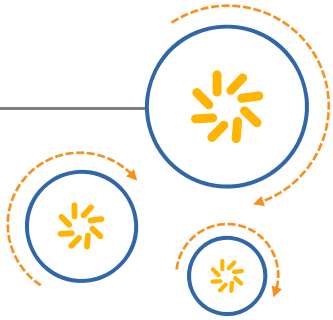
Qualcomm Technologies International, Ltd.



Confidential and Proprietary – Qualcomm Technologies International, Ltd.

(formerly known as Cambridge Silicon Radio Ltd.)

NO PUBLIC DISCLOSURE PERMITTED: Please report postings of this document on public servers or websites to: DocCtrlAgent@qualcomm.com.

Restricted Distribution: Not to be distributed to anyone who is not an employee of either Qualcomm Technologies International, Ltd.or its affiliated companies without the express approval of Qualcomm Configuration Management.

Not to be used, copied, reproduced, or modified in whole or in part, nor its contents revealed in any manner to others without the express written permission of Qualcomm Technologies International, Ltd.

Any software provided with this notice is governed by the Qualcomm Technologies International, Ltd. Terms of Supply or the applicable license agreement at https://www.csrsupport.com/CSRTermsandConditions.

Qualcomm is a trademark of Qualcomm Incorporated, registered in the United States and other countries. All Qualcomm Incorporated trademarks are used with permission. Other product and brand names may be trademarks or registered trademarks of their respective owners.

This technical data may be subject to U.S. and international export, re-export, or transfer (“export”) laws. Diversion contrary to U.S. and international law is strictly prohibited.

Qualcomm Technologies International, Ltd.
Churchill House
Cambridge Business Park
Cambridge, CB4 0WZ
United Kingdom

Push every boundary.®

CSR µEnergy®

xIDE

User Guide

Issue 12

Document History

Revision	Date	History
1	07 MAR 11	Original publication of this document
2	20 JUL 11	Editorial updates
3	20 MAR 12	Updated for v1.4
4	22 MAR 12	Correction to section 3.1.3
5	20 SEP 12	Updated for v2.0.0
6	10 JAN 13	Updated for v2.1.0
7	17 APR 13	Updated for Windows 8 support
8	11 NOV 13	Updated to new CSR branding
9	24 JAN 14	Updated page 3
10	18 JUN 14	Updated for v2.4.0; removed reference to Windows XP and CSR100x devices
11	05 JAN 15	Updated for v2.4.4
12	03 AUG 15	Updated for v2.5.0

Contacts

General information	www.csr.com
Information on this product	sales@csr.com
Customer support for this product	www.csrsupport.com
More detail on compliance and standards	product.compliance@csr.com
Help with this document	comments@csr.com

Trademarks, Patents and Licences

Unless otherwise stated, words and logos marked with ™ or ® are trademarks registered or owned by CSR plc and/or its affiliates.

Bluetooth® and the Bluetooth logos are trademarks owned by Bluetooth SIG, Inc. and licensed to CSR.

Other products, services and names used in this document may have been trademarked by their respective owners.

The publication of this information does not imply that any licence is granted under any patent or other rights owned by CSR plc or its affiliates.

CSR reserves the right to make technical changes to its products as part of its development programme.

While every care has been taken to ensure the accuracy of the contents of this document, CSR cannot accept responsibility for any errors.

Use of this document is permissible only in accordance with the applicable CSR licence agreement.

Safety-critical Applications

CSR's products are not designed for use in safety critical devices or systems such as those relating to: (i) life support; (ii) nuclear power; and/or (iii) civil aviation applications, or other applications where injury or loss of life could be reasonably foreseeable as a result of the failure of a product. The customer agrees not to use CSR's products (or supply CSR's products for use) in such devices or systems.

Performance and Conformance

Refer to www.csrsupport.com for compliance and conformance to standards information.

Contents

Tables, Figures and Equations

1. Introduction

This document provides a brief introduction to CSR's Integrated Development Environment (xIDE) supplied with CSR µEnergy Software Development Kits (SDKs).

The document is intended to provide developers with the information required to begin using xIDE to develop applications for CSR's Bluetooth Smart single-mode IC devices.

Note:

Since xIDE provides a familiar environment with the tools and utilities required to write, build, run and debug code it is not intended to detail all these features. This document concentrates on CSR µEnergy-specific aspects of developing applications using xIDE.

1.1. General

xIDE allows software engineers to build and configure the application projects provided in the SDKs and to independently develop applications to run on CSR µEnergy devices.

xIDE supports the development and debugging of applications written in ANSI C for the XAP core, and assembly language for the 8051 PIO Controller, present on the device.

Code is written in the text editor and, once complete, built and compiled along with the firmware supplied with the SDK. The resultant machine code can be downloaded to, run and debugged on a real hardware development platform such as those included in the CSR µEnergy development kits.

Example code is provided that implements the latest Bluetooth Smart Profiles and on-chip peripheral functionality and when used with firmware library functions enable developers to rapidly create their working Bluetooth Smart device applications with minimal effort.

Note:

The supplied profile code usually supports all the mandatory features and most optional features of a particular profile. See the SDK Release Note and Application Notes provided with each profile-based example application for details.

2. Installation

This section gives guidance on the installation of xIDE for CSR µEnergy.

2.1. Prerequisites

xIDE may be installed on a PC running:

- Windows 7 (32-bit or 64-bit)
- Windows 8 (32-bit or 64-bit).

xIDE requires a USB port or an LPT port to communicate with development boards. The LPT port is only supported when xIDE is running on 32-bit versions of Windows.

CSR recommends that 300 Mbytes of free disk space is available. A typical SDK installation requires 210 Mbytes and each application built will need approximately 6 Mbytes of additional space.

A Windows Administrator or Standard User account is required to install the software correctly. If you are unsure of your current level of privileges, please contact your system administrator.

New installations can coexist with previous releases.

Notes:

1. Installing using a Standard User account requires the Administrator's password to be entered during installation.
2. Spaces in folder names of the directory path are not supported i.e. you should not try to install the software in a directory which itself has spaces in its name or is contained within a folder that has spaces in its name e.g. xIDE cannot be successfully installed in the `Program Files` directory.

2.2. Installation Procedure

CSR recommends that any applications running on the PC are closed before installing the software.

1. The software is provided on a CD-ROM but can also be downloaded from www.csrsupport.com.
2. Double-click on the `CSR_uEnergy_SDK-<Version>.exe` file to launch the Setup wizard, which guides you through the rest of the installation process.
3. Follow the on-screen instructions, clicking **Next** to continue.
   For a first time installation, CSR recommends that the default settings are used.
4. Click **Finish** to complete the installation.
   If the option to install the SPI LPT device driver was selected, the PC must be restarted to complete the installation.

2.2.1. Testing the Completed Installation

2.2.1.1. Before You Begin

Connect a suitable CSR µEnergy hardware development platform, e.g. CSR101x development board, to your PC using the USB to SPI Adapter and cables provided in the CSR µEnergy Development Kit.

Install a terminal client application e.g. PuTTY or Tera Term.

Note:

The Quick Start Guide accompanying the development kit or development board gives further advice on connecting the development hardware to your PC.

2.2.1.2. Testing the Installation

Launch xIDE for CSR µEnergy SDK by double-clicking on the icon on your desktop or from the Windows **Start** menu:

- On Windows 8 open the Start page and click on **CSR µEnergy SDK <version> (xIDE)**.
- On Windows 7 open the Start menu and navigate to **CSR µEnergy SDK <version>/CSR µEnergy SDK (xIDE)**.

The xIDE application window opens, see Figure 2.1:

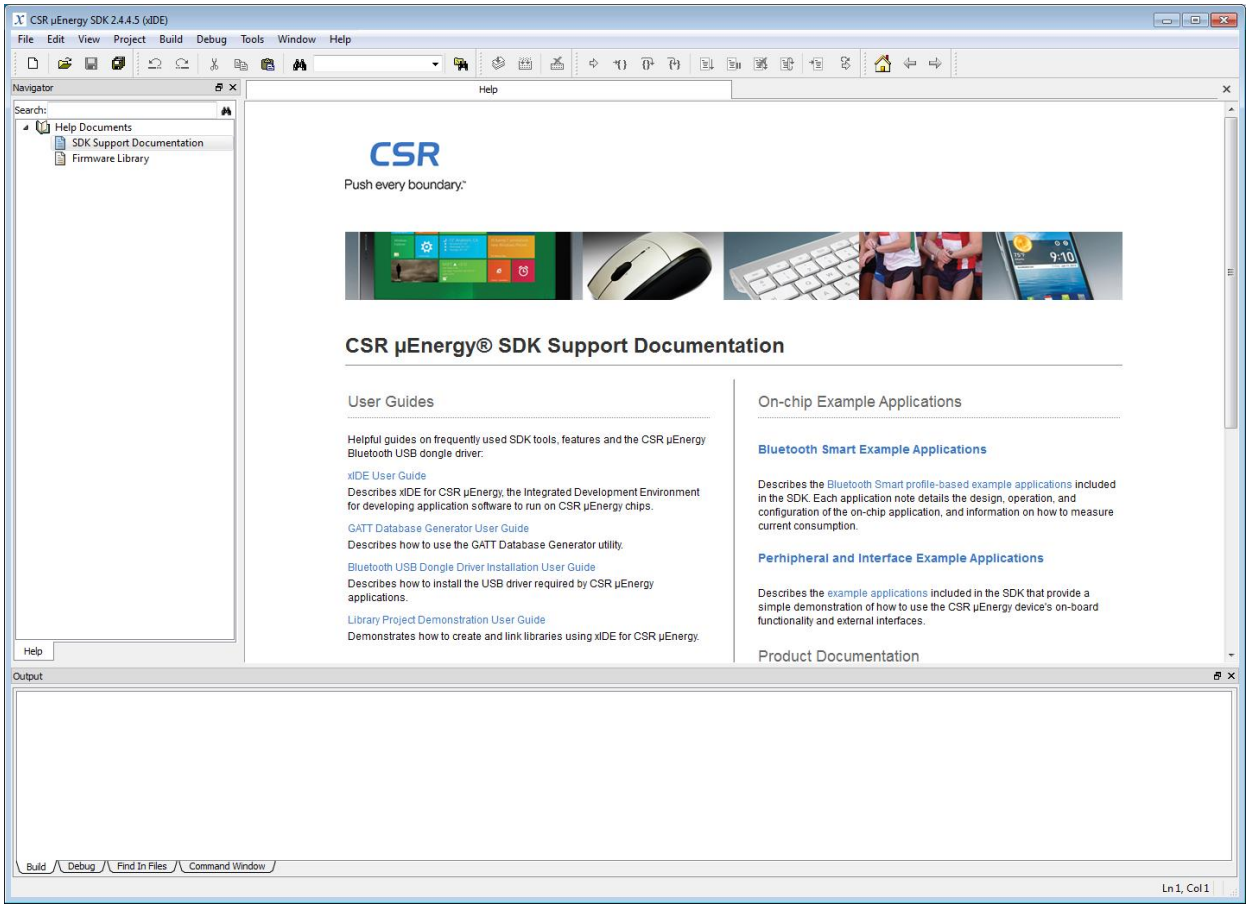


Figure 2.1: Opening Screen

To confirm the installation was successful and the software is working correctly, a new minimal project can be created, see below. When built and downloaded, the on-chip application outputs the “Hello, world” text string to a virtual COM port provided by the CSR µEnergy USB to SPI Adapter.

Use a terminal client application to open the COM port at 2400 baud, 8 data bits, no parity and one stop bit with hardware flow control.

To create a minimal project:

1. Select **New** from the **Project** menu.

   The **New Project** window appears, see Figure 2.2:

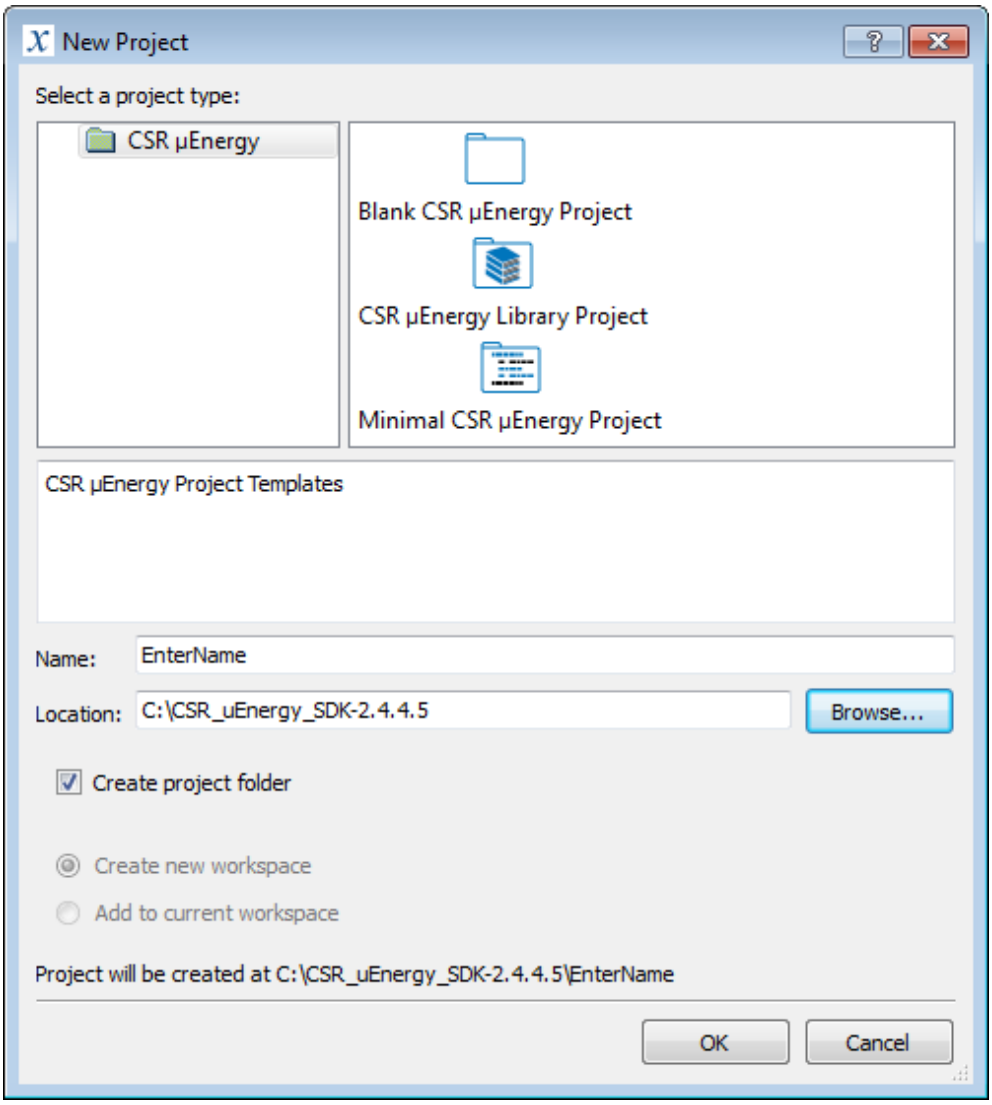


Figure 2.2: New Project

2. Select the **Minimal CSR µEnergy Project** and give the project a name e.g. `hello`.

Note:

xIDE does not accept project names and location paths if they contain spaces.

3. Click **OK**.

   The project is loaded into xIDE

4. Click on the **C Files** folder in the **Navigator** panel and select `main.c` to display the code in the **Text Editor** workspace, see Figure 2.3:

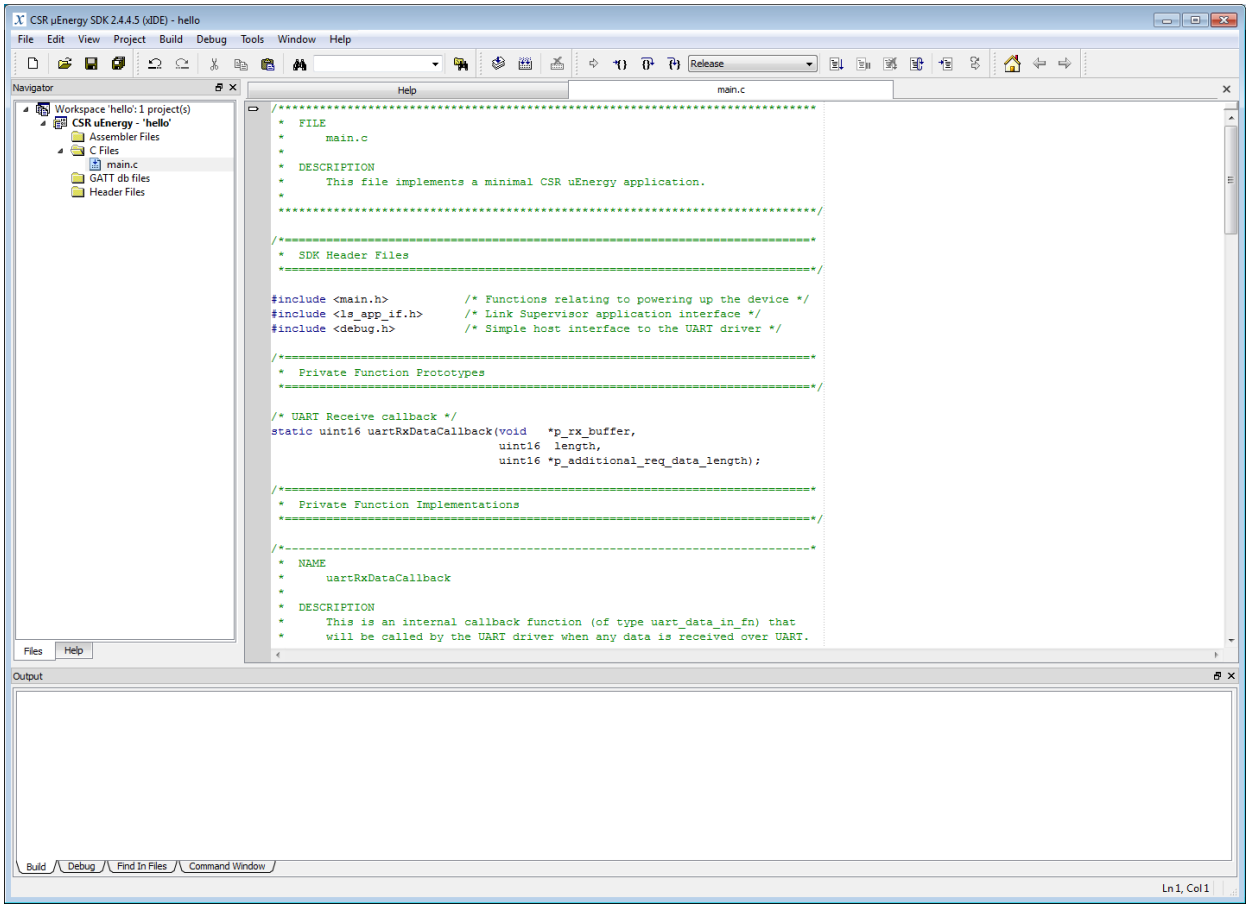


Figure 2.3: Text Editor

5. Ensure your target device is attached and the correct transport is selected by opening the **Debug** menu and selecting the **Transport…** menu option.
6. Select **Build Active Project** from the **Build** menu (or press the **F7** key).
7. Select **Run** from the **Debug** menu (or press the **F5** key).

When this process is complete the application is downloaded to the attached CSR µEnergy device, see Figure 2.4:

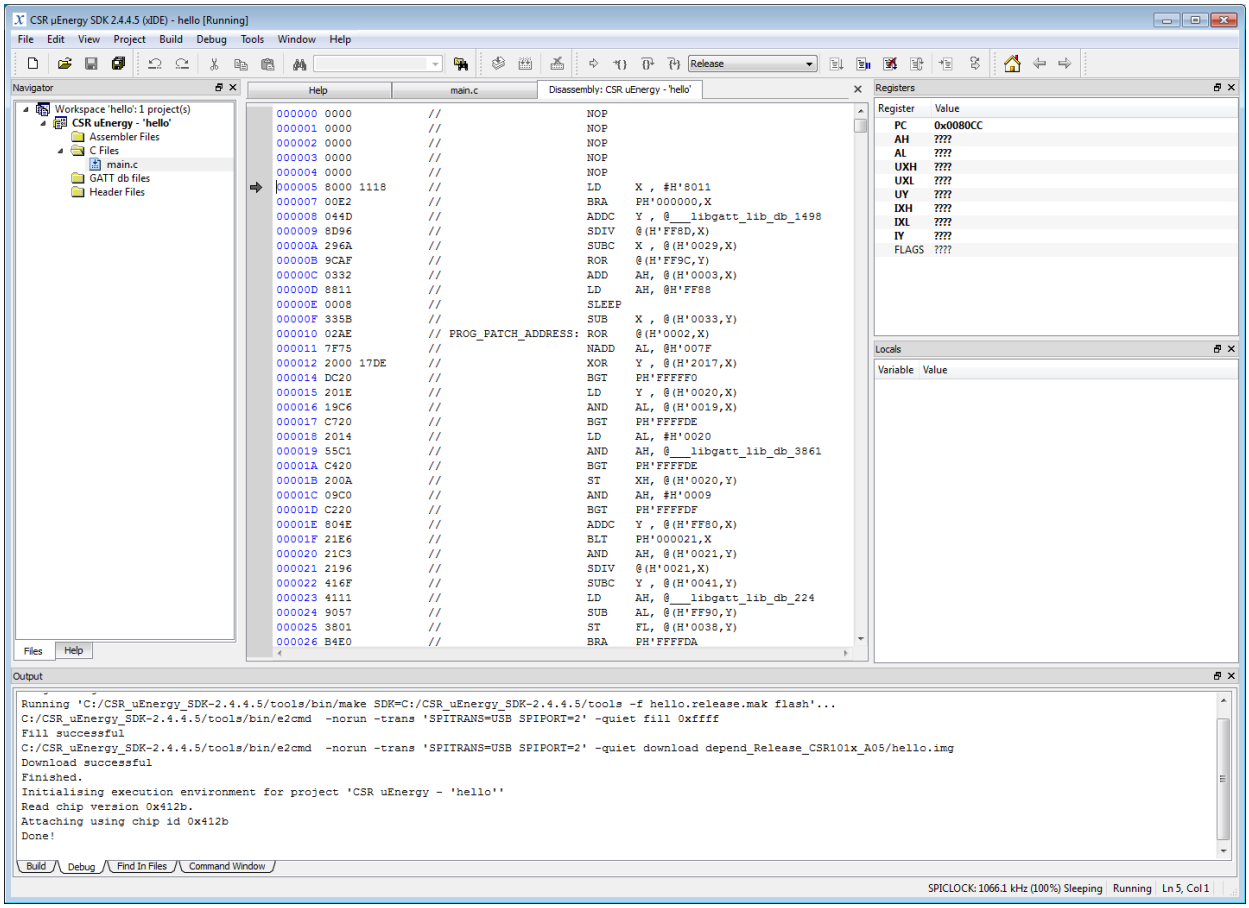


Figure 2.4: Running Project

The output `Hello, world` should be displayed by the terminal client application and confirms that the software has been installed and is working correctly.

3. Working with xIDE

Developers can make use of the reference applications provided as the basis for developing their own applications as these reference applications demonstrate basic functionality and conform to the relevant Bluetooth Smart Profiles.

Adopting this approach greatly reduces the effort required to develop a final product application and allows software engineers to concentrate on developing the additional functionality and Man Machine Interface (MMI) features required for their particular product.

This section describes the procedure for loading a reference application as a project in xIDE and running the code on a hardware development platform. Specific details will vary slightly depending on the application and hardware platform being used, and further information is provided in the relevant product documentation.

Note:

Guidance on the use of device peripheral and profile-based example applications is provided in readme files and application notes within the application subfolders, `C:\<CSR_uEnergy_SDK-Version>\apps\...` where `C:\<CSR_uEnergy_SDK-Version>` is the installation folder.

3.1. Building a Supplied Application Project in xIDE

To open a project workspace for a supplied application:

1. Select **Open Workspace** in the xIDE **Project** menu.
   An **Open workspace** window appears.
2. Browse to the folder containing the example applications provided in the SDK.
   e.g. `C:\<CSR_uEnergy_SDK-Version>\apps`
3. Open the required application folder.
   Depending on the application chosen, one or more `.xiw` project files are displayed, see Figure 3.1:

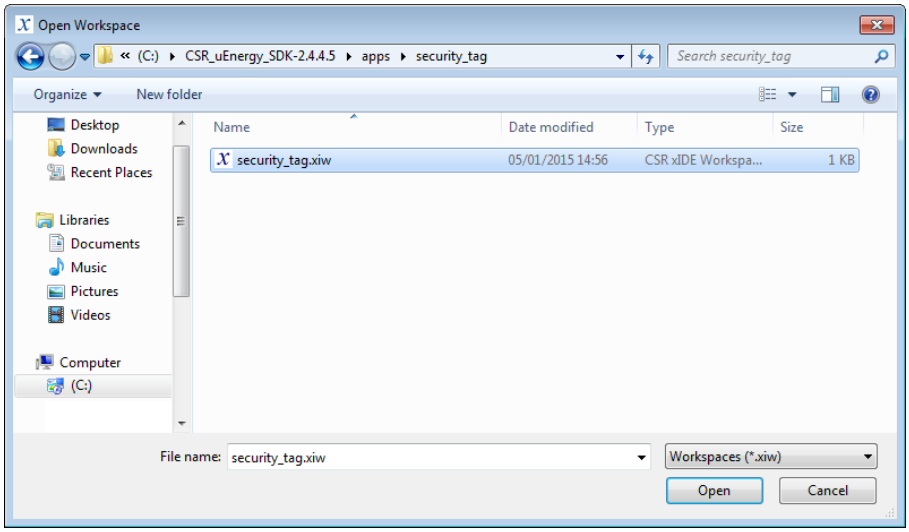


Figure 3.1: Open Workspace

3.1.1. Build Menu and Toolbar Operations

The following operations are provided on the **Build** menu and toolbar:

Compile File 

Compiles the currently selected C code (`.c`), assembly language (`.asm`) or GATT database (`.db`) file.

Note:

If a master database file is present in the project it may refer to additional database files and invoking the **Compile File** operation on the master file compiles the additional database files.

Build Active Project 

Builds all the files included in the Active project from the current Workspace using the selected configuration. The build is incremental, so the minimum set of builds are performed in order to reflect changes to source files and configurations.

Rebuild Active Project

Rebuilds all files included in the Active project from the current Workspace using the selected configuration. All output files are removed and re-built.

Clean Active Project

Deletes all output files created by previous build operations.

Stop Build 

Terminates the build process.

3.1.2. xIDE Build and Run Procedure

3.1.2.1. Build Procedure

To build all the source files making up the application:

1. Select the required `.xiw` file in the Open Workspace window (**Project /Open Workspace**)
2. Click **Open**.
   The file is loaded into xIDE.
3. Select **Build Active Project** from the xIDE **Build** menu or press the **F7** key.
   xIDE builds all the files in the project.

3.1.2.2. Run Procedure

To download the machine code to the device:

With the project loaded in xIDE,

1. Select **Run** from the **Debug** menu or press the **F5** key.

The application should now be running on the device, see the relevant application documentation for further details.

3.1.3. Configuration Store Settings

The project folder may contain a configuration (`.keyr`) file defining the values of configuration keys for a specific type of target hardware. These Configuration Store (CS) settings are downloaded automatically to the target after the combined firmware and application image file has been downloaded.

Multiple CS key files can be added to the project folder but only one will be used for each type of target hardware.

3.1.3.1. Creating the Configuration Store Key File for the First Time

1. Ensure a `.keyr` file is not present in the project folder, has not been added to the project, and is not assigned to the **CS Key File** property configured in the **Properties…** menu option in the **Project** menu.
2. Run the application using xIDE to download the application image and to use the default firmware configuration settings.
3. Launch the **CsConfig** tool from the Windows **Start** menu:
   - On Windows 8 open the Start page and click on **CsConfig**.
   - On Windows 7, open the **Start** menu, and navigate to **CSR µEnergy SDK <version>/CSR µEnergy Tools/CsConfig**
4. Select the **Transport** to connect to the target device (if required).
5. Modify the individual settings before clicking **Save** to apply the settings to the target chip.
6. Open the **Keys** menu and select **Export to file…**.
7. Enter the filename for the `.keyr` file, ensuring the file is saved to the current project folder. For example, use `CSR101x_A05.keyr` for CSR101x A05 devices.
8. Add the new configuration file to the project workspace in xIDE.
9. Using the **Properties…** menu option on the **Project** menu, specify the CS key filename in the **CS Key File** property for each supported device type.

See the **CsConfig** on-line help for more details on its use.

3.1.3.2. Modifying Existing Configuration Store Settings

If the configuration file is already present, edit the configuration file in the **Text Editor** workspace in xIDE and save your changes.

When the application is next built and downloaded, the configuration settings contained in the saved file are used to override the default values provided in the firmware.

Note:

When using new firmware libraries, CSR recommends removing the CS key file from the old project folder and xIDE project workspace before following the steps in section 3.1.3.1. This avoids compatibility issues between the configuration settings in use on a target device and the new firmware.

3.2. Developing Customised Applications

When the application has been downloaded and is working correctly, developers can begin to customise the example source code and add features to meet the specific requirements of the final product.

In order to work efficiently when developing an application it is important to become familiar with the library structure and functions provided. These are detailed in the Firmware Library documentation accessible from the **Help** tab on the Navigator panel, or from the Windows **Start** menu:

- On Windows 8 open the Start page and click on **Firmware Library**.
- On Windows 7 open the **Start** menu and navigate to **CSR µEnergy SDK <version>/Documentation/Firmware Library**.

Note:

A subset of standard library C functions are available as part of the CSR µEnergy Firmware Library.

3.2.1. To Amend the Project Properties

When a project workspace has been opened:

1. Select **Properties** from the **Project** menu.
   The **Project Properties** window appears, see Figure 3.2:

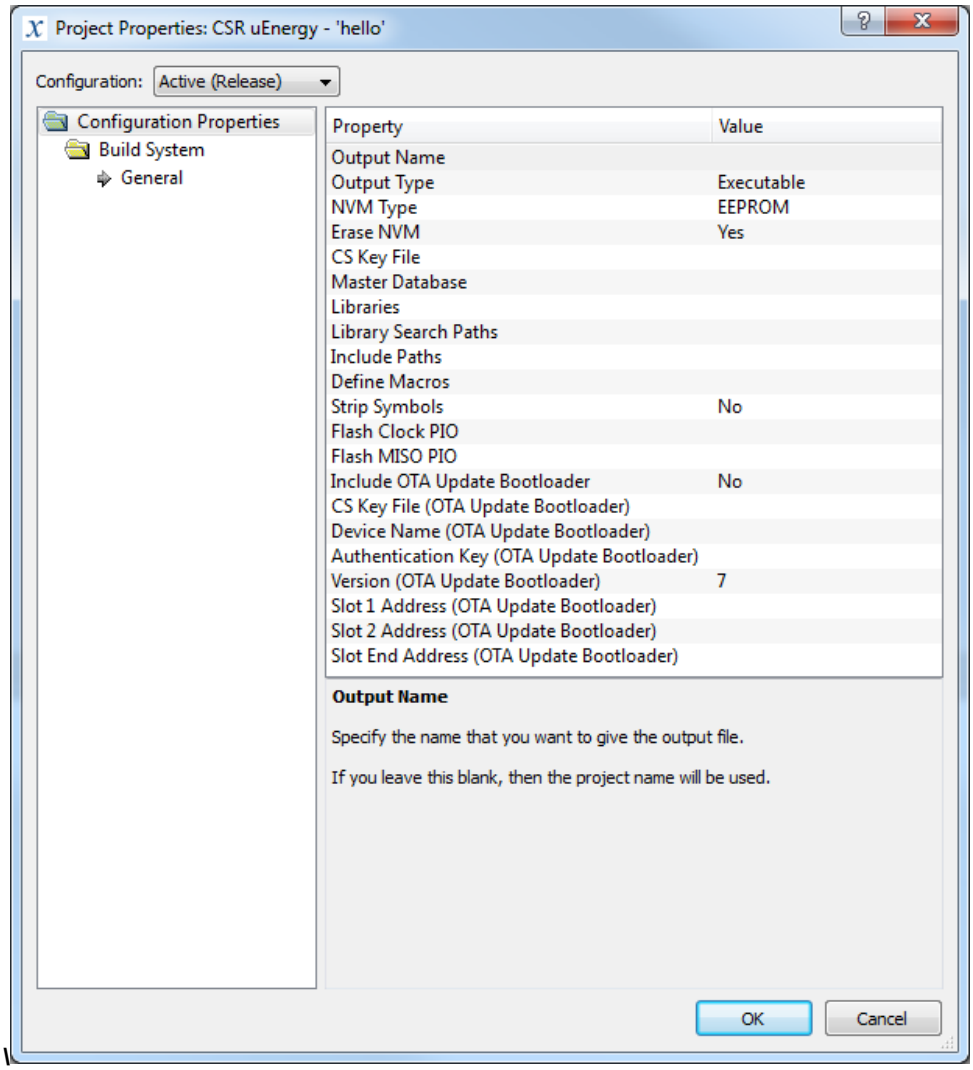


Figure 3.2: Project Properties

2. Select the **Configuration** from the drop-down list.
3. Click on a row to activate the **Value** field for the **Property** you want to amend.
   The text below the list of properties describes the currently selected **Property**.
4. When the required properties have been amended, click **OK** to set the properties for the project.

3.2.2. Project Properties

Various Project Properties can be configured as described in 3.2.1. This section describes each of them and how to be used.

3.2.2.1. Output Name

Specifies the name of the output file. If left blank, then the project name will be used.

3.2.2.2. Output Type

Specifies the type of the output produced by the build. It can be selected from a drop-down menu and have one of the following values:

- **Library**
- **Executable**

3.2.2.3. NVM Type

Specifies the type of NVM the application image will be downloaded to. It can be selected from a drop-down menu and have one of the following values:

- **EEPROM**
- **FLASH**

A macro is defined during compilation to let the application know which NVM type it is being built for. If EEPROM is selected `NVM_TYPE_EEPROM` will be defined, for FLASH the macro `NVM_TYPE_FLASH` will be defined.

3.2.2.4. Erase NVM

Specifies whether to erase the NVM prior to downloading the image. It can be selected from a drop-down menu and have one of the following values:

- **Yes**
- **No**

3.2.2.5. CS Key File

Specifies the filename with extensions for the CS key definition file (`.keyr`). For example:

```
example.keyr
```

Notes:

1. If the file is not present in the project folder, the absolute path or the path relative to the project folder can be used
2. The path must not include a mixture of absolute and relative paths. For example, `C:\folder\example.keyr, .\example.keyr, example.keyr` are allowed, but `C:\folder\debug\..\example.keyr` is not allowed.
3. The specified path and filename are case sensitive
4. The macros `$(HARDWARE_TARGET)` (see section 3.2.3.1), `$(PROJECT_PATH)` (see section 3.2.3.2) and `$(SDK_INSTALL_PATH)` (see section 3.2.3.3) can be used when specifying the path.

3.2.2.6. Master Database

Specifies a GATT database file to use as master. The file must be included in the project.

When a master database file is specified then only that file will be built. A master database file typically contains `#include` statements to select particular database files from the project and build them in a specific order.

If no master database file is specified then the build looks for a database called `app_gatt_db.db`. If found, it is chosen as the master database.

The build will process the database files and generate a header file named after the master database, which may be included in other source files (e.g. if the master database is called `master_db.db`, then the file `master_db.h` will be generated). If no master database is specified, then the header file will be called `app_gatt_db.h`.

Note:

The macros `$(HARDWARE_TARGET)` (see section 3.2.3.1), `$(PROJECT_PATH)` (see section 3.2.3.2) and `$(SDK_INSTALL_PATH)` (see section 3.2.3.3) can be used when specifying the path.

3.2.2.7. Libraries

Specifies the libraries to be used in the project. These are passed to GCC using the `-l` command line option. See the *GCC Manual* for more information.

Multiple libraries should be separated with white space. For example:

```
../libraries/lib1 ../libraries/lib2
```

Note:

The special keywords `$(HARDWARE_TARGET)`, `$(PROJECT_PATH)` and `$(SDK_INSTALL_PATH)` can be used when specifying the path (see their description in section 3.2.3).

3.2.2.8. Library Search Paths

Specifies the paths to search for libraries used in the project. These are passed to GCC using the `-L` command line option in the order stated. The project directory is automatically included to be the first path in the list. The default SDK library path is automatically included to be the last path in the list. See the *GCC Manual* for more information.

Multiple paths should be separated with white space. For example:

```
../library1 ../library2/
```

Recursive paths can be mentioned using ellipsis (`...`). For example:

```
../library1/...
```

adds `../library1` and all its sub-folders to the list.

Note:

The macros `$(HARDWARE_TARGET)` (see section 3.2.3.1), `$(PROJECT_PATH)` (see section 3.2.3.2) and `$(SDK_INSTALL_PATH)` (see section 3.2.3.3) can be used when specifying the path.

3.2.2.9. Include Paths

Specifies the paths to search for included headers used in the project. These are passed to GCC using the `-I` command line option in the order stated. The project directory is automatically included to be the first path in the list. The default SDK include path is automatically included to be the last path in the list. See the *GCC Manual* for more information.

Multiple paths should be separated with white space. For example:

```
../include1 ../include2/
```

Recursive paths can be mentioned using ellipsis (`...`). For example, when using:

```
../include1/...
```

`../include1` and all subdirectories under it will be searched for include files.

Note:

The macros `$(HARDWARE_TARGET)` (see section 3.2.3.1), `$(PROJECT_PATH)` (see section 3.2.3.2) and `$(SDK_INSTALL_PATH)` (see section 3.2.3.3) can be used when specifying the path.

3.2.2.10. Define Macros

Specifies preprocessor macros to define. These are passed to GCC using the `-D` command line option. See the *GCC Manual* for more information.

Multiple macros should be separated with a comma. For example:

```
MARCRO1, MACRO2
```

3.2.2.11. Strip Symbols

Specifies whether to strip debug and unnecessary symbols from the output library file. It can be selected from a drop-down menu and can have the following values:

- **Yes**
- **No**

Note:

This property is only applicable when the **Output Type** is **Library**.

3.2.2.12. SPI Flash Clock PIO

Specifies the PIO port to be used for communicating clock signals with SPI Flash NVM.

Range:

[0,15]

Notes:

1. PIO3 and PIO4 are reserved for the MOSI and CSB SPI lines respectively.
2. This property is only used when the **NVM Type** is set to **FLASH**, and the SPI flash device capacity is greater than 512-kbit. See *Interfacing Large Serial Flash and EEPROM Application Note* in the Support Documentation for further information.

3.2.2.13. SPI Flash MISO PIO

Specifies the PIO port to be used for communicating MISO signals with SPI Flash NVM.

Range:

[0,15]

Notes:

1. PIO3 and PIO4 are reserved for the MOSI and CSB SPI lines respectively.
2. This property is only used when the **NVM Type** is set to **FLASH**, and the SPI flash device capacity is greater than 512-kbit. See *Interfacing Large Serial Flash and EEPROM Application Note* in the Support Documentation for further information.

3.2.2.14. Include OTA Update Bootloader

Specifies whether to include the OTA Update Bootloader in the application image.

It can be selected from a drop-down menu and can have the following values:

- **Yes**
- **No**

Select **Yes** to include the CSR OTA Update Bootloader in the application image. If the application supports the CSR OTA Update Service this will make it possible to update a device over-the-air using an OTA Update host application.

The original application image, without the bootloader, will still be built as `<Output Name>_update.img`.

See *Modifying an Application to Support OTA Update* in the Support Documentation for further information.

3.2.2.15. CS Key File (OTA Update Bootloader)

Specifies the filename with extension for the Configuration Store key definition file (`.keyr`) to be used by the OTA Update Bootloader. For example:

`otau_bootloader.keyr`.

Notes:

1. This property is only applicable when the OTA Update Bootloader is included
1. If the file is not present in the project folder, specify the absolute path or the path relative to the project folder.
2. The specified path and filename are case sensitive.
3. The macros `$(HARDWARE_TARGET)` (see section 3.2.3.1), `$(PROJECT_PATH)` (see section 3.2.3.2) and `$(SDK_INSTALL_PATH)` (see section 3.2.3.3) can be used when specifying the path.

3.2.2.16. Device Name (OTA Update Bootloader)

Specifies the name of the device advertised while the OTA Update Bootloader is searching for an OTA Update host.

The device name may be up to 16 characters long. If a longer name is supplied then it shall be truncated.

Notes:

This property is only applicable when the OTA Update Bootloader is included

3.2.2.17. Authentication Key (OTA Update Bootloader)

Specifies an authentication key that may be issued by an OTA Update host to authenticate itself to the device.

This property may be left blank if no authentication is required for Over-the-Air Updates.

The authentication key is a 32-digit (128-bit) hexadecimal value. All 32 digits in the key must be specified. The application will fail to build if a key is specified with the wrong length.

Notes:

This property is only applicable when the OTA Update Bootloader is included

3.2.2.18. Version (OTA Update Bootloader)

Specifies which version of CSR OTA Update Library to link the application against, and which version of CSR OTA Update Bootloader to include in the application image when **Include OTA Update Bootloader** is set to **Yes**. It can be selected from a drop-down menu and can have the following values:

- **6**
- **7**

Version 6 only supports devices with 512-kbit EEPROM NVM storage.

Version 7 supports devices with EEPROM or SPI Flash NVM storage, 512-kbit or greater.

Version 6 is provided so that applications developed for version 6 of CSR OTA Update Bootloader can continue to be supported. In addition, the version 6 CSR OTA Update Bootloader is significantly smaller than version 7 which makes it better suited to applications using a 512-kbit EEPROM for NVM storage.

Notes:

1. Applications linked against version 7 of the CSR OTA Update Library are backwardly compatible with versions 5 and 6 of the CSR OTA Update Bootloader.
2. Applications linked against version 6 of the CSR OTA Update Library are backwardly compatible with version 5 of the CSR OTA Update Bootloader.

3. Backwardly compatible applications may be updated over-the-air onto a device running an older version of the CSR OTA Update Bootloader.

3.2.2.19. Slot 1 Address (OTA Update Bootloader)

Specifies the start address for the first application slot.

See *Over-the-Air (OTA) Update System Application Note* for more information about this property.

Note:

This property is only applicable when the OTA Update Bootloader version 7 is included

3.2.2.20. Slot 2 Address (OTA Update Bootloader)

Specifies the start address for the second application slot.

See *Over-the-Air (OTA) Update System Application Note* for more information about this property.

Note:

This property is only applicable when the OTA Update Bootloader version 7 is included

3.2.2.21. Slot End Address (OTA Update Bootloader)

Specifies the end address for the second application slot.

See *Over-the-Air (OTA) Update System Application Note* for more information about this property.

Note:

This property is only applicable when the OTA Update Bootloader version 7 is included

3.2.3. Macros

A set of macros are automatically defined by the SDK and can be used when specifying some of the project properties. They are described in this section.

Note:

Macros `$(HARDWARE_TARGET)`, `$(SDK_INSTALL_PATH)` and `$(PROJECT_PATH)` can be used when specifying directory or file paths. This applies to the following project properties: **CS Key File** (see section 3.2.2.5), **Master Database** (see section 3.2.2.6), **Libraries** (see section 3.2.2.7), **Library Search Paths** (see section 3.2.2.8), **Include Paths** (see section 3.2.2.9) and **CS Key File (OTA Update Bootloader)** (see section 3.2.2.15).

3.2.3.1. `$(HARDWARE_TARGET)`

This macro can be used to automatically insert the target hardware name (for example, CSR101x_A05). For example, a library search path can be specified using it as follows:

```
../library/depend_Release_$(TARGET_HARDWARE)
```

3.2.3.2. `$(SDK_INSTALL_PATH)`

This macro can be used to automatically insert the path of the SDK installation. Typically, it will be replaced with `C:/CSR_uEnergy_SDK-<version>`. For example, a library search path can be specified using it as follows:

```
$(SDK_INSTALL_PATH)/tools/lib
```

3.2.3.3. `$(PROJECT_PATH)`

This macro can be used to automatically insert the path of the current project. For example, it can be used for specifying a library search path within a subfolder of a project folder as follows:

```
$(PROJECT_PATH)/subfolder/lib
```

3.3. Debugging in xIDE

xIDE supports the debugging of the application code running on the device, excluding core firmware.

The application is run on-chip, thus ensuring the debug environment matches the final execution environment of the product as closely as is possible.

xIDE provides a familiar debugging toolset that includes facilities required to efficiently debug programs running on the device.

While many of the facilities provided in xIDE are typical debugging tools, a few are more specific to an integrated implementation such as CSR µEnergy.

3.3.1. Brief Overview of Debug Facilities

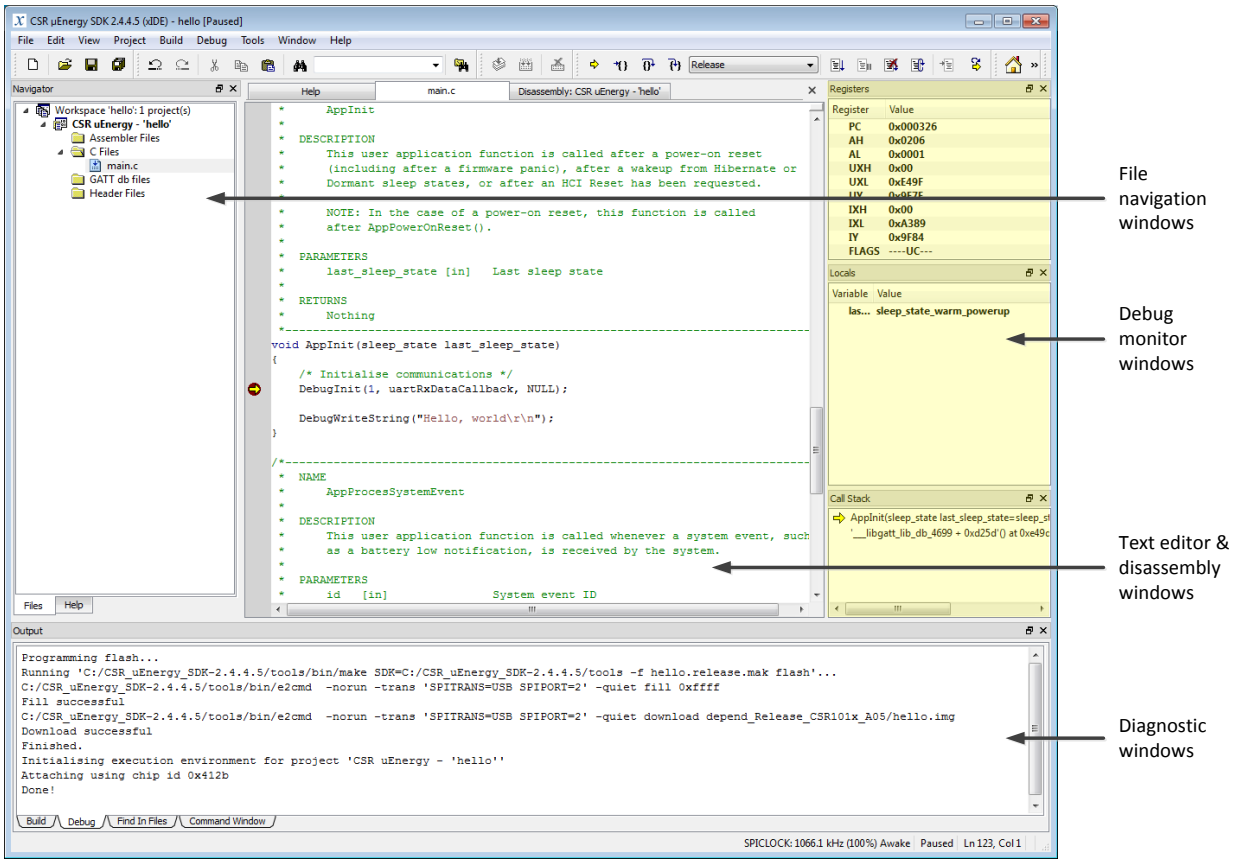


Figure 3.3: Project Debugging

The xIDE debug view consists of four basic work areas:

- File navigation windows
- Debug monitor windows (highlighted in yellow)
- Text editor and disassembly windows
- Diagnostic windows

A brief description of each is given in this section.

3.3.1.1. Debug Menu and Toolbar Operations

The following operations are provided on the **Debug** menu and toolbar.

Run 

Execute the program on chip. The program pauses if a breakpoint is reached, otherwise it continues to execute until paused.

Run To Cursor 

Execute the program on chip until the cursor is reached. This is equivalent to setting a breakpoint at the cursor and selecting the Run operation.

Step Over 

Execute the next statement. If the statement contains one or more function calls, the calls are stepped over; i.e. they are executed and the processor is paused when the functions have returned.

Step Into 

Execute the next statement. If the statement contains one or more function calls, this operation steps into the first call that is executed; i.e. the processor is paused when it reaches the first statement of that function.

The equivalent **Step Out** function is not supported on CSR µEnergy devices. To step out of a function, position the cursor on its closing brace ('}') and select the **Run To Cursor** operation. When the cursor is reached and the program has paused select the **Step Over** operation.

Attach… 

Creates a debug connection to the chip without downloading the application, resetting the device, or otherwise affecting the state of the program.

Pause 

Pause execution and display the next statement that will be executed.

Restart 

Reset the device and restart the program, leaving the device in the paused state.

Stop Debugging 

Disconnect the debug connection to the device. If the processor is running, the device continues to execute the program.

Show Next Statement 

Display the next statement to be executed in the editor or disassembler.

Set Next Statement 

Set the device processor's program counter (PC).

Note:

Run, **Run To Cursor**, **Step Over**, **Step Into** and **Restart** operations depend on the debugger connection status.

If the debugger is not connected to the device, these operations first build the application, download it to the device and create a debug connection to the device before carrying out the operation.

3.3.1.2. File Navigation Windows

This area displays an explorer-like view of the project workspace currently loaded in xIDE, see Figure 3.4:

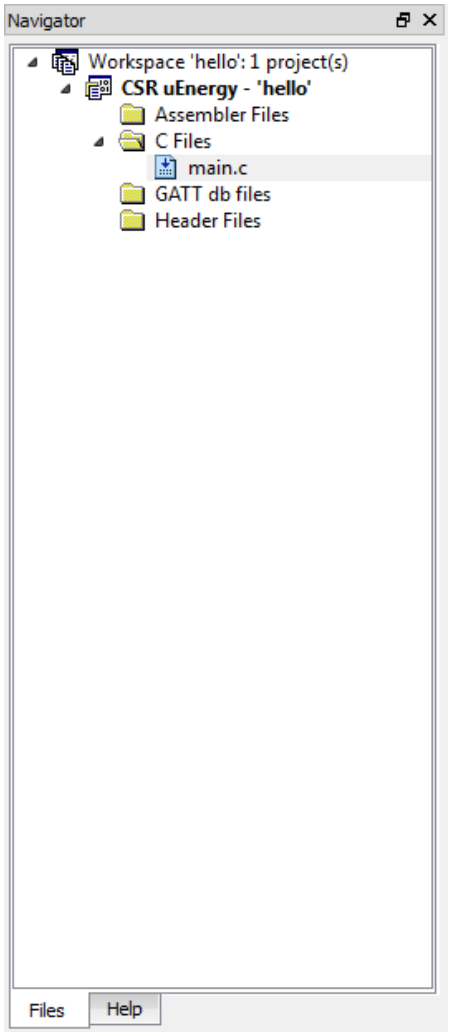


Figure 3.4: File Navigator Window

The file structure can be navigated and files opened in the text editor by double-clicking on a file.

Right-clicking on an item listed in the **File Navigation** window will open a context-sensitive menu.

Files can also be added to the project workspace from a folder other than the project folder.

The **Help** tab displays any associated support documentation.

3.3.1.3. Text Editor and Disassembly Windows

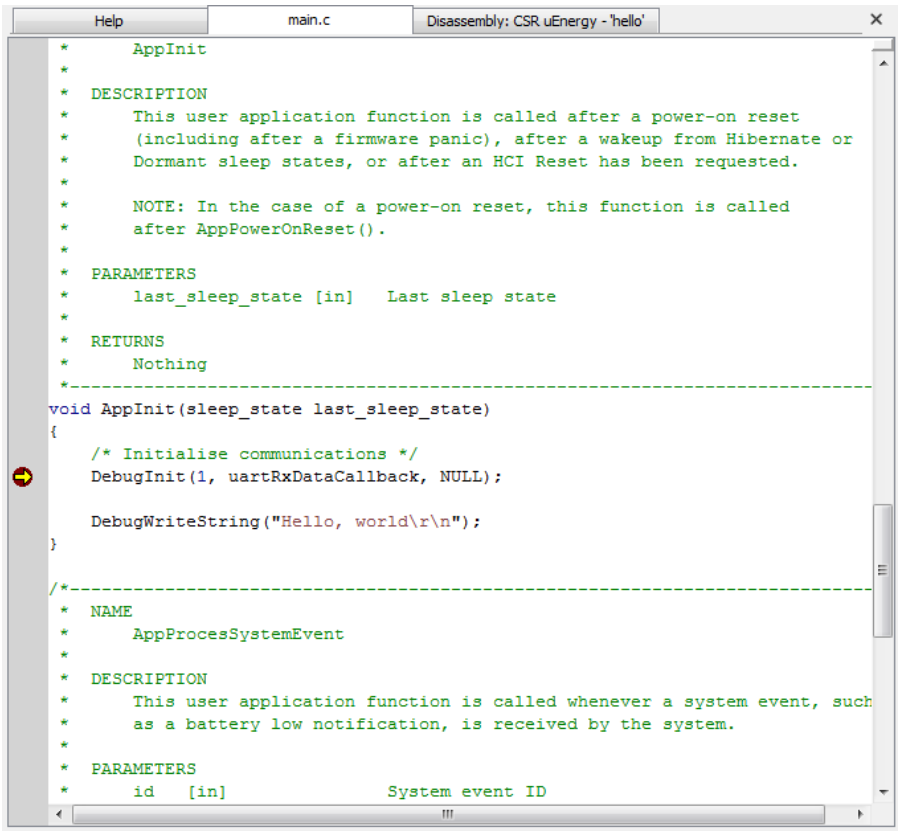


Figure 3.5: Text Editor Window

This area displays open project files and allows:

- Text editing
- Breakpoints to be set
- Code disassembly to be viewed
- Tabs allow navigation between multiple files opened in the text editor

Note:

An * displayed after the file name on a file tab (e.g. `main.c*`) indicates that the file has been amended and has not been saved.

Right-clicking on the text editor window will open a context-sensitive menu.

3.3.1.4. Debug Monitor Windows

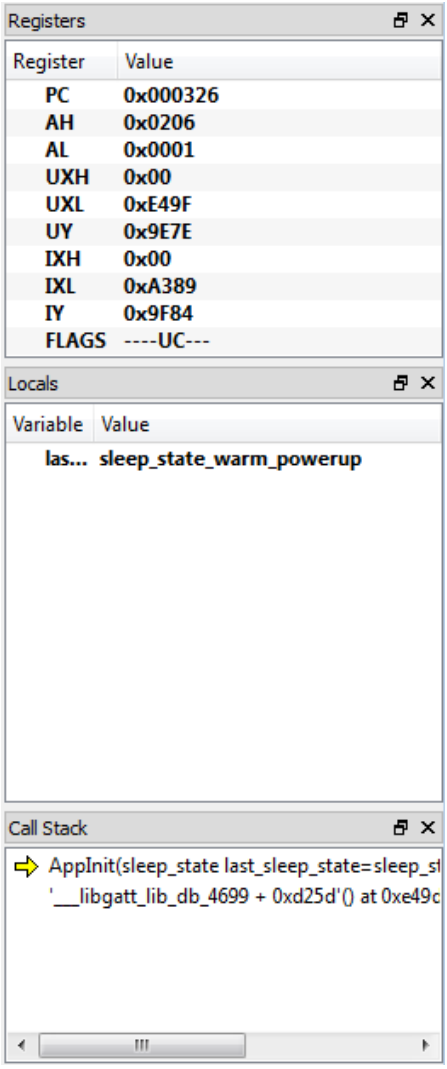


Figure 3.6: Debug Monitor Windows

Developers can select various windows that monitor the device state:

- **Registers**: Displays current values of registers
- **Memory**: Displays current values of selected memory addresses
- **Breakpoints**: Lists currently set program breakpoint locations
- **Watch**: Allows the user to view the current value of specific program variables
- **Source Locator**: Locates source code that has been moved since it was built
- **Call Stack**: Displays the current function call stack
- **Variables**: Displays all variables that are in-scope at the current program location
- **Statics**: As Variables, but shows only file-scope variables
- **Globals**: As Variables, but shows only global variables
- **Locals**: As Variables, but shows only function-scope variables

The views can be toggled on and off from the **View/Debug Windows** menu list.

3.3.1.5. Diagnostics Windows

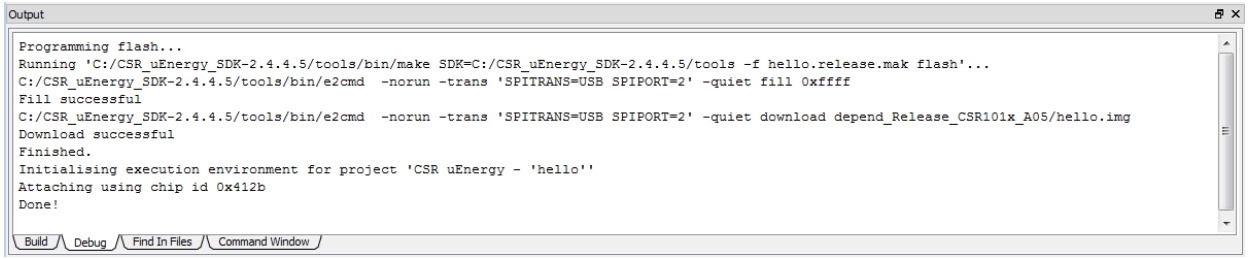


Figure 3.7: Diagnostics Window

This area displays various tabbed windows that display useful information when building and debugging code:

- **Build**: Displays information on the build process and status.
- **Debug**: Displays information on the debug process and status.
- **Find in Files**: Displays the results of the Find in Files facility accessed from the **Edit** menu or toolbar.
- **Command Window**: This window can be used to invoke Python scripts to extend xIDE.

Note:

Most developers need not concern themselves with the **Command Window**.

Right-clicking on a diagnostics window will open a context-sensitive menu.

4. SDK Build Process

The SDK's build process combines the application code, GATT database code and optional 8051 PIO Controller assembly code contained within the project, together with the firmware library to create the binary image for either EEPROM or SPI Flash external memory device.

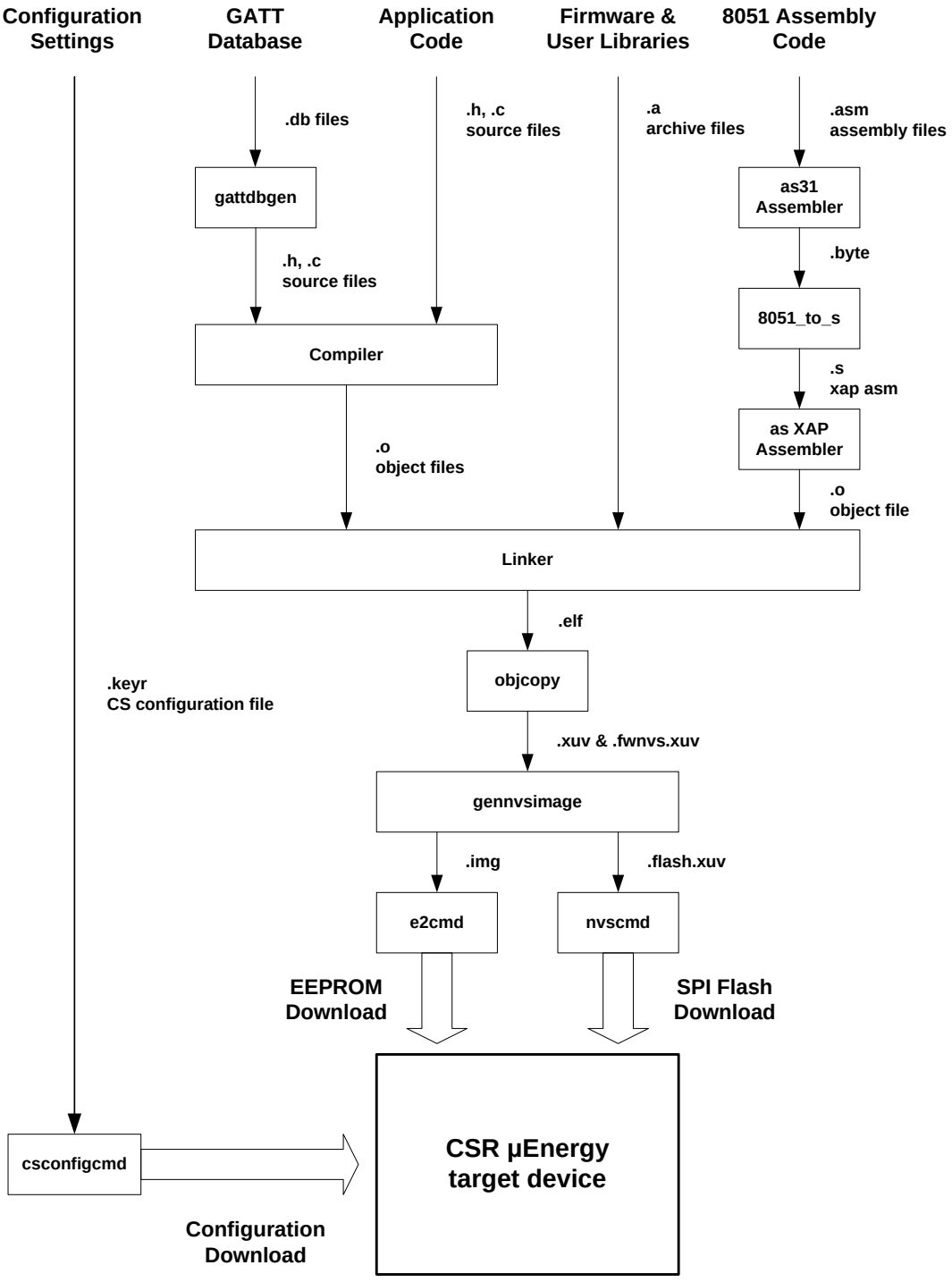


Figure 4.1: SDK Build Process

5. Frequently Asked Questions (FAQs)

1. *Can I customise the xIDE environment?*

   Yes. The **Options** menu item in the **Tools** menu allows you to select various options affecting the appearance and behaviour of xIDE.

   The **View** menu offers a number of layout options allowing you to toggle the display to show your preferred debug windows and menu items.

   Windows can be reorganised by dragging and dropping within the main xIDE window or on the desktop to create separate displays.

2. *How can I set Configuration Store values (sometimes referred as NVS Keys)?*

   Either follow the steps in section 3.1.3.1.

   Or edit the values in the `.keyr` file using xIDE before running the application.

3. *How can I restore the chip's factory settings?*

   Either use the **csconfigcmd** command line tool to load the configuration file onto the device

   Or launch the **CsConfig** tool and select **Reset All**. This assumes the configuration keys on the device are compatible with the firmware application on the PC.

4. *How can I use SPI flash instead of EEPROM?*

   The **NVM Type** can be selected from the **Properties** menu item in the **Project** menu.

5. *How can I change the SPI transport?*

   The debug transport can be configured for each project using the **Transport…** option under the **Debug** menu.

6. *How can I skip flashing the board every time I run a project?*

   If you are sure the onboard image matches the debugging target, use the Attach option (i.e. press **Ctrl+F5** or select **Attach** from the **Debug** menu) instead of using Run (i.e. pressing **F5** or selecting **Run** from the **Debug** menu).

7. *Where can I find the latest updates?*

   The latest updates to CSR µEnergy software can be found on our technical support website (www.csrsupport.com).

Document References

Document	Reference
CSR µEnergy xIDE User Guide	CS-212742-UG
GCC manual	https://gcc.gnu.org/onlinedocs/
Interfacing Large Serial Flash and EEPROM Application Note	CS-324434-AN
Modifying an Application to Support OTA Update Application Note	CS-304564-AN
Over-the-Air (OTA) Update System Application	CS-316019-AN

Terms and Definitions

8051	Family of microcontrollers based on the Intel® MCS-51
ANSI	American National Standards Institute
Bluetooth®	Set of technologies providing audio and data transfer over short-range radio connections
Bluetooth LE	Bluetooth Low Energy: a Bluetooth technology designed for ultra-low power consumption
Bluetooth Smart	A term used to indicate devices supporting Bluetooth LE
CD	Compact Disc
COM	Serial Communication Port
CS	Configuration Store (area of memory used by the firmware to store configuration values)
CSB	Chip Select
CSR	Cambridge Silicon Radio
CSR101x	Any of the CSR1010, CSR1011, CSR1012 or CSRB31010 devices
CSR µEnergy®	Group term for CSR's range of Bluetooth Smart wireless technology chips
e.g.	*exempli gratia*, for example
EEPROM	Electrically Erasable Programmable Read-Only Memory
FAQ	Frequently Asked Questions
GATT	Generic Attribute Profile
GCC	GNU Compiler Collection
IC	Integrated Circuit
i.e.	*id est*, that is
kbit	Kilobit: 1,024 bits
LPT	Local Printer Port
Mbyte	Megabyte: 1,048,576 bytes, where 1 byte is 1 octet
MISO	Master-In Slave-Out: a SPI data line
MMI	Man Machine Interface
MOSI	Master Out Slave In
NVS	Non-Volatile Storage
NVM	Non-Volatile Memory
OTA	Over-the-Air
PC	Personal Computer / Program Counter
PIO	Programmable Input Output
ROM	Read Only Memory

SDK	Software Development Kit
SPI	Serial Peripheral Interface
USB	Universal Serial Bus
XAP	Family of processors included in CSR µEnergy devices
xIDE	Integrated Development Environment for CSR µEnergy devices