unless you have an IT department that is proficient with the setup and configuration of licensing systems we recommend to use the facilities offered by the TASKING Remote License Server.

2.1.3.1. Obtaining a License

You need a license key when you install a TASKING product on a computer. If you have not received such a license key follow the steps below to obtain one. Otherwise, you cannot install the software.

Obtaining a server based license (floating or node-locked)

- Order a TASKING product from TASKING or one of its distributors.

A license key will be sent to you by email or on paper.

If your node-locked server based license is not yet bound to a specific computer ID, the license server binds the license to the computer that first uses the license.

Obtaining a client based license (node-locked)

To use a TASKING product on one particular computer with a license file, TASKING needs to know the computer ID that uniquely identifies your computer. You can do this with the **getcid** program that is available on the TASKING website. The detailed steps are explained below.

1. Download the **getcid** program from https://www.tasking.com/support/tlm/downloads.

2. Execute the **getcid** program on the computer on which you want to use a TASKING product. The tool has no options. For example,

    Windows (GUI):

    ```
    getcid_version
    ```
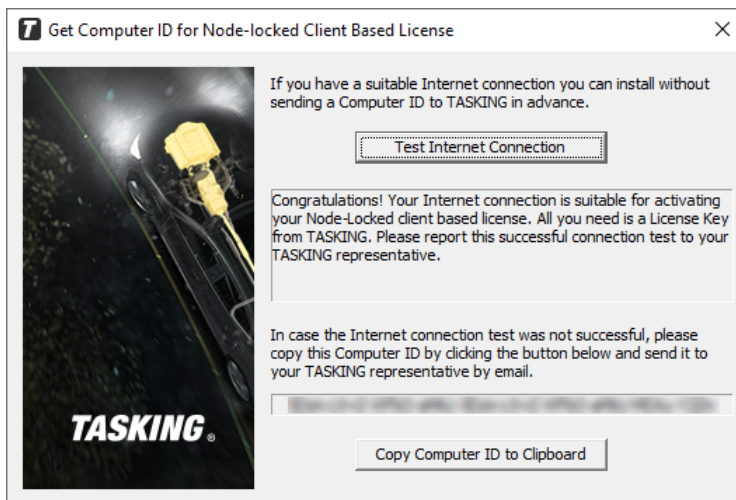

    The computer ID is displayed in the lower part of the dialog.

    Windows (command prompt):

    ```
    getcidcmd_version
    ```

    The computer ID is displayed on your screen.

    Linux:

    ```
    ./getcid_version
    ComputerID: 5Dzm-L9+Z-WFbO-aMkU-5Dzm-L9+Z-WFbO-aMkU-MDAy-Y2Zm
    ```

    The computer ID is displayed on your screen.

3.   Order a TASKING product from TASKING or one of its distributors and supply the computer ID.

A license key and a license file will be sent to you by email or on paper.

When you have received your TASKING product, you are now ready to install it.

2.1.3.2. Frequently Asked Questions (FAQ)

If you have questions or encounter problems you can check the support page on the TASKING website.

https://www.tasking.com/support/tlm/faqs

This page contains answers to questions for the TASKING license management system TLM.

If your question is not there, please contact your nearest TASKING Sales & Support Center or Value Added Reseller.

2.1.3.3. Installing a License

On Windows the license setup procedure is done by the installation program. On Linux you have to install a license manually.

If the installation program on Windows can access the internet then you only need the license key. Given the license key the installation program retrieves all required information from the remote license server. The installation program sends the license key and the computer ID of the computer on which the installation program is running to the remote license server, no other data is transmitted.

If the installation program cannot access the internet the installation program asks you to enter the required information by hand. If you install a node-locked client based license you should have the license file at hand (see Section 2.1.3.1, *Obtaining a License*).

Floating licenses are always server based and node-locked licenses can be server based. All server based licenses are installed using the same procedure.

2.1.3.3.1. Configure the Firewall in your Network

For using the TASKING license servers the TASKING license manager tries to connect to the remote license servers `lic1.tasking.com`, `lic2.tasking.com`, `lic3.tasking.com`, `lic4.tasking.com` at the TCP ports 8080, 8936 or 80. Make sure that the firewall in your network is transparently enabled for one of these ports.

2.1.3.3.2. Installing Server Based Licenses (Floating or Node-Locked)

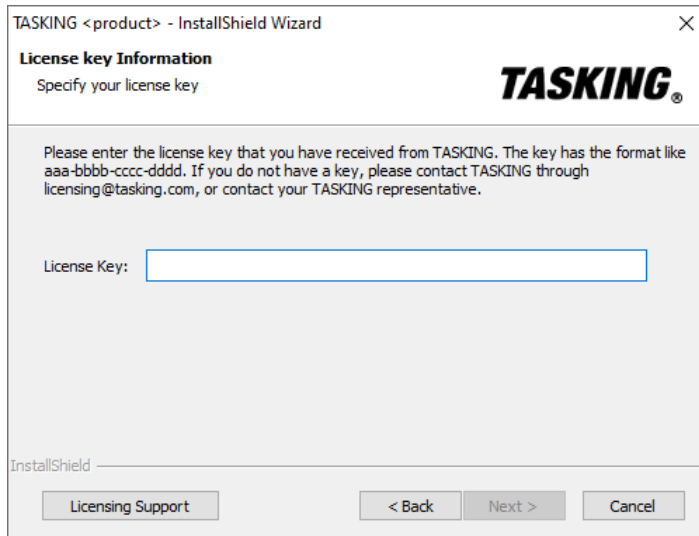If you do not have received your license key, read Section 2.1.3.1, *Obtaining a License* before you continue.

Windows

1.   If you want to use a local license server, first install and run the local license server before you continue with step 2. You can order a local license server as a separate product (SW000089).

2.   Install the TASKING product and follow the instructions that appear on your screen.

The installation program asks you to enter the license information.

3. In the **License Key** field enter the license key you have received from TASKING and click **Next** to continue.

   The installation program tries to retrieve the license information from a remote license server. Wait until the license information is retrieved. If the license information is retrieved successfully subsequent dialogs are already filled-in and you only have to confirm the contents of the dialogs by clicking the **Next** *button. If the license information is not retrieved successfully you have to enter the information by hand.*

4. Select your **License Type** and click **Next** to continue. If the license type is already filled in and grayed out, you can just click **Next** to continue.

   You can find the license type in the email or paper that contains the license key.

5. (For floating licenses only) Select **Remote license server** to use one of the remote license servers, or select **Local license server** for a local license server. The latter requires optional software.

   (For local license server only) specify the **Server name** and **Server port** of the local license server.

6. Click **Next** and follow the rest of the instructions to complete the installation.

Linux

1. If you want to use a local license server, first install and run the local license server before you continue with step 2. You can order a local license server as a separate product (SW000089).

2. After installation of the TASKING product, the license options file `licopt.txt` is present in the directory *<install-dir>*/*<product-name>*/etc, for example /opt/SmartCode/etc for the SmartCode product, assuming you have installed the TASKING product in /opt. Edit this file with a text editor and change the line:

```
TSK_LICENSE_KEY_SW260800 = 0000-0000-0000-0000
```

Replace `0000-0000-0000-0000` with the license key you obtained from TASKING.

3. Set the environment variable `TSK_OPTIONS_FILE_<Product-Code><version>` to the absolute location of `licopt.txt`. For example:

```
TSK_OPTIONS_FILE_SW260800v10_3r1=/opt/SmartCode/etc/licopt.txt
export TSK_OPTIONS_FILE_SW260800v10_3r1
```

2.1.3.3.3. Installing Client Based Licenses (Node-Locked)

If you do not have received your license key and license file, read Section 2.1.3.1, *Obtaining a License* before continuing.

Windows

1. Install the TASKING product and follow the instructions that appear on your screen.

   The installation program asks you to enter the license information.

2. In the **License Key** field enter the license key you have received from TASKING and click **Next** to continue.

   *The installation program tries to retrieve the license information from a remote license server. Wait until the license information is retrieved. If the license information is retrieved successfully subsequent dialogs are already filled-in and you only have to confirm the contents of the dialogs by clicking the **Next** button. If the license information is not retrieved successfully you have to enter the information by hand.*
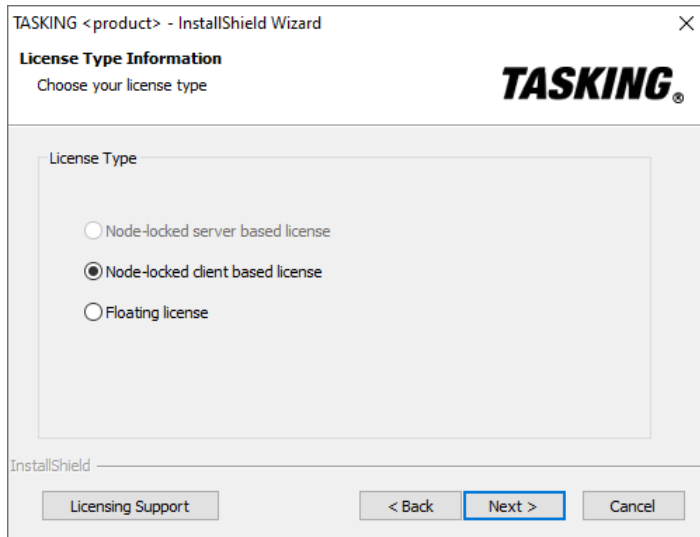
3. Select **Node-locked client based license** and click **Next** to continue.
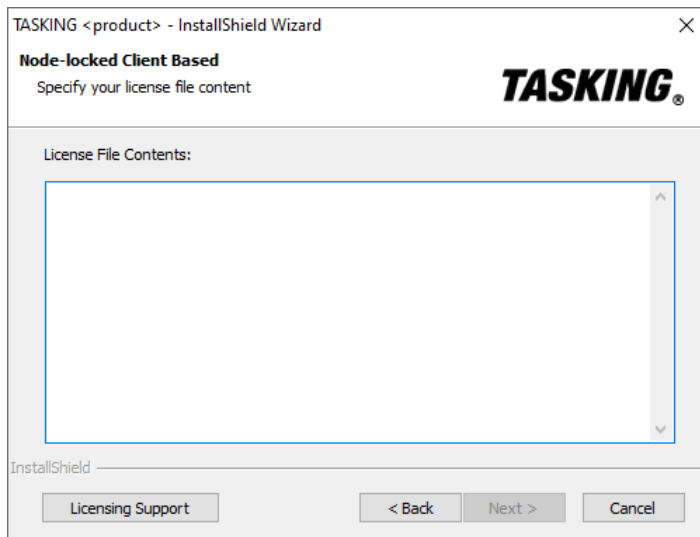
4. In the **License File Contents** field enter the contents of the license file you have received from TASKING.

    The license data is stored in the file licfile.txt in the etc directory of the product (<install_dir>\etc).

5. Click **Next** and follow the rest of the instructions to complete the installation.

Linux

1. Install the TASKING product and put the license file `licfile.txt` you have received from TASKING in the directory *<install-dir>*/*<product-name>*/`etc`. For example, if you installed a SmartCode product in `/opt`, the path is `/opt/SmartCode/etc`

2. After installation of the TASKING product, the license options file `licopt.txt` is present in the directory *<install-dir>*/*<product-name>*/`etc`. Edit this file with a text editor:

    a. Change the line:

    ```
    TSK_LICENSE_KEY_SW260800 = 0000-0000-0000-0000
    ```

    Replace `0000-0000-0000-0000` with the license key you obtained from TASKING.

    b. Change the line:

    ```
    #TSK_LICENSE_FILE_SW260800 = licfile.txt
    ```

    to:

    ```
    TSK_LICENSE_FILE_SW260800=/opt/SmartCode/etc/licfile.txt
    ```

    Specify the absolute path to the file `licfile.txt`.

3. Set the environment variable `TSK_OPTIONS_FILE_<Product-Code><version>` to the absolute location of `licopt.txt`. For example:

```
TSK_OPTIONS_FILE_SW260800v10_3r1=/opt/SmartCode/etc/licopt.txt
export TSK_OPTIONS_FILE_SW260800v10_3r1
```
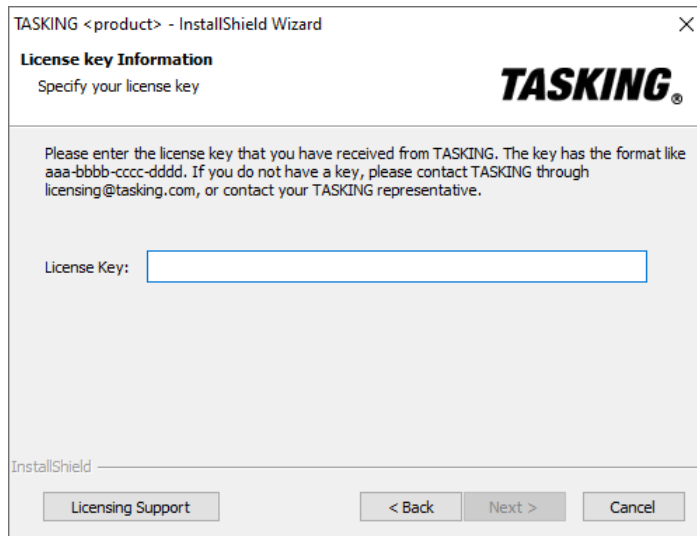
2.2. Starting / Closing the SmartCode Eclipse IDE

The TASKING SmartCode uses Eclipse as the Integrated Development environment (IDE).

Starting the SmartCode Eclipse IDE

To start the SmartCode Eclipse IDE:

1. On Windows PCs, from the **Start** menu, select **TASKING SmartCode v***x.yrz* **» SmartCode Eclipse IDE**.

    On UNIX/Linux workstations, type **eclipse** at a shell command prompt, assuming that the TASKING SmartCode eclipse directory is in the search path.

    The Workspace Launcher dialog appears.

2. Enter the path to the workspace.

    In the remainder of this manual, we assume you use the default.

3. Enable the option **Use this as the default and do not ask again**.

4. Click **OK** to proceed.

Initially, the SmartCode Eclipse IDE opens with a workbench displaying the C/C++ perspective with several views and the **Welcome** view visible. The Welcome view provides some general information and alternative ways to access the online documentation.

*   Close the **Welcome** view when you want to create more space on your screen.

Your workbench should now look similar to this:
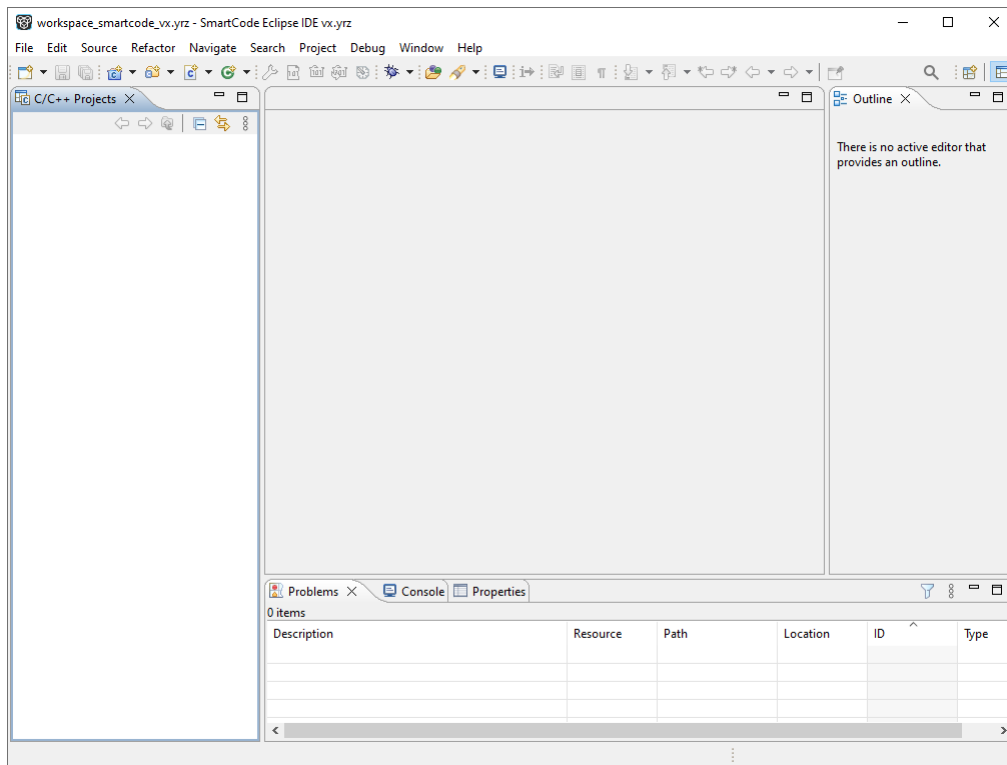
At any time you can get the Welcome view back again by selecting **Welcome** from the **Help** menu.

Closing the SmartCode Eclipse IDE

To close SmartCode Eclipse IDE:

• From the **File** menu, select **Exit**.

Upon exit, the SmartCode Eclipse IDE saves the current workbench layout. The next time you start the SmartCode Eclipse IDE, the last saved workbench layout is used.

2.3. How to Use the Documentation

The documentation for TASKING SmartCode consists of:

• documentation for Eclipse

• this Getting Started manual

• TASKING SmartCode - TriCore User Guide

• TASKING SmartCode - MCS User Guide

• TASKING SmartCode - 8051 User Guide

• TASKING SmartCode - ARC/PPU User Guide

• TASKING winIDEA Help

It is strongly recommended to read the documentation in this order:

Getting acquainted with Eclipse

If you are new to Eclipse, start familiarizing with Eclipse. Eclipse comes with several online documents. One document describes how Eclipse is organized as a Workbench, with Perspectives that contain Views; another document explains how to create a sample C/C++ project, build and debug it (CDT documentation).

To start with this documentation:

1. Start Eclipse.

2. From the **Help** menu, select **Help Contents**.

   The help screen overlays the Eclipse Workbench.

3. In the left pane, select **Workbench User Guide** to learn more about working in Eclipse.

4. Continue with **C/C++ Development User Guide** to learn more about creating and developing a C/C++ project.

   This part of the documentation explains how to create a "hello world" example. Be aware that this example does not use the TASKING tools, it uses the standard GNU compiler in Eclipse instead.

Getting started with SmartCode (this manual)

The TASKING Getting Started and TriCore User Guide contain specific information for the TASKING toolset for TriCore. Its content overrides any information found in the Eclipse and CDT documentation.

The next chapters of this manual explain how to setup and work with a TriCore project. It shows some important features of the TriCore toolset.

TASKING User Guides

Once you are introduced to Eclipse and the TriCore toolset, you can start creating your own projects. The documentation for the TriCore toolset covers the TriCore C/C++/Assembly language, as well as detailed description of the various tools and options. Accessing the documentation for the TriCore toolset is similar to accessing the documentation for Eclipse:

1. Start Eclipse.

2. From the **Help** menu, select **Help Contents**.

   The help screen overlays the Eclipse Workbench.

3. In the left pane, select **TASKING SmartCode - TriCore User Guide** to access the documentation for the TriCore toolset or select one of the other toolset User Guides.

The TASKING manuals are also available in PDF format via de Windows Start menu:

• In the **Start** menu, browse to **TASKING SmartCode v*x.y.rz*** and select the manual you need.

TASKING winIDEA Help

For information how to use the winIDEA debugger, see the online winIDEA Help.

2.4. Related Publications

C Standard

• C A Reference Manual (fifth edition) by Samual P. Harbison and Guy L. Steele Jr. [2002, Prentice Hall]

• ISO/IEC 9899:1999(E), Programming languages - C [ISO/IEC]

• ISO/IEC 9899:2011(E), Information technology - Programming languages - C [ISO/IEC]

• ISO/IEC 9899:2018(E), Information technology - Programming languages - C [ISO/IEC]

  More information on the standards can be found at http://www.iso.org/

• DSP-C, An Extension to ISO/IEC 9899:1999(E), Programming languages - C [TASKING, TK0071-14]

C++ Standard

• ISO/IEC 14882:2011 C++ standard [ISO/IEC]

• ISO/IEC 14882:2014 C++ standard [ISO/IEC]

• ISO/IEC 14882:2017 C++ standard [ISO/IEC]

More information on the standards can be found at http://www.iso.org/

• The C++ Programming Language (second edition) by Bjarne Straustrup [1991, Addison Wesley]

• The Annotated C++ Reference Manual by Margaret A. Ellis and Bjarne Straustrup [1990, Addison Wesley]

CERT C Secure Coding Standard

• The CERT C Secure Coding Standard by Robert C. Seacord [October 2008, Addison Wesley]

• The CERT C Secure Coding Standard web site http://www.securecoding.cert.org/

  For general information about CERT secure coding, see http://www.cert.org/secure-coding

MISRA C

• MISRA C:2012, Guidelines for the use of the C language in critical systems [MIRA Ltd, 2013]

  See also https://misra.org.uk/

• MISRA-C:2004, Guidelines for the Use of the C Language in Critical Systems [MIRA Ltd, 2004]

  See also https://misra.org.uk/

• Guidelines for the Use of the C Language in Vehicle Based Software [MIRA Ltd, 1998]

  See also https://misra.org.uk/

TriCore

• TriCore 1.8 Core Architecture User's Manual [Infineon]

• TC1.8 Functional Description [Infineon]

8051

• TASKING 8051 EABI, Document ID: C51-75 [2022, TASKING BV]

ARC/PPU

• DesignWare ARCv2 ISA Programmer's Reference Manual for DW EV7x Processors, Version 6367-001 [April 2020, Synopsys, Inc.]

• DesignWare EV7x Processor Databook, Version 6368-004 [April 2020, Synopsys, Inc.]

• DesignWare ARCv2 System V ABI Supplement User's Guide, Version 4092-007 [Nov 01, 2019, Synopsys, Inc.]

Chapter 3. Setting up a Project

This tutorial shows how to create an embedded software project with the TriCore toolset. It lets you create your own project with a simple "Hello World" example. The steps to create and build projects for the other toolsets that are part of the TASKING SmartCode product, MCS, 8051 and ARC/PPU, are similar to that of the TriCore toolset.

By now you should be familiar with the Eclipse workbench, perspectives and views. If you are not, please read the Eclipse documentation as described in Section 2.3, *How to Use the Documentation*.

3.1. Create a Project

Set the TASKING C/C++ perspective

Before creating a TriCore project, it is necessary to have the TASKING C/C++ perspective on the workbench. By default, this should be the case when you start Eclipse, but if it is not, do the following:

1. Start Eclipse.

   Eclipse starts with the last saved workbench layout.

2. To open the TASKING C/C++ perspective: from the **Window** menu, select **Open Perspective » Other... » TASKING C/C++**.

   The name of the perspective is displayed in the title bar of the workbench window.

> If you attempt to create a TriCore project while the TASKING C/C++ perspective is not active, Eclipse will ask you to activate the TASKING C/C++ perspective after you finish the **New C/C++ Project** wizard.

Create a TriCore project with the New C/C++ Project wizard

1. From the **File** menu, select **New » TASKING TriCore C/C++ Project**

   The New C/C++ Project wizard appears.

2. Enter a name for your project, for example `myproject`.

   *In the **Location** field you will see the location where the new project will be stored. To change the default location, you can uncheck the **Use default location** check box and browse for an alternative location. However, use the default location for now.*

3. In the **Project type** box you can select whether to create an application, a position independent module or a library.

   * Expand **TASKING TriCore Application** and select **Hello World C Project**. This creates the file `myproject.c` with a simple main function.

   * Click **Next** to continue.

The TriCore Project Settings page appears.
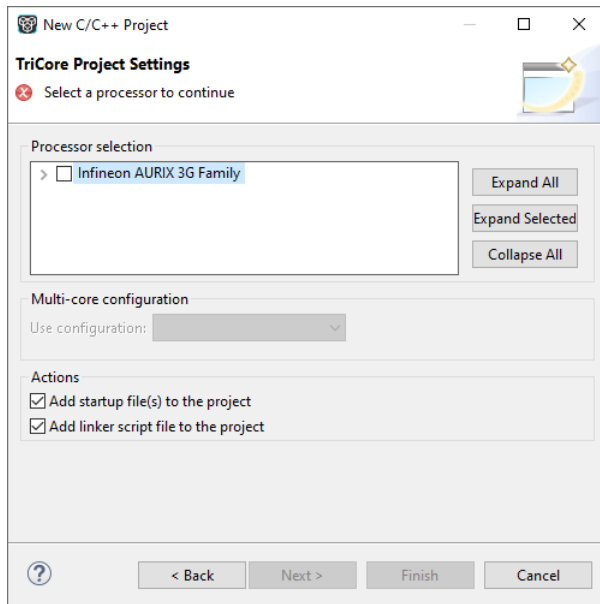
4. Select the target processor for which you want to build the application. For example TC49x. Afterwards you can always change the processor in the **Project » Properties for** dialog.

5. When you selected a multi-core processor, select the multi-core configuration you want to use. If you want to create a multi-core project, select **All cores**. If you want to create a project for a single core on a multi-core processor, select the specific core.

6. You can choose to add C startup code to your project and/or add a linker script file to your project.

   • Enable **Add startup file(s) to the project**. This adds the files `cstart.c` and `cstart.h` to your project (or the `_tccore` variants), which are necessary for building C programs. These files are copies of `lib/src/cstart*.c` and `include/cstart*.h`. If you do not add the startup file(s) here, you can always add them later with **File » New » Startup Files**.

   • Enable **Add linker script file to the project**. This adds the file `myproject.lsl` to your project which can be edited to customize linking and locating. If you do not add the linker script file here, you can always add it later with **File » New » Linker Script File (LSL)**.

   For details on changing the C startup code and linker script file refer to the *TASKING SmartCode - TriCore User Guide.*

7. Click **Next**. This button is only enabled for 'All cores' and 'Core 0' projects.

   The Target Settings page appears.
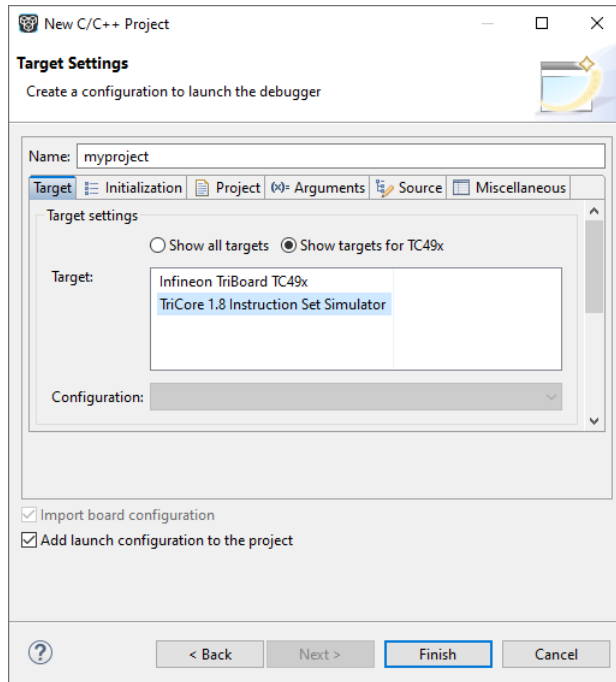
> Note that you can only specify a target launch configuration for an 'All cores' or 'Core 0' project. If you create a single-core project for one of the other cores, you also need a 'Core 0' project to start the other core project. See Section 3.10, *Refer to Another Project from a TriCore Project*.

8. In order to debug your project you need to create a debug configuration.

   • Select a target. You can select a target board or a simulator. For this example we select the **TriCore 1.8 Instruction Set Simulator**.

   • (Optional) If you selected a target board, specify the **Configuration** and **Connection settings**. For the simulator you can skip this.

   • (Optional) If you selected a target board, enable **Import board configuration**. Your project settings, such as memory, flash settings and LSL file (`myproject.lsl`) and startup code (`cstart.h`) will be adjusted to the selected board configuration for you to build your application.

   • Enable **Add launch configuration to the project**. This allows you to debug your project.

   • Leave the other tabs as is. For more information, see section *Creating a Customized Debug Configuration* in Chapter *Using the Debugger* of the *TASKING SmartCode - TriCore User Guide*.

9. Click **Finish** to finish the wizard and to create the project.

*The project has now been created and is the active project. If you click on the **Build myproject** button (⬛), the project is built and should give no errors or warnings.*

The left-hand pane of the Workbench window has two views. The **C/C++ Projects** view shows the structure of your projects, complete with all files that are used in the project.

> In the standard Eclipse documentation about the workbench is described how you can move and organize views on your workbench.

3.2. Delete a Project

The project as you just created, is stored as a subfolder named `myproject` in the folder `C:\Users\`*name*`\workspace_smartcode_`*version*. To delete a project, it needs to be properly removed from the workbench. To delete the project which you just created:

1. In the C/C++ Projects view, right-click on the name of the project, `myproject`, and select **Delete**.

   A dialog appears which asks for confirmation.

2. Enable **Delete project contents on disk (cannot be undone)**. This will remove your project from the workbench and also removes the entire `myproject` subfolder from your hard disk.

   If you disable this option, this would have removed your project from the workbench, but leaves it on you hard disk. Files can be used later in other projects, or you can later import the whole project.

3. Click **OK** to confirm.

3.3. Manually Add a File to Your Project

We will recreate the project as described in Section 3.1, *Create a Project*; however, this time without the automatic 'Hello World' example C source file. Instead, the example below illustrates how you can manually add a file to your project.

Recreate your project without 'Hello World'

1. First repeat steps 1 and 2 of **Create a TriCore project with the New C/C++ Project wizard** in Section 3.1, *Create a Project*.

2. In the **Project type** box you can select whether to create an application or a library.

   • Expand **TASKING TriCore Application** and select **Empty Project**. This creates a project without a C source file containing the function `main()`.

   • Click **Next** to continue.

   The TriCore Project Settings page appears.

3.   Repeat steps 4 through 9 of **Create a TriCore project with the New C/C++ Project wizard** in Section 3.1, *Create a Project*.

4.   Click **Finish** to finish the wizard and to create the project.

Add a new file to your project

To add a new, empty file:

1.   In the C/C++ Projects view, right-click on the name of the project, myproject, and select **New » Source File**.

     The New Source File dialog appears.

2.   Specify a source folder and a name for the new file. By default, the new file will be stored in the project folder (in this case: myproject). If your projects contains multiple folders, you can browse for an alternative source folder to store the new file in.

     •   In the **Source folder** field, make sure it refers to myproject.

     •   In the **Source file** field, type the name of the new file, for example myfile.c. Note that for C files you must specify the extension .c! For C++ files use the extension .C, .cc, .cpp or .cxx.

     •   In the **Template** field, select a code template for your source file, for example Default C source template or select <None> if you want to start with an empty file. Note that you can configure your own templates if you click on the **Configure...** button.

3.   Click **Finish** to continue.

     The new file myfile.c is created and ready for editing in the editor view.

Add an existing file to your project (import)

There are three ways to add a file to your project:

•   Import a file (the original file is copied to the project folder)

•   Create a file in the project folder

•   Create a link in the project folder to an existing file

Import a file

Instead of creating a new file, it is also possible to import an existing file into your project or to create a file directly in the myproject folder. To demonstrate this, follow the steps below. Do not close Eclipse.

•   First create a C source file (for example existing.c) with a standard editor outside Eclipse. (As content you can, for example, use a single line containing comments only).

•   You can store the file anywhere on your hard disk, but not in your project folder (for example in C:\TEMP).

In Eclipse, follow the next steps to import the existing file:

1. In the C/C++ Projects view, right-click on the project `myproject` and select **Import...**

   The Import wizard appears.

2. Select **General » File System**. Click **Next** to continue.

3. In the **From directory** field, type the path to the directory where you saved `existing.c` (for example `C:\TEMP`) and click in the empty white box below.

   The left box shows the file structure of the directory, the right box shows the files located in that directory, similar to the Windows Explorer.

4. In the left box, select the folder `TEMP`.

5. In the right box, select the file `existing.c`.

6. Click **Finish** to finish the wizard and import the file into your project.

The file `existing.c` is copied from its location at `C:\TEMP` into your project folder and is added to your project. It is now visible as a C source file in the C/C++ Projects view. Changes you make to this file, will not affect the original file stored in `C:\TEMP`. Also, removing this file from your project will remove the file also from your project folder, but the original file remains untouched.

Create a file in the project folder

Instead of importing a file, you can create the file `existing.c` with a standard editor outside Eclipse, and store it directly in the `myproject` folder. To add the file to your project:

• In the C/C++ Projects view, right-click on `myproject` and select **Refresh**.

   The file `existing.c` should now be visible as part of your project.

Create a link in the project folder to an existing file

The third way to add a file to your project, is to create a link to an existing file which is stored on a different location:

1. In the C/C++ Projects view, right-click on the project `myproject` and select **New » File from Template**.

   The New File wizard appears.

2. Select the project folder in which to create the link: type the name of your project (`myproject`) or select the project in the box below.

3. In the **File name** field, enter a name for the link, for example `link2existing.c`.

4. Click the **Advanced >>** button.

   Additional options appear on the dialog to let you create a link to an existing file.

5. Enable the option **Link to file in the file system**.

6. Browse to the location where `existing.c` is located, select this file and click the **Open** button.

7. Click **Finish** to finish the wizard and create the link to the file in your project.

Remove a file or link from your project

As we do not need this file for the remainder of this tutorial, we can safely remove it again from the project:

*   In the C/C++ Projects view, right-click on the file `link2existing.c` and select **Delete**.

    The link `link2existing.c` is no longer part of your project and has been removed from your project folder. The original file, however, remains untouched at its original location.

*   In the C/C++ Projects view, right-click on the file `existing.c` and select **Delete**.

    The file `existing.c` is no longer part of your project and has been removed from your project folder.

Be aware that when you remove a file from your project, it always will be removed from its location in the project folder on your hard disk too!

3.4. Editing Files: C/C++ Editor

Editing a file

Enter the following simple C source in your new source document (the code deliberately contains a mistake, which you will correct later on):

```
#include <stdio.h>

int main( void )
{
    printf( "Hello World\n" )    /* <- missing semicolon */
}
```

Note the following:

*   The tab label of the editor view shows an asterisk in front of the file name (`*myfile.c`) to indicate that the file has been modified.

*   The C/C++ editor view uses syntax coloring.

*   The Outline view shows the structure of the file. You can use this view to navigate through (larger) source files easily. Alternatively you can expand the structure of the file in the C/C++ Projects view.

*   Right-clicking in the editor view presents you with a list of menu commands.

*   To receive more help about the editor view, make sure it is active and press **F1**.

Saving and closing a file

To save the file:

• From the **File** menu, select **Save** (Ctrl+S).

The project will be saved.

To close the file:

• From the **File** menu, select **Close Editor** (Ctrl+W).

Eclipse will ask you to save the files that have been modified since the last save.

Notice also the menu commands **Save All** and **Close All Editors** which you can use when you are working with multiple files.

Opening a file in the C/C++ editor

There are several ways to open an existing file. An easy way to open the C source file `myfile.c` directly in the C/C++ editor is:

• In the C/C++ Projects view, double-click on the file name.

Eclipse recognizes the file as a C source file and opens the file in the C/C++ editor.

• Correct the file by entering the missing semicolon. Save and close the file.

Opening a file in a system editor

If you want to open a C source file in an application (editor) outside Eclipse (instead of the built-in C/C++ editor), proceed as follows:

• In the C/C++ Projects view, right-click on the file `myfile.c` and select **Open With » System Editor**.

The file opens in the application that is associated with the file extension `.c`.

3.5. Closing, Opening and Activating a Project

Closing a project

Like files, you can close a complete project. To do so:

1. In the C/C++ Projects view, right-click on the project `myproject` and select **Close Project**.

   If there are unsaved files, the Save Resources dialog appears in which you can choose which modified files need to be saved before closing the project.

2. Select the files you want to be saved and click **OK** to continue.

Any selected unsaved files are saved first, then the project closes. In the C/C++ Projects view the project myproject *is now visible as a closed map.*

Opening a project

To reopen the project again:

• In the C/C++ Projects view, right-click on the project myproject and select **Open Project**.

The project is open for modifications again. You may need to expand the project structure to view its contents.

Activating a project

The project related menu items and buttons act on the currently active project. When you create a new project, the new project automatically becomes the active project. If you want to work with another project, you have to make it active first.

To make a project active:

• In the C/C++ Projects view, right-click on the project myproject and select **Set Active Project**.

The project is now active. This is mentioned after the project name. Project properties, build and debug will act on this active project.

3.6. Copy a Project

If you want to use a project as a starting point for a new project, you can make a copy of a project. Not only a copy of a project is made, but also the project specific filenames and settings will reflect the new project name.

To copy a project:

1. In the C/C++ Projects view, right-click on the project myproject and select **Copy**.

2. In the C/C++ Projects view, right-click on the project myproject and select **Paste**.

    The Copy Project dialog appears.

3. Enter a name for your new project, for example Copy of myproject.

    *In the **Location** field you will see the location where the copy of your project will be stored. To change the default location, you can uncheck the **Use default location** check box and browse for an alternative location. However, use the default location for now.*

4. Click **Copy** to continue.

    Eclipse makes a copy of the project, renames the LSL file and updates the project configuration and launch configurations. In the C/C++ Projects view the project Copy of myproject *is now visible as a closed map and is the active project.*

3.7. Configuring the Target

In order to build your application, run it and debug it, your project needs information about the target execution environment. The target can be a simulator or an evaluation board. If you are using the TASKING simulator you do not need to configure a target, you only need a launch configuration to start the debugger.

If you are using an evaluation board you have to create a launch configuration for your board in order to debug on a board. For all boards you also need to import a board configuration into your project. Based on your selections your project settings are adjusted, such as memory settings, the linker script file (added LSL statements have the tag "dtc") and startup code.

> The steps below are only necessary if you have not configured the target when you created a project with the **New C/C++ Project** wizard, or if you want to create another configuration.
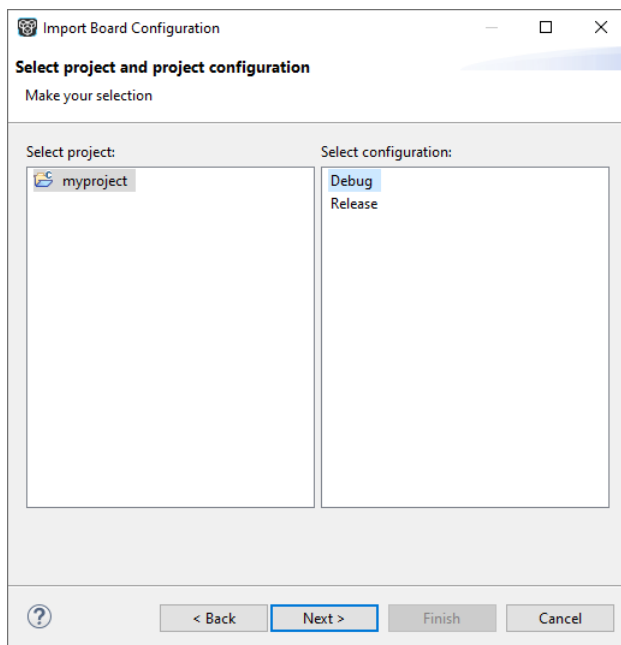
Import TriCore board configuration

1. From the **File** menu in Eclipse, select **Import**.

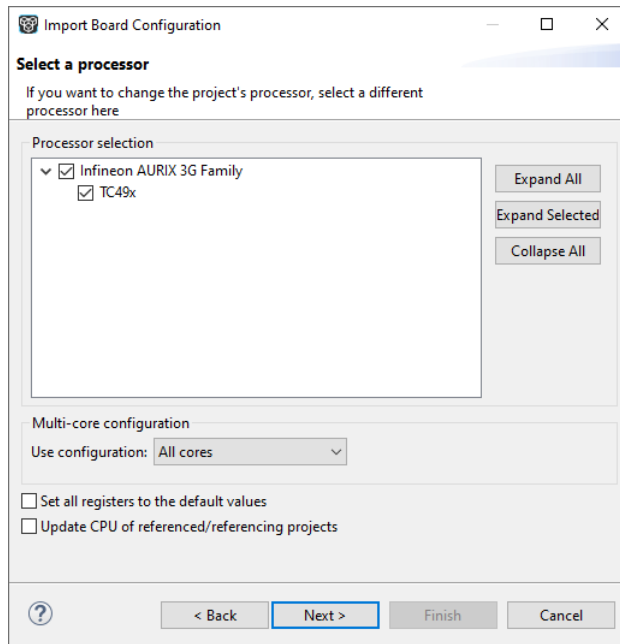   *The Import wizard appears.*

2. Expand **TASKING C/C++**, select **Board Configuration** and click **Next**.

   The Import Board Configuration page appears.

3. Select your project (myproject) and project configuration (Debug) and click **Next**.

The Select a processor page appears.

4. If you want to change the project's processor, select a different processor.

5. If you changed the processor, you can choose to reset the register values and update the processor selection of related projects.

    • Enable **Set all registers to the default values** if you want to make sure that you start debugging with the default register values.

    • Enable **Update CPU of referenced/referencing projects** if you want to make sure that all projects related to your active project have the same processor selected. This check box is only visible when there is a project reference.

6. Click **Next**.

    The Select a target and configuration page appears.
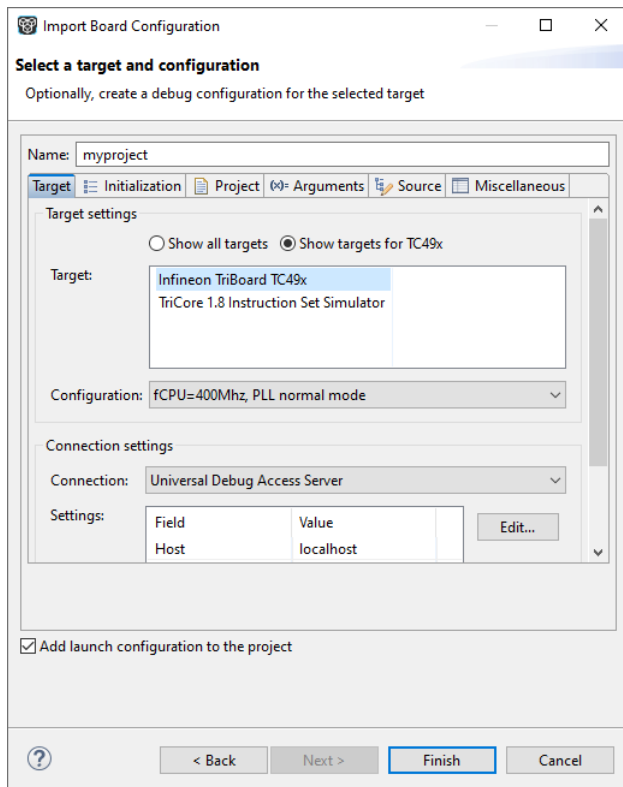
7. In the **Target** field, select the target evaluation board that you use to debug your application. By default only the boards are shown for the selected processor.

8. In the **Configuration** field, select the configuration that matches the settings on your board.

9. Enable **Add launch configuration to the project**. This allows you to debug your project.

    For more information, see section *Creating a Customized Debug Configuration* in Chapter *Using the Debugger* of the *TASKING SmartCode - TriCore User Guide*.

10. Click the **Finish** button to import the configuration settings.

    Your project settings, such as memory, flash settings and LSL file (myproject.lsl) and startup code (cstart.h) are adjusted to the selected board configuration for you to build your application. Note that only those registers are changed that are needed for the board to operate.

The information in the Import Board Configuration wizard is based on Debug Target Configuration (DTC) files. DTC files define all possible configurations for a debug target. The files are located in the `etc` directory of the installed product and use `.dtc` as filename suffix. For more information on DTC files, see the *TASKING SmartCode - TriCore User Guide*.

3.8. Setting Project Options

Now you are familiar with opening and editing (files in) your project, and you have selected a target configuration, we will have a look at the options you can set for building your project.

First make sure the project myproject is open.
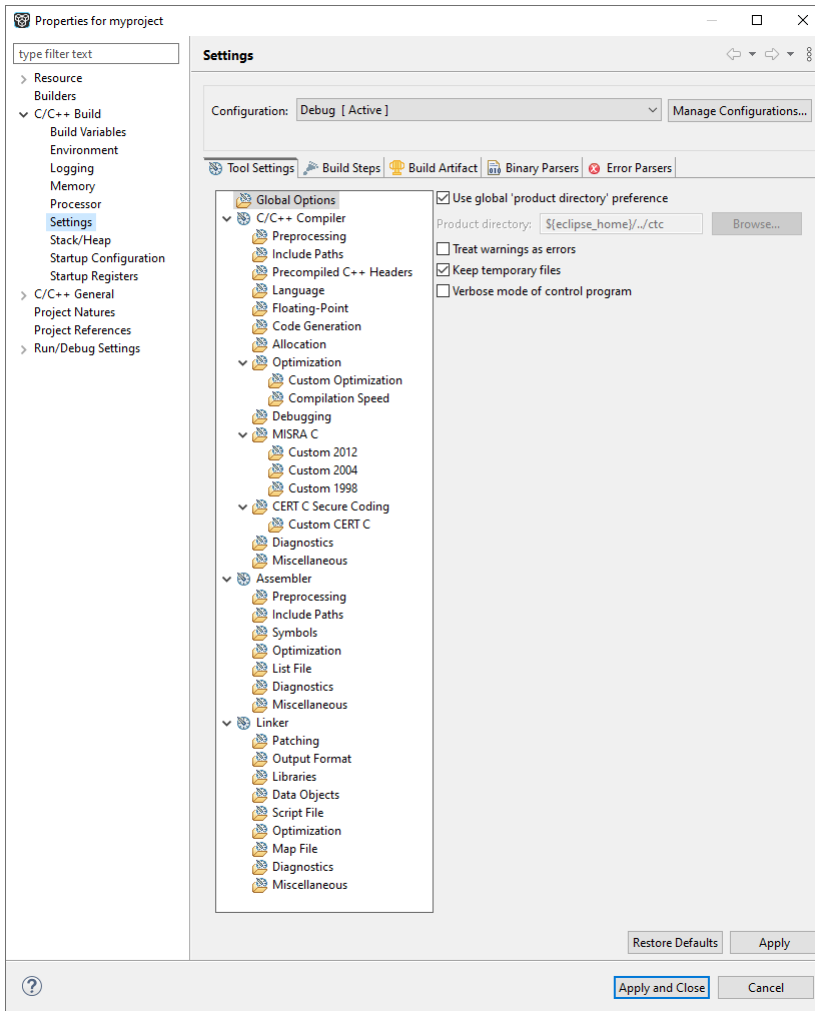
To access the options for your project:

1. From the **Project** menu, select **Properties for**. Alternatively, you can click the 🔧 button.

    The Properties for myproject dialog appears.

2. If not selected, expand **C/C++ Build** and select **Settings** to access the TriCore tool settings.

    A screen similar to the following should now appear.

On the **Tool Settings** tab, the options are grouped in Global Options, C/C++ Compiler, Assembler and Linker or Archiver if you are building a library. Note that the options you enter in the Assembler page are not only used for hand-coded assembly files, but also for the assembly files generated by the compiler. In the **Configuration** field, you can choose a configuration for which you want to make changes. Note that this does not make the configuration active.

For a detailed description of all TriCore toolset options refer to the *TASKING SmartCode - TriCore User Guide.*

Selecting a predefined build configuration

A build configuration is a predefined set of options. When you created the sample project `myproject` as described in Section 3.1, *Create a Project*, you should be able to choose between the **Debug** and the **Release** configuration. Both have their own settings. Check this for your self:

1. Select the **Release** configuration.

2. Expand the **C/C++ Compiler** entry and select **Debugging**.

    *The option **Generate symbolic debug information** is set to **None**.*

3. Select the **Debug** configuration.

    *The option **Generate symbolic debug information** is set to **Default**.*

Setting options and restoring defaults

You can use one of the available configurations as starting point for setting your options. For now, choose the Debug configuration.

1. Change the option **Default** to **Full**.

    At this point you can change as many options as you like.

2. Click **Apply** to apply the new setting(s) to your project.

    The dialog does not close, but the new options are saved to the Debug configuration.

To restore to the default Debug configuration options:

1. Click the **Restore Defaults** button.

    The option settings are changed to the default settings of the chosen configuration.

2. Click **Apply** to apply the default settings to your project.

If you change options without applying them and you try to change the configuration, you are asked whether to apply the changes first.

Creating your own build configuration

Because of the amount of possible options, it may be very convenient to create your own build configuration.

1. Click on the **Manage Configurations...** button next to the **Configuration** field.

    The Manage Configurations dialog appears.

2. Click on the **New...** button.

    The Create New Configuration dialog appears.

3. Type a **Name** (`Myconfig`) and optional a **Description** for your configuration.

    *In the **Copy settings from** box, you can choose the initial option settings for your configuration:*

4. Select **Existing configuration** and choose the **Debug** configuration.

The existing Debug configuration is the same as the default Debug configuration because we applied the default settings in the previous example.

5. Click **OK**.

    The Manage Configurations dialog shows the new configuration.

6. Select the new configuration (`Myconfig`) and click **Set Active**.

7. Click **OK**.

Your new configuration has become the active configuration. From now on, a build will use the option settings from the `Myconfig` configuration. Note that when you select a configuration from the **Configuration** field, this only affects the property pages; it does not make the configuration active.

Important: the **Restore Defaults** button is still associated with the default Debug configuration! Because the new configuration `Myconfig` is based on the Debug configuration, the defaults of the Debug configuration also apply to the `Myconfig` configuration.

Creating your own defaults

The previous example showed how to create your own build configuration to store settings. However, it was impossible to return to your own defaults, only the original **Debug** and/or original **Release** defaults were available. Below it is described how you can create your own defaults. Basically, you create a configuration A in which you set your own defaults; then you create a new configuration B which will be based on configuration A:

If you have found a satisfying combination of option settings, you can create a configuration named `Mydefaults`.

1. First change the option settings to your own needs.

2. Repeat steps 1 through 7 of *Creating your own build configuration* but in step 3, type the name `Mydefaults`.

Normally, any settings you change from here, are saved to `Mydefaults`, thus loosing your original defaults. To prevent this:

1. Repeat steps 1 through 3 of *Creating your own build configuration*, to create a second new configuration and name it `Myworkoptions`. The name suggests that this will be the configuration for experimentally changing option settings.

2. Select **Existing configuration** and choose the **Mydefaults** configuration.

    Your new 'working' configuration is now the same as the configuration named Mydefaults.

3. Click **OK**.

    The Manage Configurations dialog shows the new configuration.

4. Select the new configuration (`Myworkoptions`) and click **Set Active**.

5.   Click **OK**.

Now you can work with the `Myworkoptions` configuration. If you want to return to your defaults, you can either make the `Mydefaults` configuration active, or create a new configuration using the `Mydefaults` configuration to copy the settings from.

3.9. Build a Project

When you build a TASKING TriCore C/C++ project in Eclipse, the TASKING TriCore C/C++ compiler, assembler and linker are used to compile and link all the source code and the libraries associated with the project.

To build a project:

• From the **Project** menu, select **Build myproject**.

From the **Project** menu, the following "Build" commands are available:

Build Project	Builds the selected project.
Build Working Set »	Opens a wizard in which you can create a customized set of files that will be built.
Clean...	Removes all intermediate files that are created during a build. As a consequence, the next build cannot rely on existing results from previous builds (thus simulating a rebuild).
Build Automatically	If you set this option, the selected project will be built automatically after each applied change in the project properties and after each saved change in the source files. This way of building is not recommended for C/C++ development. In order for this option to work, you must also enable option **Build on resource save (Auto build)** on the **Behavior** tab of the **C/C++ Build** page of the **Project » Properties for** dialog.
Build *project*	Builds the active project.
Rebuild *project*	Rebuilds the active project. This builds every file in the project whether or not a file has been modified since the last build. A rebuild is a clean followed by a build.

3.10. Refer to Another Project from a TriCore Project

To tell a TriCore project that another project (a TriCore single-core project, MCS project, 8051 project, ARC project or MIL library project) should be part of it, you need to create a project reference:

1.   In the C/C++ Projects view, right-click on the name of a TriCore project and select **Properties**.

     The Properties dialog appears.

2.   In the left pane, select **Project References**.

3. In the right pane, select the project that must be part of the TriCore project and click **Apply and Close**.

3.11. Using the Sample Projects

TASKING SmartCode comes with a number of examples (delivered in the directory `<installation path>\<toolset>\examples`) for each toolset. Each directory contains a file `readme.txt` with information about the example.
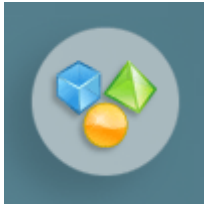
You can import the TriCore examples via the Welcome page. This is an alternative for importing existing projects via the **File » Import » TASKING C/C++ » TASKING TriCore Example Projects** wizard.

Import an existing project from the Welcome page

1. From the **Help** menu, select **Welcome**.

   The Welcome page appears.

2. Click the following button:

   The Samples page appears.

3. Click **TriCore examples**.

   The Import dialog appears.

4. Select the TriCore examples you want to import into the current workspace.

5. Click **Finish**

   The original examples are copied into the current workspace.

   The project(s) should now be visible in the C/C++ Projects view.

You can set additional project options and build the sample projects as explained in the previous sections.

Note that you can also use the Import Board Configuration wizard to configure the examples for an evaluation board.

3.12. Import/Export Project Properties

You can export project properties into a file (`.prop`), so that you can import a specific configuration into a project whenever you want (for example in another workspace).

Export project properties

1. From the **File** menu, select **Export**.

    The Export dialog appears.

2. Select **TASKING C/C++ » TASKING C/C++ Project Properties** and click **Next**.

    The Export TASKING C/C++ Project Properties dialog appears.

3. Select the project and configuration from which you want to export the project properties.

4. Specify the destination properties file (extension `.prop`) and click **Finish**.

    The properties will be saved in the specified file.

Import project properties

1. From the **File** menu, select **Import**.

    The Import dialog appears.

2. Select **TASKING C/C++ » TASKING C/C++ Project Properties** and click **Next**.

    The Import TASKING C/C++ Project Properties dialog appears.

3. Specify the properties file (extension `.prop`) you want to import.

4. Select the destination project and configuration into which you want to import the project properties and click **Finish**.

    The properties of the selected project will be replaced by the properties from the selected file.

Chapter 4. Debugging your Application

Before you start with this chapter, it is recommended to read the Eclipse documentation first. It provides general information about the debugging process. This chapter guides you through a number of examples using the TASKING debugger with simulation as target.

You can find the Eclipse documentation as follows:

1. Start Eclipse.

2. From the **Help** menu, select **Help Contents**.

    The help screen appears.

3. In the left pane, select **C/C++ Development User Guide**.

4. Open the **Getting Started** entry and select **Debugging projects**.

    This Eclipse tutorial provides an overview of the debugging process. Be aware that the Eclipse example does not use the TASKING tools and TASKING debugger.

4.1. Setting up a Project for Debugging

4.1.1. Create a Sample Project

1. Create or reopen the project `myproject` as created in Section 3.1, *Create a Project*. Use the default values and make sure that you:

    *   select **Hello World C project** in the New C/C++ Project wizard.

    *   enable at least the **Debug** configuration.

2. Edit the file `myproject.c` as follows:

```
#include <stdio.h>

int main( void )
{
    int i;
    for (i=1; i<=3; i++)
    {
        printf( "%d\n",i );
    }
    printf( "Hello world, " );
    printf( "this is \n" );
    printf( "a small %dst\n",i-3 );
    printf( "debugging example.\n" );
}
```

3. Save the file.

4. Build your project.

To be able to debug, it is essential that your project has been built properly!

All steps required above are demonstrated in Chapter 3, *Setting up a Project*.

Debugging an 8051 project

In order to debug an 8051 project, follow the steps below.

1. Create an 8051 project (for example, `myproject`), as explained above.

2. (Optional) Build the 8051 project. This step is optional because the `.out` file is built automatically when the project is referenced and built from a TriCore project. See step 5.

    This results in a linked output file (`.out`).

3. Create a TriCore project.

4. In the TriCore project, make a project reference to the 8051 project.

5. Build the TriCore project.

    This builds the referenced 8051 project and creates the `.out` file in the Debug directory of the 8051 project. Furthermore this creates a TriCore ELF file and an 8051 ELF file in the Debug directory of the TriCore project.

6. Make the 8051 project the active project.

7. Create a debug configuration for the 8051 project, as explained in Section 4.1.2, *Create a Debug Configuration*.

8. Start the debugger, as explained in Section 4.2, *Start a Debug Session*.

If you want to debug with the 8051 simulator, you can add an extra post link step to build a standalone 8051 project and create an absolute ELF file that you can debug. To do this:

1. From the **Project** menu, select **Properties for myproject**.

    The Properties dialog appears.

2. Select **C/C++ Build » Settings**.

3. Open the **Build Steps** tab.

4. Add the following command line to the **Command** field under **Post-build steps**:

    ```
    "${PRODDIR}/bin/lk51" -dtc49x.lsl -M -o ${PROJ}.elf ${PROJ}.out --core=mpe:xc800
    ```

5.   Click **Apply and Close**.

Debugging an ARC project

In order to debug an ARC project, follow the steps below.

1.   Make sure the Synopsys ARC® nSIM Instruction Set Simulator is present. This simulator is not part of the SmartCode product, you need to order it separately from Synopsys®. Then copy the 64-bit Windows file libsim.dll to the following directory:

```
<installation-dir>\eclipse\plugins\com.tasking.arc.debug.win32.x86_64_1.4.0.0\carc\bin
```

2.   Create an ARC project (for example, myproject), as explained above.

3.   Build the ARC project.

This results in an ELF file (.elf).

4.   Create a debug configuration for the ARC project with the Synopsys nSIM Simulator as target, if you have not already done so when creating the project. See Section 4.1.2, *Create a Debug Configuration*.

5.   Start the debugger, as explained in Section 4.2, *Start a Debug Session*.

Note that when you have a project reference from a TriCore project to the ARC project, the ARC project is built automatically when you build the TriCore project.

4.1.2. Create a Debug Configuration

Before you can debug a project, you need a Debug launch configuration. Such a configuration, identified by a name, contains all information about the debug project: which debugger is used, which project is used, which binary debug file is used, which perspective is used, ... and so forth.

You can create a launch configuration when you create a new project with the New C/C++ Project wizard. In Section 3.1, *Create a Project* we created one for the TASKING simulator. At any time you can change this configuration. If you want to debug on a target board, you have to create a custom debug configuration for your target board.

For details on creating or changing a debug configuration, refer to section *Creating a Customized Debug Configuration* in Chapter *Using the Debugger* of the *TASKING SmartCode - TriCore User Guide*.

4.1.3. Setting TASKING winIDEA Preferences

In order to select the winIDEA installation of your choice, follow the steps below.

1.   From the **Window** menu, select **Preferences**.

The Preferences dialog appears.

2.   Select **TASKING winIDEA**.

3. Select your preference for the desired winIDEA setup. Select **Use embedded installation** to use the winIDEA that is already embedded in the product, or select **Use this installation** to use the winIDEA installation in your system.

4. Click **Apply and Close**.

4.2. Start a Debug Session

1. From the **Debug** menu select **Debug** *project*.

    Alternatively you can click the 🔆 button in the main toolbar.

    The TASKING Debug perspective is associated with the TASKING C/C++ Debugger. Because the TASKING C/C++ perspective is still active, Eclipse asks to open the TASKING Debug perspective.

    Please note that the TASKING winIDEA session can be viewed by switching to the CDT Debug perspective.

2. Optionally, enable the option **Remember my decision** and click **Switch**.

    The debug session is launched. This may take a few seconds.
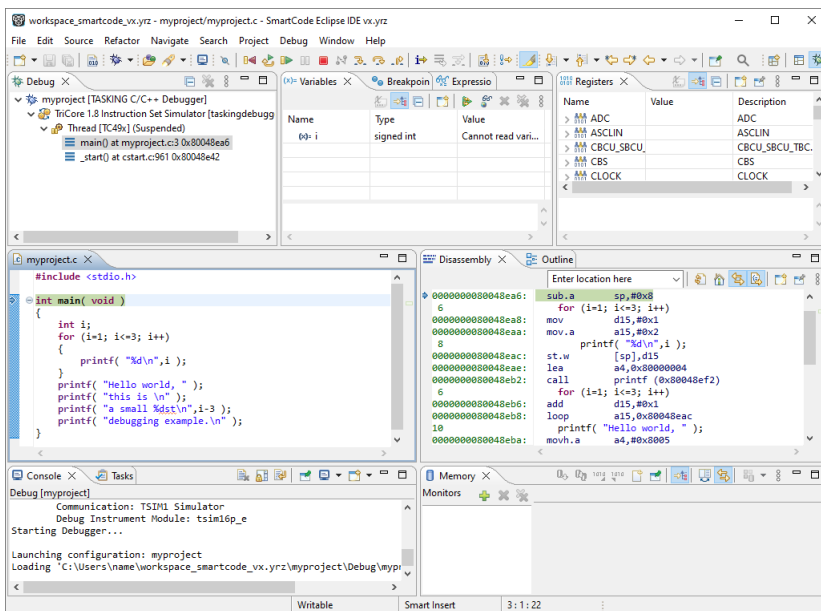
*   The Debug view shows your running application. Because of the settings in the debug configuration, execution has suspended at the first instruction in the function `main()`.

*   The Editor view shows the C source files of your application and shows the line where the execution has suspended.

- The Variables view shows the variables in your application; in this case `int i`.

4.3. Stepping through the Application

At this moment your application is executing but suspended on the function `main()`. This means the C startup code has been executed already. From this point, you can step through your application while inspecting what happens.

1. From the **Debug** menu, select **Step Over**, or press **F6**, or click on the **Step Over** button (🔄) in the Debug view.

   The highlight in the Edit view moves to the next statement.

2. Press **F6** again.

   The highlight in the Edit view moves to the next statement.

   In the Variables view, you can inspect the value of the variable `i`. *It is now set to 1.*

3. Press **F6** again.

   The `printf` *statement has been executed now. The bottom area of your workbench now shows a new view: FSS # 1 - myproject.*

   FSS stands for *File System Simulation.* The FSS view simulates the input and output to and from the target board or simulator when you are debugging. The value of `int i` is printed and sent to the FSS view for output.

   > To clear the FSS view, right-click in the view and select **Clear**.
   >
   > To restart your application, from the **Debug** menu, select **Restart** (🔄).

4. Step further through your application.

   Watch the value of `int i` *in the Variables view and observe the output in the FSS view. The output is only flushed after a newline (*`\n`*)!*

> When you debug your application in an interrupt enabled environment, it might be useful to enable
> **Interrupt aware stepping** (🔄). This prevents stepping into an interrupt handler when an interrupt
> occurs.

4.4. Setting and Removing Breakpoints

Instead of stepping, you can set breakpoints to suspended the application at a certain point.

A breakpoint is set on an executable line of a program. If a breakpoint is enabled during debugging, the execution suspends *before* that line of code executes.

Add breakpoints

To add a breakpoint:

* Double-click the marker bar located in the left margin of the C/C++ Editor next to the line of code where you want to add a breakpoint.

  A dot ● is displayed in the marker bar and in the Breakpoints view, along with the name of the associated file. When the breakpoint is actually set, a check mark ✔ appears in front of the dot.

Disable breakpoints

You can disable a breakpoint or completely remove it. To disable a breakpoint, do one of the following:

* In the Breakpoints view, disable a breakpoint by clearing the check box.

* In the Editor view, right-click on a breakpoint dot in the margin and select **Disable Breakpoint**.

  The blue breakpoint dot turns white.

Remove breakpoints

To completely remove the breakpoint, do one of the following:

* In the Breakpoints view, right-click on a breakpoint and select **Remove**.

* In the Editor view, right-click on a breakpoint dot in the margin and select **Toggle Breakpoint**.

* In the Editor view, double-click on a breakpoint.

  The blue breakpoint dot disappears.

Example

With the techniques described above:

1. Set a line breakpoint on the code line `printf( "a small %dst\n",i-3 );`.

2. Clear the FSS view.

3. Restart your application.

   The application suspends when entering the main() function because this was defined in the Debug configuration.

4. To resume execution, from the **Debug** menu, select **Resume**, or press **F8**, or click on the **Resume** button (▶).

The application suspends execution, before this line is executed. The FSS view now shows:

```
1
2
3
Hello world, this is
```

5.   Resume execution again to finish execution.

Note that though the application has finished execution, it has not been terminated yet. Your debug session is still active.

4.5. Reload Current Application

When your application had changed, for example because you solved a bug, you can reload the application in the debugger without restarting it.

1.   Make the necessary changes in your source.

2.   Rebuild your application ().

3.   Click on the **Reload current application** button ().

The new application is loaded in the debugger.

You can also instruct the debugger to reload the application automatically after a rebuild of the application.

1.   From the **Window** menu, select **Preferences**.

The Preferences dialog appears.

2.   Select **TASKING » Debugger Miscellaneous**.

3.   Select your preference for **Re-download after rebuilding project**. Select **Always** to always download the new built application, or select **Never**, or select **Prompt** to get a question each time after rebuilding the project.

4.   Click **Apply and Close**.

4.6. End a Debug Session

To end the debug session:

1.   From the **Debug** menu select **Terminate** or click on the **Terminate** button ().

2.   To remove the debug session from the Debug view, right-click on the debug session and select **Remove All Terminated** or click on the **Remove All Terminated Launches** button () in the Debug view.

4.7. Multiple Debug Sessions

It is possible to run multiple debug sessions. To do so, just repeat the steps for starting a debug session. First make sure that you have terminated all debug sessions.

1. From the **Window** menu, select **Preferences**.

   The Preferences dialog appears.

2. Select **TASKING » Debugger Start-up**.

3. Enable the option **Allow multiple simultaneous debug sessions**.

4. Select what you want to happen **When trying to start another debug session for the same configuration**. Select **Re-download** to download the absolute file again, or select **Start new session**, or select **Prompt** to get a question each time you try to start a new session.

5. Click **Apply and Close**.

6. From the **Debug** menu, select **Debug Configurations...**

   The Debug Configurations dialog appears.

7. Select the debug configuration `myproject.simulator` and click on the **Debug** button.

   The debug session launches.

8. Repeat steps 1 and 2, but in step 2 choose `myproject.board`.

There are now two debug sessions for the same application. In case you have multiple projects, you can make dedicated debug configurations for them. You can use these debug configurations to run multiple debug sessions at the same time.

Each session uses its own FSS view for output. In the Debug view you can select the debug session (or file in the debug session) for which you want to inspect, for example, the value of its variables in the Variables view.