IAR Embedded Workbench

IDE Project Management and Building Guide

for the Renesas

RH850 Family

UIDERH850-5

IAR SYSTEMS

COPYRIGHT NOTICE
© 2014–2020 IAR Systems AB.

No part of this document may be reproduced without the prior written consent of IAR Systems AB. The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such a license.

DISCLAIMER
The information in this document is subject to change without notice and does not represent a commitment on any part of IAR Systems. While the information contained herein is assumed to be accurate, IAR Systems assumes no responsibility for any errors or omissions.

In no event shall IAR Systems, its employees, its contractors, or the authors of this document be liable for special, direct, indirect, or consequential damage, losses, costs, charges, claims, demands, claim for lost profits, fees, or expenses of any nature or kind.

TRADEMARKS
IAR Systems, IAR Embedded Workbench, Embedded Trust, C-Trust, IAR Connect, C-SPY, C-RUN, C-STAT, IAR Visual State, IAR KickStart Kit, I-jet, I-jet Trace, I-scope, IAR Academy, IAR, and the logotype of IAR Systems are trademarks or registered trademarks owned by IAR Systems AB.

Microsoft and Windows are registered trademarks of Microsoft Corporation.

Renesas is a registered trademark of Renesas Electronics Corporation. RH850 is a trademark of Renesas Electronics Corporation.

Adobe and Acrobat Reader are registered trademarks of Adobe Systems Incorporated.

All other product names are trademarks or registered trademarks of their respective owners.

EDITION NOTICE
Fifth edition: April 2020

Part number: UIDERH850-5

This guide applies to version 2.x of IAR Embedded Workbench® for the Renesas RH850 Family.

Internal reference: BB6, Mym8.4, V_110411, IJOA.

Brief contents

Contents

Tables

Preface

- Who should read this guide
- How to use this guide
- What this guide contains
- Other documentation
- Document conventions

Who should read this guide

Read this guide if you plan to develop an application using IAR Embedded Workbench and want to get the most out of the features and tools available in the IDE.

REQUIRED KNOWLEDGE

To use the tools in IAR Embedded Workbench, you should have working knowledge of:

- The architecture and instruction set of the RH850 microcontroller (refer to the chip manufacturer's documentation)
- The C or C++ programming language
- Application development for embedded systems
- The operating system of your host computer.

For more information about the other development tools incorporated in the IDE, refer to their respective documentation, see *Other documentation*, page 17.

How to use this guide

Each chapter in this guide covers a specific *topic area*. In many chapters, information is typically divided into different sections based on *information types*:

- *Concepts*, which describes the topic and gives overviews of features related to the topic area. Any requirements or restrictions are also listed. Read this section to learn about the topic area.
- *Tasks*, which lists useful tasks related to the topic area. For many of the tasks, you can also find step-by-step descriptions. Read this section for information about required tasks as well as for information about how to perform certain tasks.

- *Reference information*, which gives reference information related to the topic area. Read this section for information about certain GUI components. You can easily access this type of information for a certain component in the IDE by pressing F1.

If you are new to using IAR Embedded Workbench, the tutorials, which you can find in the IAR Information Center, will help you get started using IAR Embedded Workbench.

Finally, we recommend the *Glossary* if you should encounter any unfamiliar terms in the IAR Systems user documentation.

What this guide contains

This is a brief outline and summary of the chapters in this guide.

PART 1. PROJECT MANAGEMENT AND BUILDING

This section describes the process of editing and building your application:

- *The development environment* introduces you to the IAR Embedded Workbench development environment. The chapter also demonstrates the facilities available for customizing the environment to meet your requirements.
- *Project management* describes how you can create workspaces with multiple projects, build configurations, groups, source files, and options that help you handle different versions of your applications.
- *Building projects* discusses the process of building your application.
- *Editing* contains detailed descriptions of the IAR Embedded Workbench editor, how to use it, and the facilities related to its usage. The final section also contains information about how to integrate an external editor of your choice.

PART 2. REFERENCE INFORMATION

- *Product files* describes the directory structure and the types of files it contains.
- *Menu reference* contains detailed reference information about menus and menu commands.
- *General options* specifies the target, output, library, and MISRA C options.
- *Compiler options* specifies compiler options for language, optimizations, code, output, list file, preprocessor, diagnostics, and MISRA C.
- *Assembler options* describes the assembler options for language, output, list, preprocessor, and diagnostics.
- *Output converter options* describes the options available for converting linker output files from the ELF format.
- *Custom build options* describes the options available for custom tool configuration.

- *Build actions options* describes the options available for pre-build and post-build actions.
- *Linker options* describes the options for setting up for linking.
- *Library builder options* describes the options for building a library.

Other documentation

User documentation is available as hypertext PDFs and as a context-sensitive online help system in HTML format. You can access the documentation from the Information Center or from the **Help** menu in the IAR Embedded Workbench IDE. The online help system is also available via the F1 key.

USER AND REFERENCE GUIDES

The complete set of IAR Systems development tools is described in a series of guides. Information about:

- System requirements and information about how to install and register the IAR Systems products are available in the *Installation and Licensing Quick Reference Guide* and the *Licensing Guide*.
- Using the IDE for project management and building, is available in the *IDE Project Management and Building Guide for RH850.*
- Using the IAR C-SPY® Debugger, is available in the *C-SPY® Debugging Guide for RH850.*
- Programming for the IAR C/C++ Compiler for RH850 and linking using the IAR ILINK Linker, is available in the *IAR C/C++ Development Guide for RH850.*
- Programming for the IAR Assembler for RH850, is available in the *IAR Assembler Reference Guide for RH850.*
- Performing a static analysis using C-STAT and the required checks, is available in the *C-STAT® Static Analysis Guide*.
- Developing safety-critical applications using the MISRA C guidelines, is available in the *IAR Embedded Workbench® MISRA C:2004 Reference Guide* or the *IAR Embedded Workbench® MISRA C:1998 Reference Guide*.
- Migrating from an older UBROF-based product version to a newer version that uses the ELF/DWARF object format, is available in the guide *IAR Embedded Workbench® Migrating from UBROF to ELF/DWARF*.

Note: Additional documentation might be available depending on your product installation.

THE ONLINE HELP SYSTEM

The context-sensitive online help contains information about:

- IDE project management and building
- Debugging using the IAR C-SPY® Debugger
- The IAR C/C++ Compiler
- The IAR Assembler
- Keyword reference information for the DLIB library functions. To obtain reference information for a function, select the function name in the editor window and press F1.
- C-STAT
- MISRA C

WEB SITES

Recommended web sites:

- The Renesas web site, **www.renesas.com**, that contains information and news about the RH850 Family.
- The IAR Systems web site, **www.iar.com**, that holds application notes and other product information.
- The web site of the C standardization working group, **www.open-std.org/jtc1/sc22/wg14**.
- The web site of the C++ Standards Committee, **www.open-std.org/jtc1/sc22/wg21**.
- The C++ programming language web site, **isocpp.org**. This web site also has a list of recommended books about C++ programming.
- The C and C++ reference web site, **en.cppreference.com**.

Document conventions

When, in the IAR Systems documentation, we refer to the programming language C, the text also applies to C++, unless otherwise stated.

When referring to a directory in your product installation, for example `rh850\doc`, the full path to the location is assumed, for example `c:\Program Files\IAR Systems\Embedded Workbench N.n\rh850\doc`, where the initial digit of the version number reflects the initial digit of the version number of the IAR Embedded Workbench shared components.

TYPOGRAPHIC CONVENTIONS

The IAR Systems documentation set uses the following typographic conventions:

Style	Used for
`computer`	• Source code examples and file paths. • Text on the command line. • Binary, hexadecimal, and octal numbers.
`parameter`	A placeholder for an actual value used as a parameter, for example *`filename`*`.h` where *`filename`* represents the name of the file.
[`option`**]**	An optional part of a linker or stack usage control directive, where **[** and **]** are not part of the actual directive, but any `[`, `]`, `{`, or `}` are part of the directive syntax.
{`option`**}**	A mandatory part of a linker or stack usage control directive, where **{** and **}** are not part of the actual directive, but any `[`, `]`, `{`, or `}` are part of the directive syntax.
`[option]`	An optional part of a command line option, pragma directive, or library filename.
`[a|b|c]`	An optional part of a command line option, pragma directive, or library filename with alternatives.
`{a|b|c}`	A mandatory part of a command line option, pragma directive, or library filename with alternatives.
bold	Names of menus, menu commands, buttons, and dialog boxes that appear on the screen.
italic	• A cross-reference within this guide or to another guide. • Emphasis.
…	An ellipsis indicates that the previous item can be repeated an arbitrary number of times.
	Identifies instructions specific to the IAR Embedded Workbench® IDE interface.
	Identifies instructions specific to the command line interface.
	Identifies helpful tips and programming hints.
	Identifies warnings.

Table 1: Typographic conventions used in this guide

NAMING CONVENTIONS

The following naming conventions are used for the products and tools from IAR Systems®, when referred to in the documentation:

Brand name	Generic term
IAR Embedded Workbench® for RH850	IAR Embedded Workbench®
IAR Embedded Workbench® IDE for RH850	the IDE
IAR C-SPY® Debugger for RH850	C-SPY, the debugger
IAR C-SPY® Simulator	the simulator
IAR C/C++ Compiler™ for RH850	the compiler
IAR Assembler™ for RH850	the assembler
IAR ILINK Linker™	ILINK, the linker
IAR DLIB Runtime Environment™	the DLIB runtime environment

Table 2: Naming conventions used in this guide

Part 1. Project management and building

This part contains these chapters:

- The development environment
- Project management
- Building projects
- Editing

The development environment

- Introduction to the IAR Embedded Workbench IDE
- Using and customizing the IDE
- Reference information on the IDE

Introduction to the IAR Embedded Workbench IDE

These topics are covered:

- Briefly about the IDE and the build toolchain
- Tools for analyzing and checking your application
- An extensible and modular environment
- The layout of the windows on the screen

BRIEFLY ABOUT THE IDE AND THE BUILD TOOLCHAIN

The IDE is the environment where all tools needed to build your application—the *build toolchain*—are integrated: a C/C++ compiler, C/C++ libraries, an assembler, a linker, library tools, an editor, a project manager with Make utility, and the IAR C-SPY® Debugger. The tools used specifically for building your source code are referred to as the *build tools*.

The toolchain that comes with your product package supports a specific microcontroller. However, the IDE can simultaneously contain multiple toolchains for various microcontrollers. This means that if you have IAR Embedded Workbench installed for several microcontrollers, you can choose which microcontroller to develop for.

Note: The compiler, assembler, and linker and library tools can also be run from a command line environment, if you want to use them as external tools in an already established project environment.

TOOLS FOR ANALYZING AND CHECKING YOUR APPLICATION

IAR Embedded Workbench comes with various types of support for analyzing and finding errors in your application, such as:

- Compiler and linker errors, warnings, and remarks

  All diagnostic messages are issued as complete, self-explanatory messages. Errors reveal syntax or semantic errors, warnings indicate potential problems, and remarks (default off) indicate deviations from the standard. Double-click a message and the corresponding source code construction is highlighted in the editor window. For more information, see the *IAR C/C++ Development Guide for RH850*.

- Stack usage analysis during linking

  Under the right circumstances, the linker can accurately calculate the maximum stack usage for each call tree, such as `cstartup`, interrupt functions, RTOS tasks, etc. For more information, see the *IAR C/C++ Development Guide for RH850*.

- C-STAT for static analysis

  C-STAT is a static analysis tool that tries to find deviations from specific sets of *rules*, where each rule specifies an unsafe source construct. The rules come from various institutes, like MISRA (MISRA C:2004, MISRA C++:2008, and MISRA C:2012), CWE, and CERT. For information about how to use C-STAT and the rules, see the *C-STAT® Static Analysis Guide*.

- MISRA C:1998 and 2004

  In addition to the MISRA checks in C-STAT, the IDE provides compiler checks for MISRA C:1998 and 2004. For more information, see the *IAR Embedded Workbench® MISRA C:2004 Reference Guide* or the *IAR Embedded Workbench® MISRA C:1998 Reference Guide*.

- C-SPY debugging features such as, Profiling, Code Coverage, Trace, and Power debugging. For more information, see the *C-SPY® Debugging Guide for RH850*.

AN EXTENSIBLE AND MODULAR ENVIRONMENT

Although the IDE provides all the features required for your project, you can also integrate other tools. For example, you can:

- Use the Custom Build mechanism to add other tools to the toolchain, see *Extending the toolchain*, page 109.
- Add IAR visualSTATE to the toolchain, which means that you can add state machine diagrams directly to your project in the IDE.
- Use the Subversion version control system to keep track of different versions of your source code. The IDE can attach to files in a Subversion working copy.

- Add an external analyzer, for example a lint tool, of your choice to be used on whole projects, groups of files, or an individual file of your project. Typically, you might want to perform a static code analysis on your source code, using the same settings and set of source code files as when you compile. See *Getting started using external analyzers*, page 31.
- Add external tools to the **Tools** menu, for convenient access from within the IDE. For this reason, the menu might look different depending on which tools you have preconfigured to appear as menu commands.
- Configure custom argument variables, which typically can be useful if you install a third-party product and want to specify its include directory. Custom argument variables can also be used for simplifying references to files that you want to be part of your project.

THE LAYOUT OF THE WINDOWS ON THE SCREEN

In the IDE, each window that you open has a default location, which depends on other currently open windows. You can position the windows and arrange a layout according to your preferences. Each window can be either *docked* or *floating*.

You can dock each window at specific places, and organize them in *tab groups*. If you rearrange the size of one docked window, the sizes of any other docked windows are adjusted accordingly. You can also make a window floating, which means it is always on top of other windows. The location and size of a floating window does not affect other currently open windows. You can move a floating window to any place on your screen, including outside of the IAR Embedded Workbench IDE main window.

Each time you open a previously saved workspace, the same windows are open, and they have the same sizes and positions.

For every project that is executed in the C-SPY environment, a separate layout is saved. In addition to the information saved for the workspace, information about all open debugger-specific windows is also saved.

Note: The editor window is always docked. When you open the editor window, its placement is decided automatically depending on other currently open windows. For more information about how to work with the editor window, see *Introduction to the IAR Embedded Workbench editor*, page 125.

Using and customizing the IDE

These tasks are covered:

- Running the IDE
- Working with example projects

- Organizing windows on the screen
- Specifying tool options
- Adding a button to a toolbar
- Removing a button from a toolbar
- Showing/hiding toolbar buttons
- Recognizing filename extensions
- Getting started using external analyzers
- Invoking external tools from the Tools menu
- Adding command line commands to the Tools menu
- Using an external editor

See also *Extending the toolchain*, page 109.

For more information about customizations related to C-SPY, see the *C-SPY® Debugging Guide for RH850*.

RUNNING THE IDE

Click the **Start** button on the Windows taskbar and choose **All Programs>IAR Systems>IAR EW for RH850>IAR EW for RH850**.

The file `IarIdePm.exe` is located in the `common\bin` directory under your IAR Systems installation, in case you want to start the program from the command line or from within Windows Explorer.

Double-clicking the workspace filename

The workspace file has the filename extension `eww`. If you double-click a workspace filename, the IDE starts.

If you have several versions of IAR Embedded Workbench installed, the workspace file is opened by the most recently used version of your IAR Embedded Workbench that uses that file type, regardless of which version the project file was created in.

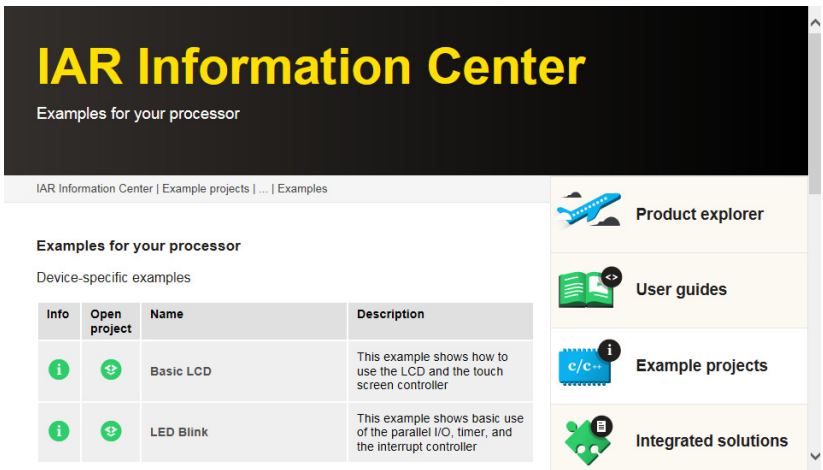
WORKING WITH EXAMPLE PROJECTS

Example applications are provided with IAR Embedded Workbench. You can use these examples to get started using the development tools from IAR Systems. You can also use the examples as a starting point for your application project.

You can find the examples in the `rh850\examples` directory. The examples are ready to be used as is. They are supplied with ready-made workspace files, together with source code files and all other related files.

To run an example project:

1. Choose **Help>Information Center** and click **Example projects**.
2. Browse to the example that matches the specific evaluation board or starter kit you are using.



Click the **Open Project** button.

3. In the dialog box that appears, choose a destination folder for your project.
4. The available example projects are displayed in the workspace window. Select one of the projects, and if it is not the active project (highlighted in bold), right-click it and choose **Set as Active** from the context menu.
5. To view the project settings, select the project and choose **Project>Options**. Verify the settings for **General Options>Target>Device** and **Debugger>Setup>Driver**. As for other settings, the project is set up to suit the target system you selected.

   For more information about the C-SPY options and how to configure C-SPY to interact with the target board, see the *C-SPY® Debugging Guide for RH850*.

   Click **OK** to close the project **Options** dialog box.

6. To compile and link the application, choose **Project>Make** or click the **Make** button.
7. To start C-SPY, choose **Project>Download and Debug** or click the **Download and Debug** button. If C-SPY fails to establish contact with the target system, see the *C-SPY® Debugging Guide for RH850*.

8. Choose **Debug>Go** or click the **Go** button to start the application.

Click the **Stop** button to stop execution.

ORGANIZING WINDOWS ON THE SCREEN

Use these methods to organize the windows on your screen:

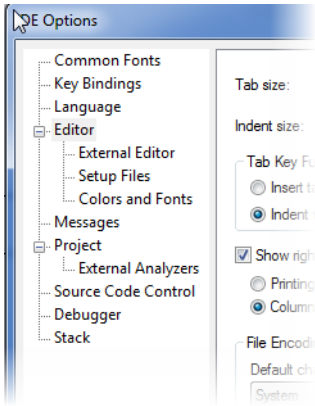
- To disconnect a tabbed window from a tab group and place it as a *separate* window, drag the tab away from the tab group.
- To make a window or tab group floating, double-click on the window's title bar.
- When dragging a window to move it, press Ctrl to prevent it from docking.
- To place a window in the same tab group as another open window, drag the window you want to relocate and drop it on the other window. Drop it on one of the arrow buttons of the organizer control, to control how to dock it.

See also *The layout of the windows on the screen*, page 25.

SPECIFYING TOOL OPTIONS

You can find commands for customizing the IDE on the **Tools** menu.

1 To display the **IDE Options** dialog box, choose **Tools>Options** to get access to a wide variety of options:



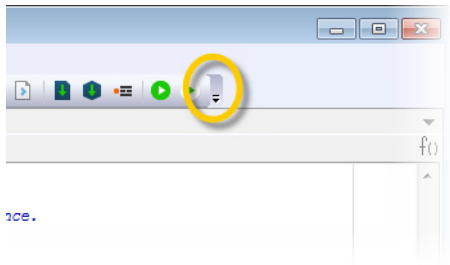
2 To access the options to the right in the dialog box, select a category to the left.

For more information about various options for customizing the IDE, see *Tools menu*, page 194.

ADDING A BUTTON TO A TOOLBAR

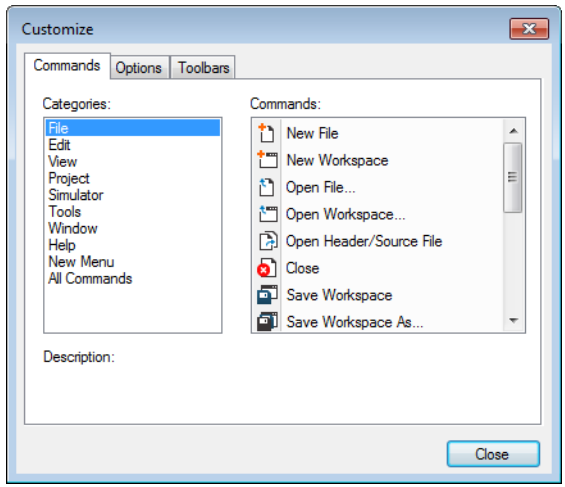
The buttons on the IDE toolbars provide shortcuts for commands on the IDE menus.

1 To add a new button to a toolbar in the main IDE window, click the **Toolbar Options** button and choose **Add or Remove Buttons>Customize**.



2 The **Customize** dialog box opens on the **Commands** page.

In the **Categories** list, select the menu on which the command you want to add to the toolbar is located.



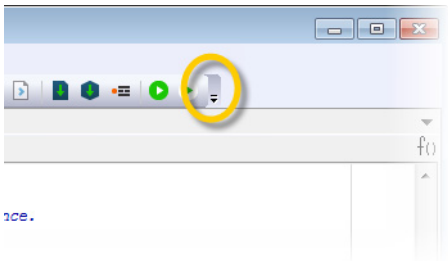
3 Drag a command from the **Commands** list to one of the toolbars where you want to insert the command as a button.

You can rearrange the existing buttons by dragging them to new positions.

Note: If you instead of adding a button want to show a button that has been hidden temporarily, see *Showing/hiding toolbar buttons*, page 30.

REMOVING A BUTTON FROM A TOOLBAR

1 To remove a button from any of the toolbars in the main window of the IDE, click the **Toolbar Options** button and choose **Add or Remove Buttons>Customize**. Ignore the **Customize** dialog box that is opened.



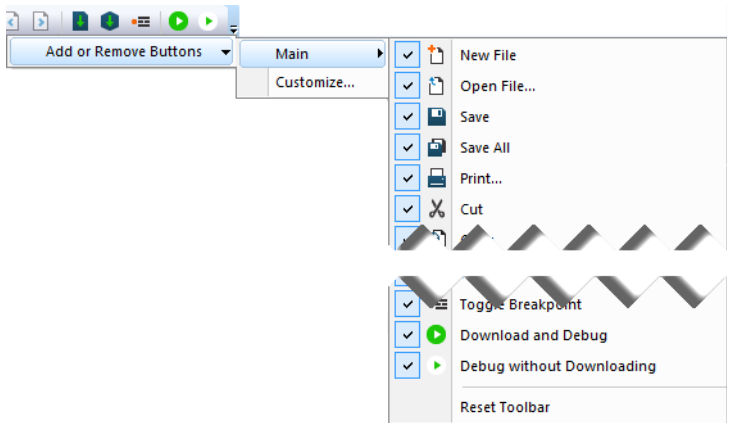
2 Right-click on the toolbar button that you want to remove and choose **Delete** from the context menu.

Note: If you instead of removing a button want to hide it temporarily, see *Showing/hiding toolbar buttons*, page 30.

SHOWING/HIDING TOOLBAR BUTTONS

As an alternative to removing a button from an IDE toolbar, you can toggle its visibility on/off.

1 To hide a button temporarily from any of the toolbars in the main window of the IDE, click the **Toolbar Options** button and choose **Add or Remove Buttons>*toolbar***.



2 Select or deselect the command button you want to show/hide.

Note: If you want to delete a button entirely from the toolbar, see *Removing a button from a toolbar*, page 30.

RECOGNIZING FILENAME EXTENSIONS

In the IDE, you can increase the number of recognized filename extensions. By default, each tool in the build toolchain accepts a set of standard filename extensions. Also, if you have source files with a different filename extension, you can modify the set of accepted filename extensions.

To get access to the necessary commands, choose **Tools>Filename Extensions**.

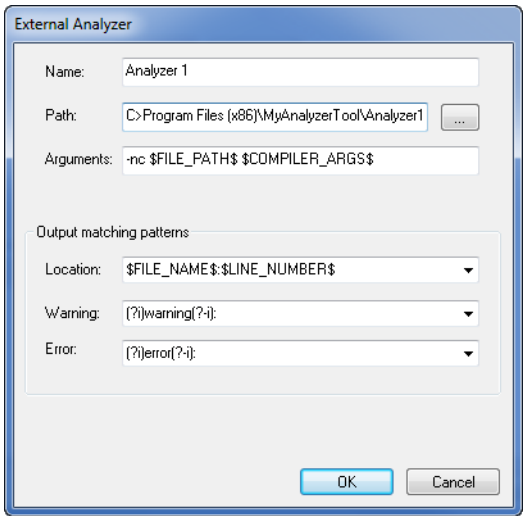
See *Filename Extensions dialog box*, page 78.

To override the default filename extension from the command line, include an explicit extension when you specify a filename.

GETTING STARTED USING EXTERNAL ANALYZERS

1 To add an external analyzer to the **Project** menu, choose **Tools>Options** to open the **IDE Options** dialog box and select the **Project>External Analyzers** page.

2 To configure the invocation, click **Add** to open the **External Analyzer** dialog box.



Specify the details required for the analyzer you want to be able to invoke.

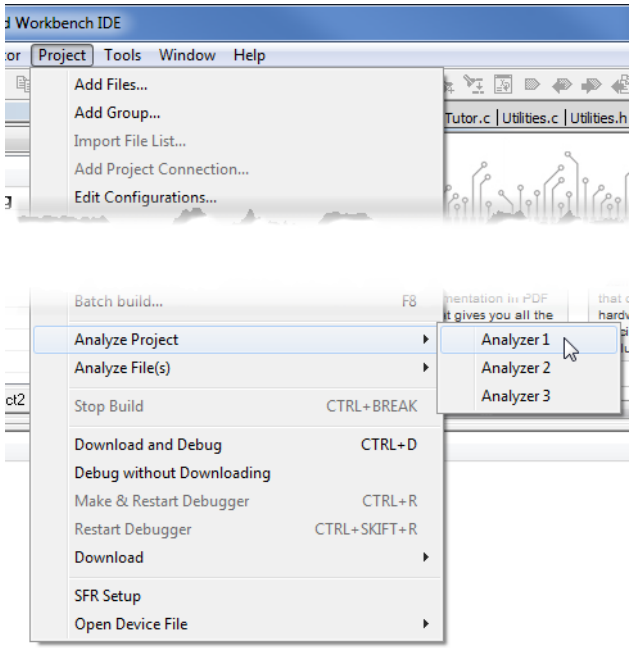
Use **Output matching patterns** to specify (or choose from a list) three regular expressions for identifying warning and error messages and to find references to source file locations.

Click **OK** when you have finished.

For more information about this dialog box, see *External Analyzer dialog box*, page 65.

3 In the **IDE Options** dialog box, click **OK**.

4 Choose **Project>Analyze Project** and select the analyzer that you want to run, alternatively choose **Analyze File(s)** to run the analyzer on individual files.

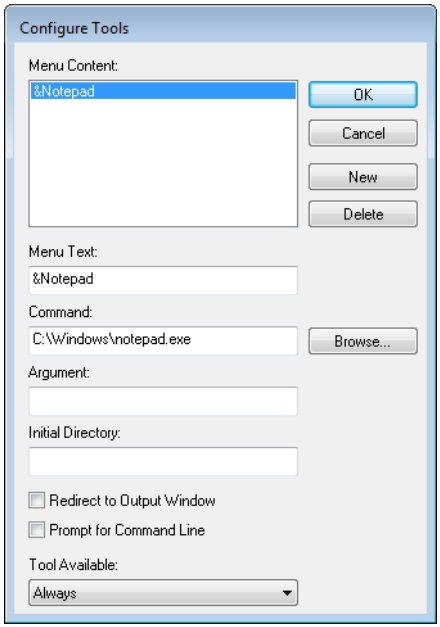


Each of the regular expressions that you specified will be applied on each line of output from the external analyzer. Output from the analyzer is listed in the **Build Log** window. You can double-click any line that matches the **Location** regular expression you specified in the **External Analyzer** dialog box to jump to the corresponding location in the editor window.

Note: If you want to stop the analysis before it is finished, click the **Stop Build** button.

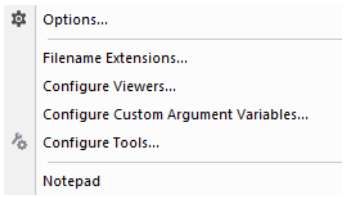
INVOKING EXTERNAL TOOLS FROM THE TOOLS MENU

1 To add an external tool to the menu, for example Notepad, choose **Tools>Configure Tools** to open the **Configure Tools** dialog box.



2 Fill in the text fields according to the screenshot. For more information about this dialog box, see *Configure Tools dialog box*, page 74.

3 After you have entered the appropriate information and clicked **OK**, the menu command you have specified is displayed on the **Tools** menu.



Note: You cannot use the **Configure Tools** dialog box to extend the toolchain in the IDE. If you intend to add an external tool to the standard build toolchain, see *Extending the toolchain*, page 109.

ADDING COMMAND LINE COMMANDS TO THE TOOLS MENU

Command line commands and calls to batch files must be run from a command shell. You can add command line commands to the **Tools** menu and execute them from there.

To add a command, for example Backup, to the **Tools** menu to make a copy of the entire `project` directory to a network drive:

1. Choose **Tools>Configure Tools** to open the **Configure Tools** dialog box.
2. Type or browse to the **cmd.exe** command shell in the **Command** text box.
3. Type the command line command or batch file name in the **Argument** text box, for example:

   ```
   /C copy c:\project\*.* F:
   ```

   Alternatively, use an argument variable to allow relocatable paths:

   ```
   /C copy $PROJ_DIR$\*.* F:
   ```

   The argument text should be specified as:

   ```
   /C name
   ```

   where *name* is the name of the command or batch file you want to run.

   The `/C` option terminates the shell after execution, to allow the IDE to detect when the tool has finished.

USING AN EXTERNAL EDITOR

The **External Editor** options—available by choosing **Tools>Options>Editor>External Editor**—let you specify an external editor of your choice.

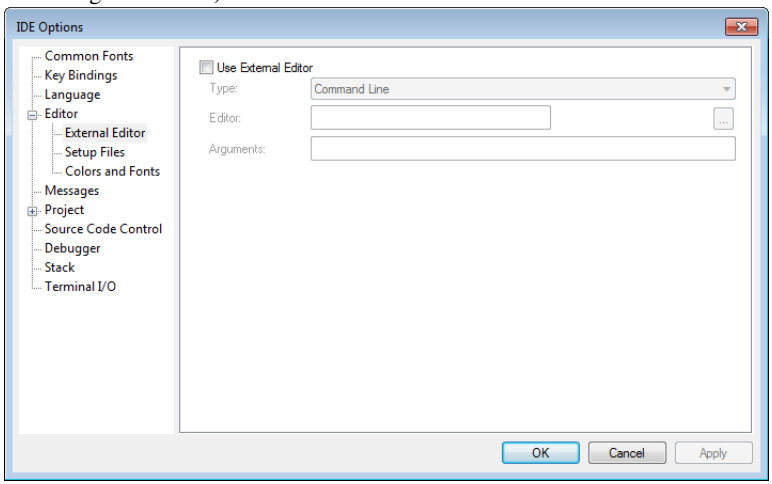
Note: While you are debugging using C-SPY, C-SPY will not use the external editor for displaying the current debug state. Instead, the built-in editor will be used.

To specify an external editor of your choice:

1. Select the option **Use External Editor**.
2. An external editor can be called in one of two ways, using the **Type** drop-down menu:
   - **Command Line** calls the external editor by passing command line parameters.
   - **DDE** calls the external editor by using DDE (Windows Dynamic Data Exchange).
3. If you use the command line, specify the command to pass to the editor, that is, the name of the editor and its path, for instance:

   ```
   C:\Windows\NOTEPAD.EXE
   ```

To send an argument to the external editor, type the argument in the **Arguments** field. For example, type `$FILE_PATH$` to start the editor with the active file (in editor, project, or messages windows).



Note: Options for Terminal I/O are only available when the C-SPY debugger is running.

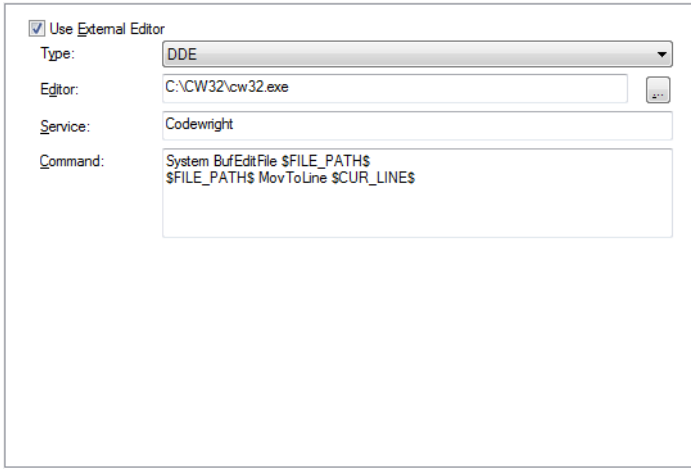
4 If you use DDE, specify the editor's DDE service name in the **Service** field. In the **Command** field, specify a sequence of command strings to send to the editor.

The service name and command strings depend on the external editor that you are using. Refer to the user documentation of your external editor to find the appropriate settings.

The command strings should be entered as:

```
DDE-Topic CommandString1
DDE-Topic CommandString2
```

as in this example, which applies to Codewright®:



The command strings used in this example will open the external editor with a dedicated file activated. The cursor will be located on the current line as defined in the context from where the file is open, for instance when searching for a string in a file, or when double-clicking an error message in the message window.

5 Click **OK**.

When you double-click a filename in the **Workspace** window, the file is opened by the external editor.

Variables can be used in the arguments. For more information about the argument variables that are available, see *Argument variables*, page 81.

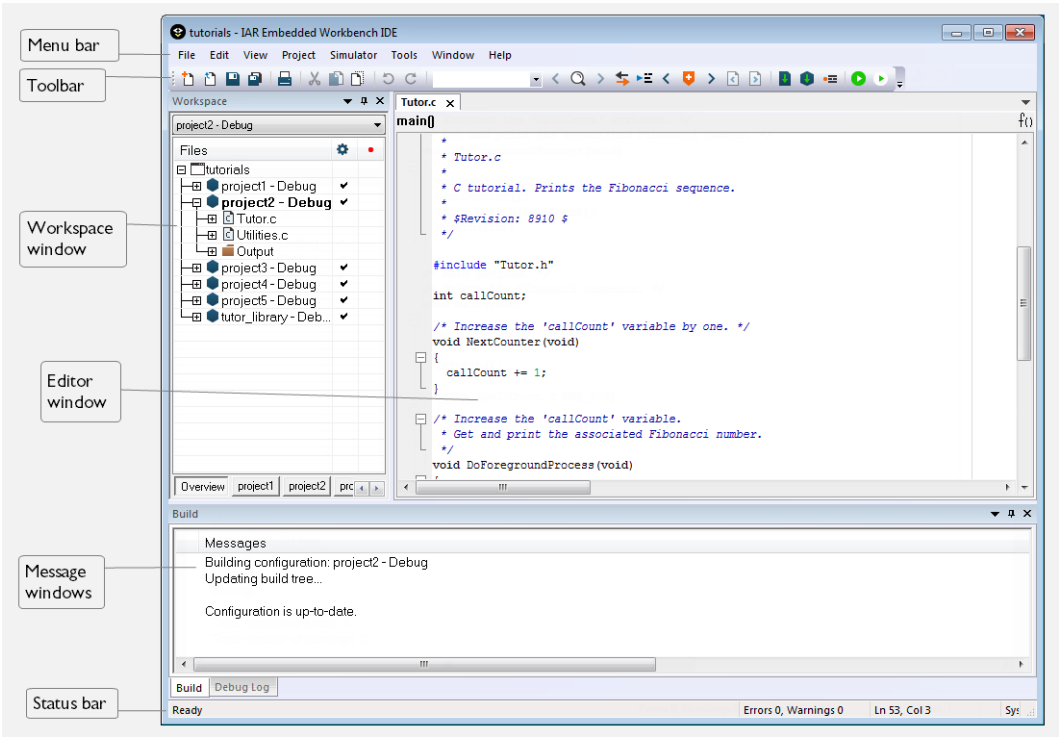
Reference information on the IDE

Reference information about:

- *IAR Embedded Workbench IDE window*, page 39
- *Customize dialog box*, page 44
- *Button Appearance dialog box*, page 46
- *Tool Output window*, page 47
- *Common Fonts options*, page 48
- *Key Bindings options*, page 49
- *Language options*, page 51
- *Editor options*, page 52

IAR Embedded Workbench IDE window

The main window of the IDE is displayed when you launch the IDE.



The figure shows the window and its default layout.

Menu bar

The menu bar contains:

File

Commands for opening source and project files, saving and printing, and exiting from the IDE.

Edit

Commands for editing and searching in editor windows and for enabling and disabling breakpoints in C-SPY.

View

Commands for opening windows and controlling which toolbars to display.

Project

Commands for adding files to a project, creating groups, and running the IAR Systems tools on the current project.

Simulator

Commands specific for the C-SPY simulator. This menu is only available when you have selected the simulator driver in the **Options** dialog box.

C-SPY hardware driver

Commands specific for the C-SPY hardware debugger driver you are using, in other words, the C-SPY driver that you have selected in the **Options** dialog box. For some IAR Embedded Workbench products, the name of the menu reflects the name of the C-SPY driver you are using and for others, the name of the menu is **Emulator**.

Tools

User-configurable menu to which you can add tools for use with the IDE.

Window

Commands for manipulating the IDE windows and changing their arrangement on the screen.

Help

Commands that provide help about the IDE.

For more information about each menu, see *Menus*, page 179.

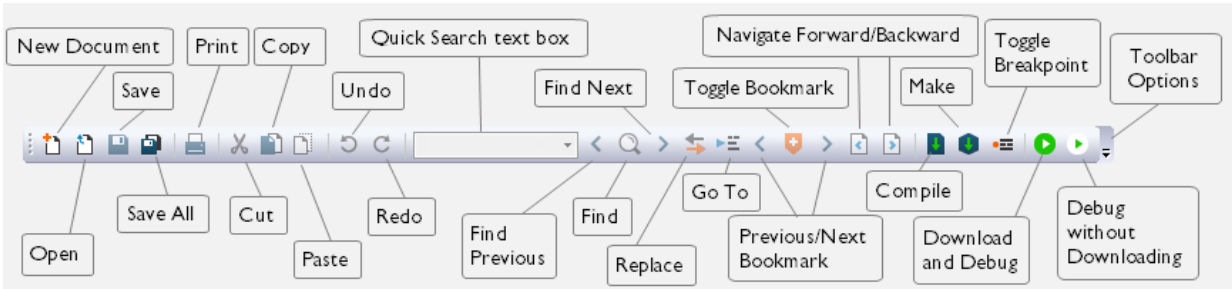
Toolbar

The buttons on the IDE toolbar provide shortcuts for the most useful commands on the IDE menus, and a text box for typing a string to do a quick search. For information about how to add and remove buttons on the toolbars, see *Using and customizing the IDE*, page 25.

For a description of any button, point to it with the mouse pointer. When a command is not available, the corresponding toolbar button is dimmed, and you will not be able to click it.

The toolbars are dockable; drag and drop to rearrange them.

This figure shows the menu commands corresponding to each of the toolbar buttons:



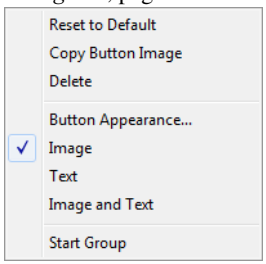
Note: When you start C-SPY, the **Download and Debug** button will change to a **Make and Restart Debugger** button, and the **Debug without Downloading** will change to a **Restart Debugger** button.

Toolbar Options

Click the **Toolbars Options** button to open the **Toolbars Options** menu.

Context menu

This context menu is available by right-clicking a toolbar button when the **Customize** dialog box is open. For information about how to open this dialog box, see *Customize dialog box*, page 44.



These commands are available:

Reset to Default

Hides the button icon and displays the name of the button instead.

Copy Button Image

Copies the button icon and stores the image on the clipboard.

Delete

Removes the button from the toolbar.

Button Appearance

Displays the **Button Appearance** dialog box, see *Button Appearance dialog box*, page 46.

Image

Displays the button only as an icon.

Text

Displays the button only as text.

Image and Text

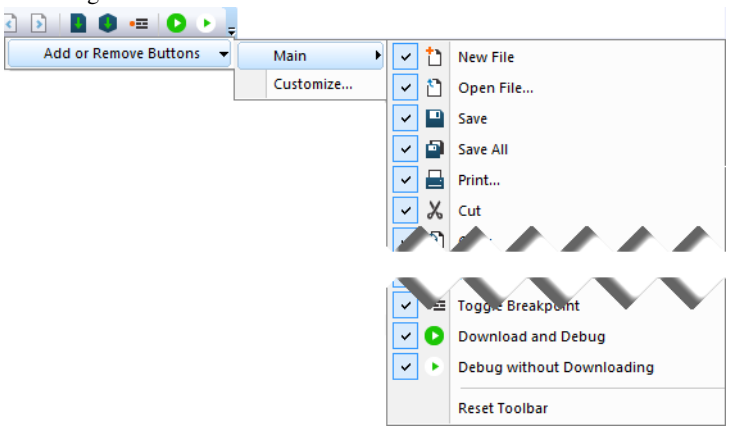
Displays the button both as an icon and as text.

Start Group

Inserts a delimiter to the left of the button.

Toolbars Options menu

This menu and its submenus are available by clicking the **Toolbars Options** button on the far right end of a toolbar:



These commands are available:

Add or Remove Buttons

Opens a submenu.

toolbar

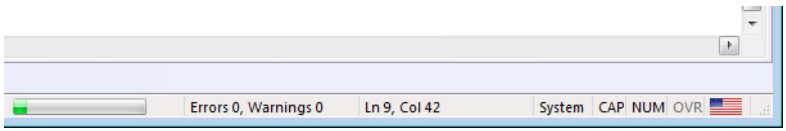
Opens a submenu that lists all command buttons on the toolbar. Select or deselect a checkbox to show/hide the button on the toolbar. Choose **Reset Toolbar** to restore the toolbar to its default appearance.

Customize

Displays the **Customize** dialog box, see *Customize dialog box*, page 44.

Status bar

The status bar at the bottom of the window can be enabled from the **View** menu.

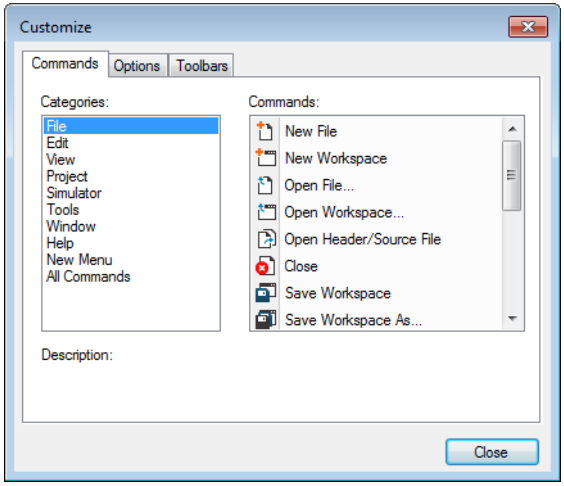


The status bar displays:

- Source browser progress information
- The number of errors and warnings generated during a build
- The position of the insertion point in the editor window. When you edit, the status bar shows the current line and column number containing the insertion point.
- The character encoding
- The state of the modifier keys Caps Lock, Num Lock, and Overwrite.
- If your product package is available in more languages than English, a flag in the corner shows the language version you are using. Click the flag to change the language. The change will take force the next time you launch the IDE.

Customize dialog box

The **Customize** dialog box is available by clicking the **Toolbars Options** button on the far right end of the a toolbar in the main IDE window and choosing **Add or Remove Buttons>Customize**.



These are the options on the **Commands** page of the **Customize** dialog box:

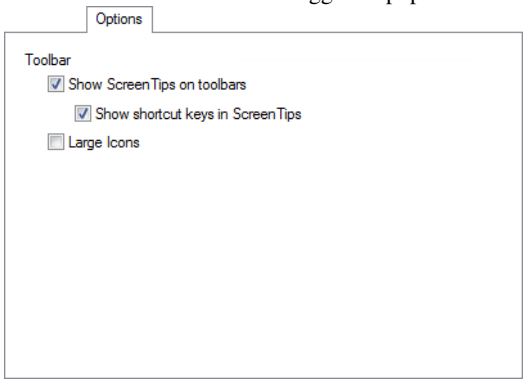
Categories

Lists the menus in the IDE. Select a menu name to make the commands on that menu available for adding as buttons to a toolbar. Select **New Menu** to add a custom drop-down menu to a toolbar.

Commands

Lists menu commands that can be dragged to one of the toolbars and inserted as buttons. If **New Menu** is the selected **Category**, the command **New Menu** can be dragged to a

toolbar to add a custom drop-down menu to the toolbar. Commands from the **Commands** list can then be dragged to populate the custom menu.



These are the options on the **Options** page of the **Customize** dialog box:

Show Screen Tips on toolbars

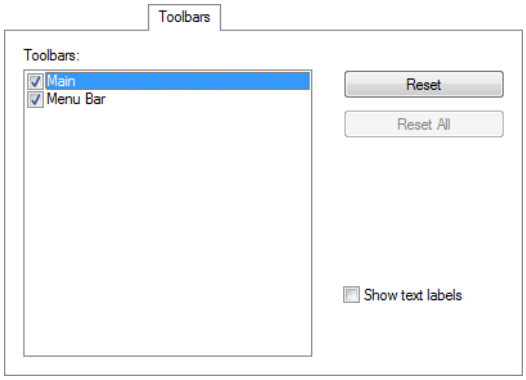
Enables tooltips for the buttons on the toolbars. The tooltips contain the display names of the buttons.

Show shortcut keys in Screen Tips

Includes the keyboard shortcut in the tooltip text for the buttons on the toolbar.

Large Icons

Increases the size of the buttons on the toolbars.



These are the options on the **Toolbars** page of the **Customize** dialog box:

Toolbars

Select/deselect a toolbar to show/hide it in the main IDE window. The menu bar cannot be hidden.

Reset

Restores the selected toolbar to its default appearance.

Reset All

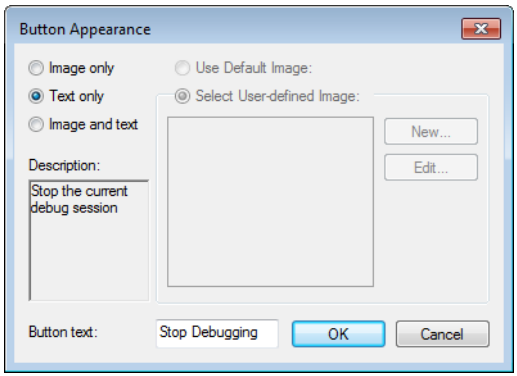
This button is disabled.

Show text labels

Displays the names of the buttons on the selected toolbar.

Button Appearance dialog box

The **Button Appearance** dialog box is available by right-clicking a toolbar button when the **Customize** dialog box is open and choosing **Button Appearance** from the context menu.



Use this dialog box to change the display name of a toolbar button.

Image only

This option has no effect.

Text only

Enables the text box **Button text**.

Image and text

Enables the text box **Button text**.

Use Default Image

This option is disabled.

Select User-defined Image

This option is disabled.

New

This button is disabled.

Edit

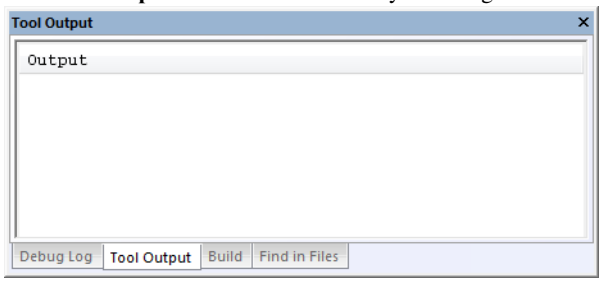
This button is disabled.

Button text

The display name of the toolbar button. Edit the text to change the name.

Tool Output window

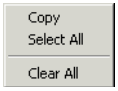
The **Tool Output** window is available by choosing **View>Messages>Tool Output**.



This window displays any messages output by user-defined tools in the **Tools** menu, provided that you have selected the **Redirect to Output Window** option in the **Configure Tools** dialog box, see *Configure Tools dialog box, page 74*. When opened, this window is, by default, grouped together with the other message windows.

Context menu

This context menu is available:



These commands are available:

Copy

Copies the contents of the window.

Select All

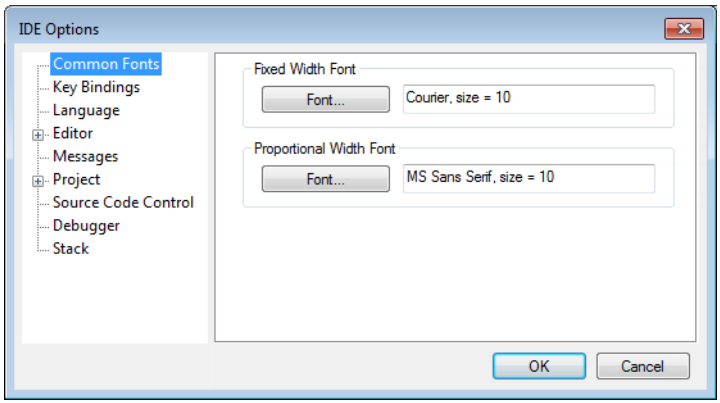
Selects the contents of the window.

Clear All

Deletes the contents of the window.

Common Fonts options

The **Common Fonts** options are available by choosing **Tools>Options**.



Use this page to configure the fonts used for all project windows except the editor windows.

For information about how to change the font in the editor windows, see *Editor Colors and Fonts options*, page 59.

Fixed Width Font

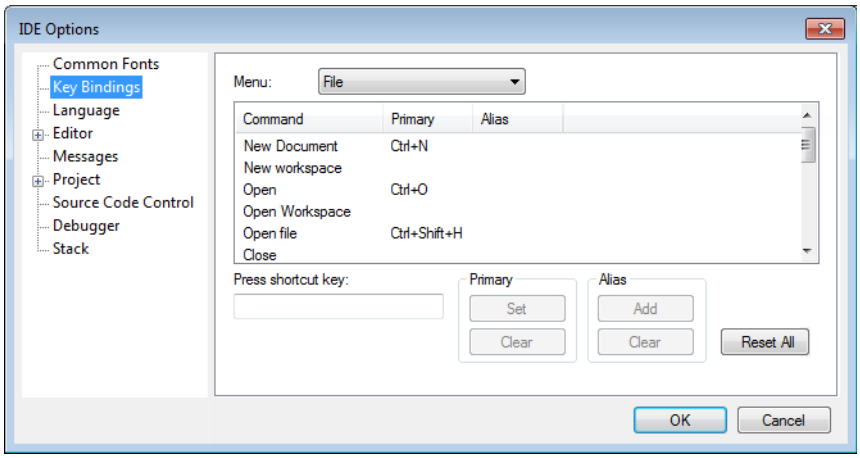
Selects which font to use in the **Disassembly**, **Register**, and **Memory** windows.

Proportional Width Font

Selects which font to use in all windows except the **Disassembly**, **Register**, **Memory**, and editor windows.

Key Bindings options

The **Key Bindings** options are available by choosing **Tools>Options**.



Use this page to customize the shortcut keys used for the IDE menu commands.

Menu

Selects the menu to be edited. Any currently defined shortcut keys for the selected menu are listed below the **Menu** drop-down list.

List of commands

Selects the menu command you want to configure your own shortcut keys for, from this list of all commands available on the selected menu.

Press shortcut key

Type the key combination you want to use as shortcut key for the selected command. You cannot set or add a shortcut if it is already used by another command.

Primary

Choose to:

Set

Saves the key combination in the **Press shortcut key** field as a shortcut for the selected command in the list.

Clear

Removes the listed primary key combination as a shortcut for the selected command in the list.

The new shortcut will be displayed next to the command on the menu.

Alias

Choose to:

Add

Saves the key combination in the **Press shortcut key** field as an alias—a hidden shortcut—for the selected command in the list.

Clear

Removes the listed alias key combination as a shortcut for the selected command in the list.

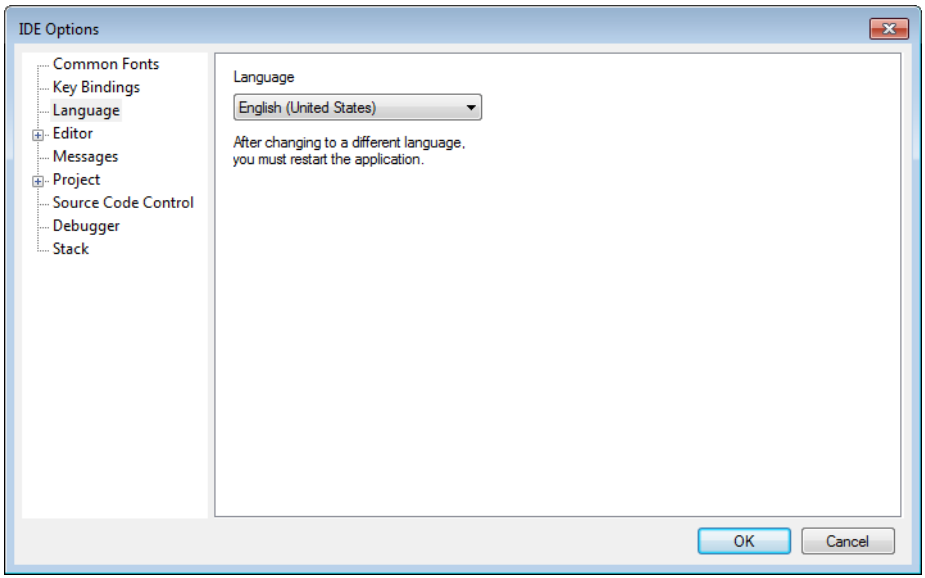
The new shortcut will be not displayed next to the command on the menu.

Reset All

Reverts the shortcuts for all commands to the factory settings.

Language options

The **Language** options are available by choosing **Tools>Options**.



Use this page to specify the language to be used in windows, menus, dialog boxes, etc.

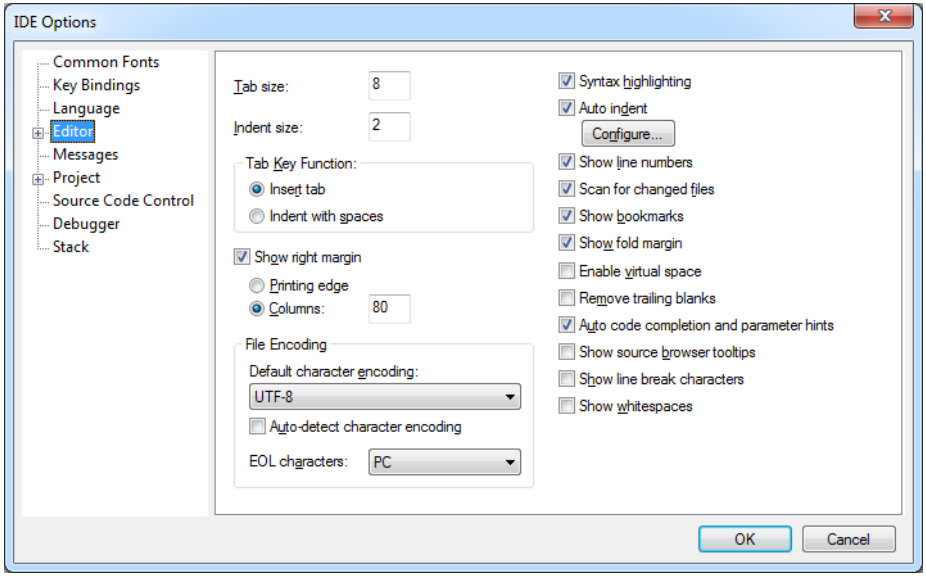
Language

Selects the language to be used. The available languages depend on your product package, **English (United States)** and **Japanese (Japan)**.

Note: If you have installed IAR Embedded Workbench for several different toolchains in the same directory, the IDE might be in mixed languages if the toolchains are available in different languages.

Editor options

The **Editor** options are available by choosing **Tools>Options**.



Use this page to configure the editor. For more information about the editor, see *Editing*, page 125.

Tab size

Specify the width of a tab character, in terms of character spaces.

Indent size

Specify the number of spaces to be used when tabulating with an indentation.

Tab Key Function

Controls what happens when you press the Tab key. Choose between:

Insert tab

Inserts a tab character when the Tab key is pressed.

Indent with spaces

Inserts an indentation (space characters) when the Tab key is pressed.

Show right margin

Displays the area of the editor window outside the right margin as a light gray field. If this option is selected, you can set the width of the text area between the left margin and the right margin. Choose to set the width based on:

Printing edge

Bases the width on the printable area, which is taken from the general printer settings.

Columns

Bases the width on the number of columns.

File Encoding

Controls file encoding. Choose between:

Default character encoding

Selects the character encoding to be used by default for new files. Choose between:

System (uses the Windows settings)
Western European
UTF-8
Japanese (Shift-JIS)
Chinese Simplified (GB2312)
Chinese Traditional (Big5)
Korean (Unified Hangul Code)
Arabic
Central European
Greek
Hebrew
Thai
Baltic
Russian
Vietnamese

Note that if you have specified a character encoding from the editor window context menu, that encoding will override this setting for the specific document.

Auto-detect character encoding

Detects automatically which character encoding that should be used when you open an existing document.

EOL characters

Selects which line break character to use when editor documents are saved. Choose between:

PC (default), Windows and DOS end of line characters.

UNIX, UNIX end of line characters.

Preserve, the same end of line character as the file had when it was opened, either PC or UNIX. If both types or neither type are present in the opened file, PC end of line characters are used.

Syntax highlighting

Makes the editor display the syntax of C or C++ applications in different text styles.

For more information about syntax highlighting, see *Editor Colors and Fonts options*, page 59 and *Syntax coloring*, page 131.

Auto indent

Makes the editor indent the new line automatically when you press Return. For C/C++ source files, click the **Configure** button to configure the automatic indentation, see *Configure Auto Indent dialog box*, page 55. For all other text files, the new line will have the same indentation as the previous line.

Show line numbers

Makes the editor display line numbers in the editor window.

Scan for changed files

Makes the editor reload files that have been modified by another tool.

If a file is open in the IDE, and the same file has concurrently been modified by another tool, the file will be automatically reloaded in the IDE. However, if you already started to edit the file, you will be prompted before the file is reloaded.

Show bookmarks

Makes the editor display a column on the left side in the editor window, with icons for compiler errors and warnings, **Find in Files** results, user bookmarks, and breakpoints.

Show fold margin

Makes the editor display the fold margin in the left side of the editor window. For more information, see *Code folding*, page 128.

Enable virtual space

Allows the insertion point to move outside the text area.

Remove trailing blanks

Removes trailing blanks from files when they are saved to disk. Trailing blanks are blank spaces between the last non-blank character and the end of line character.

Auto code completion and parameter hints

Enables code completion and parameter hints. For more information, see *Editing a file*, page 126.

Show source browser tooltips

Toggles the display of detailed information about the identifier that the cursor currently hovers over.

Show line break characters

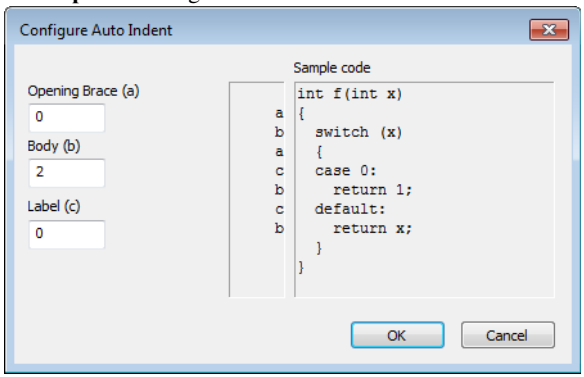
Toggles the display of carriage return and line feed characters in the editor window.

Show whitespaces

Toggles the display of period (.) characters for single blank spaces and arrow (–>) characters for tabs in the editor window.

Configure Auto Indent dialog box

The **Configure Auto Indent** dialog box is available from the **Editor** category in the **IDE Options** dialog box.



Use this dialog box to configure the editor's automatic indentation of C/C++ source code.

For more information about indentation, see *Indenting text automatically*, page 127.

Opening Brace (a)

Specify the number of spaces used for indenting an opening brace.

Body (b)

Specify the number of additional spaces used for indenting code after an opening brace, or a statement that continues onto a second line.

Label (c)

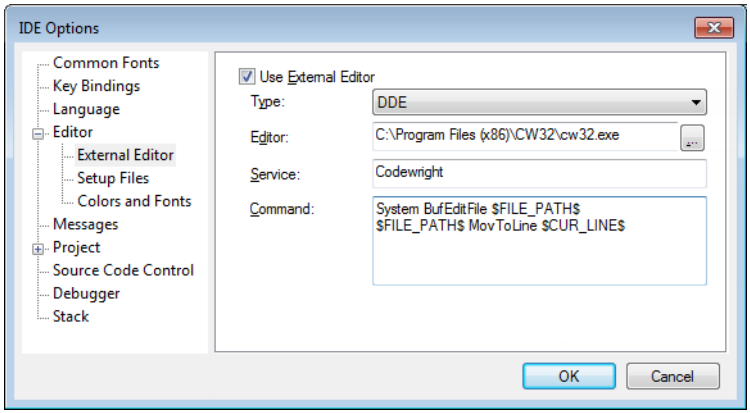
Specify the number of additional spaces used for indenting a label, including case labels.

Sample code

This area reflects the settings made in the text boxes for indentation. All indentations are relative to the preceding line, statement, or other syntactic structures.

External Editor options

The **External Editor** options are available by choosing **Tools>Options**.



Use this page to specify an external editor of your choice.

Note: The contents of this dialog box depends on the setting of the **Type** option.

See also *Using an external editor*, page 35.

Use External Editor

Enables the use of an external editor.

Type

Selects the type of interface. Choose between:

- **Command Line**
- **DDE** (Windows Dynamic Data Exchange).

Editor

Specify the filename and path of your external editor. A browse button is available.

Arguments

Specify any arguments to be passed to the editor. This is only applicable if you have selected **Command Line** as the interface type.

Service

Specify the DDE service name used by the editor. This is only applicable if you have selected **DDE** as the interface type.

The service name depends on the external editor that you are using. Refer to the user documentation of your external editor to find the appropriate settings.

Command

Specify a sequence of command strings to be passed to the editor. The command strings should be typed as:

```
DDE-Topic CommandString1
DDE-Topic CommandString2
```

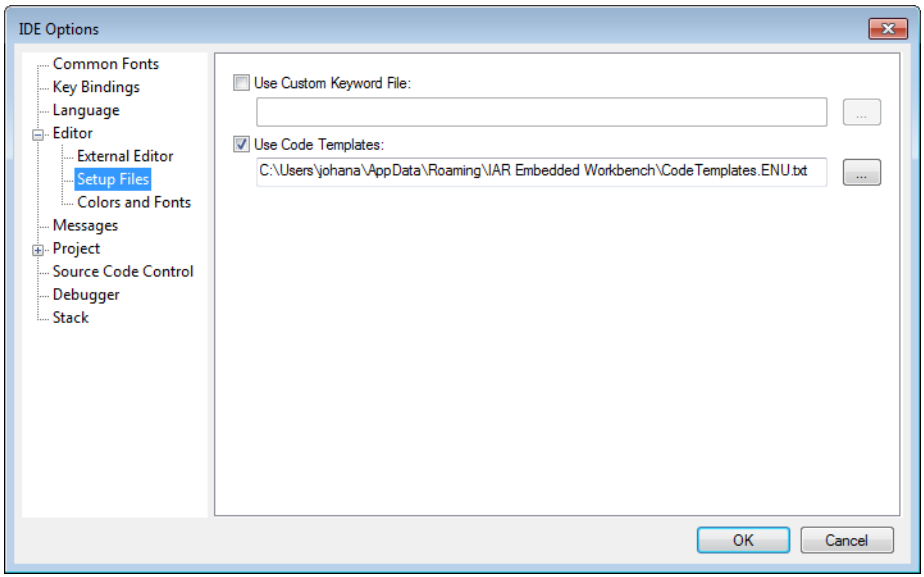
This is only applicable if you have selected **DDE** as the interface type.

The command strings depend on the external editor that you are using. Refer to the user documentation of your external editor to find the appropriate settings.

Note: You can use variables in arguments, see *Argument variables*, page 81.

Editor Setup Files options

The **Editor Setup Files** options are available by choosing **Tools>Options**.



Use this page to specify setup files for the editor.

Use Custom Keyword File

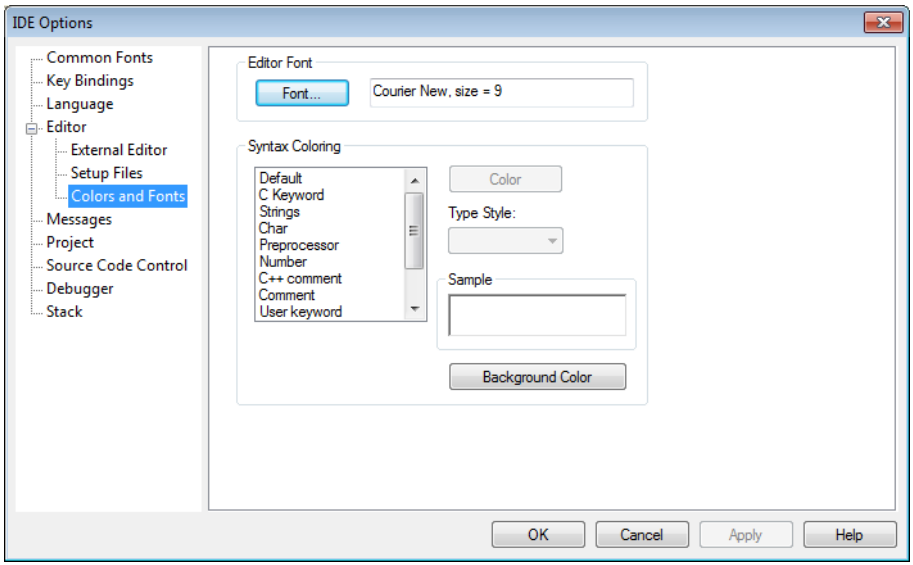
Specify a text file containing keywords that you want the editor to highlight. For information about syntax coloring, see *Syntax coloring*, page 131.

Use Code Templates

Specify a text file with code templates that you can use for inserting frequently used code in your source file. For information about using code templates, see *Using and adding code templates*, page 130.

Editor Colors and Fonts options

The **Editor Colors and Fonts** options are available by choosing **Tools>Options**.



Use this page to specify the colors and fonts used for text in the editor windows. The keywords controlling syntax highlighting for assembler and C or C++ source code are specified in the files `syntax_icc.cfg` and `syntax_asm.cfg`, respectively. These files are located in the `rh850\config` directory.

Editor Font

Click the **Font** button to open the standard **Font** dialog box where you can choose the font and its size to be used in editor windows.

Syntax Coloring

Selects a syntax element in the list and sets the color and style for it:

Color

Lists colors to choose from. Choose **Custom** from the list to define your own color.

Type Style

Select **Normal**, **Bold**, or **Italic** style for the selected element.

Sample

Displays the current appearance of the selected element.

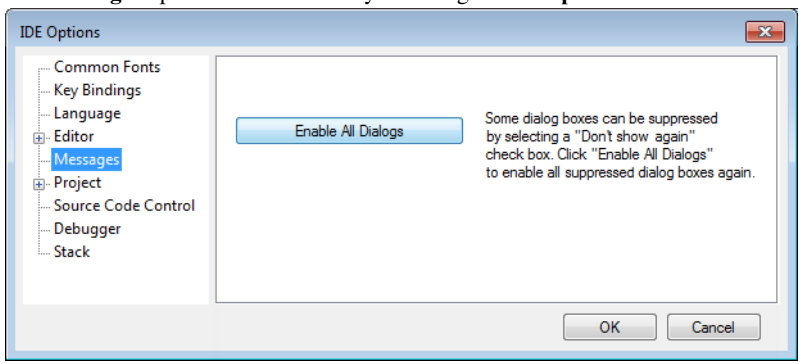
Background Color

Click to set the background color of the editor window.

Note: The **User keyword** syntax element refers to the keywords that you have listed in the custom keyword file, see *Editor Setup Files options*, page 58.

Messages options

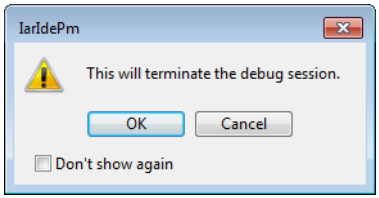
The **Messages** options are available by choosing **Tools>Options**.



Use this page to re-enable suppressed dialog boxes.

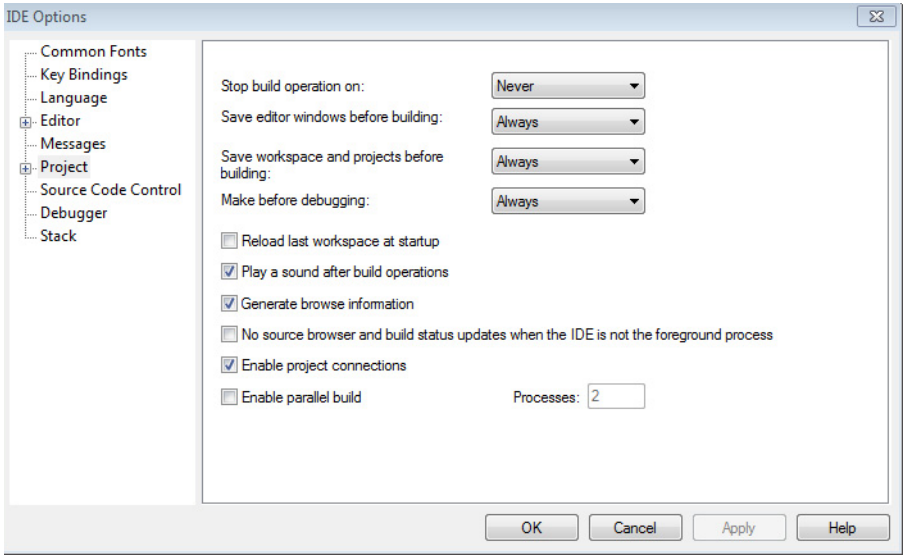
Enable All Dialogs

Enables all dialog boxes you have suppressed by selecting a **Don't show again** check box, for example:



Project options

The **Project** options are available by choosing **Tools>Options**.



Use this page to set options for the **Make** and **Build** commands.

Stop build operation on

Selects when the build operation should stop. Choose between:

Never

Never stops.

Warnings

Stops on warnings and errors.

Errors

Stops on errors.

Save editor windows before building

Selects when the editor windows should be saved before a build operation. Choose between:

Never

Never saves.

Ask

Prompts before saving.

Always

Always saves before Make or Build.

Save workspace and projects before building

Selects when a workspace and included projects should be saved before a build operation. Choose between:

Never

Never saves.

Ask

Prompts before saving.

Always

Always saves before Make or Build.

Make before debugging

Selects when a Make operation should be performed as you start a debug session. Choose between:

Never

Never performs a Make operation before a debug session.

Ask

Prompts before performing a Make operation.

Always

Always performs a Make operation before a debug session.

Reload last workspace at startup

Loads the last active workspace automatically the next time you start the IAR Embedded Workbench IDE.

Play a sound after build operations

Plays a sound when the build operations are finished.

Generate browse information

Enables the generation of source browse information to display in the **Source Browser** window, see *Source Browser window*, page 157.

No source browser and build status updates when the IDE is not the foreground process

Halts the source browser when the IDE is not the foreground process. This also means that the build status is no longer updated in the **Workspace** window. This option is useful, for example, if you are using a laptop and want to reduce power consumption.

Enable project connections

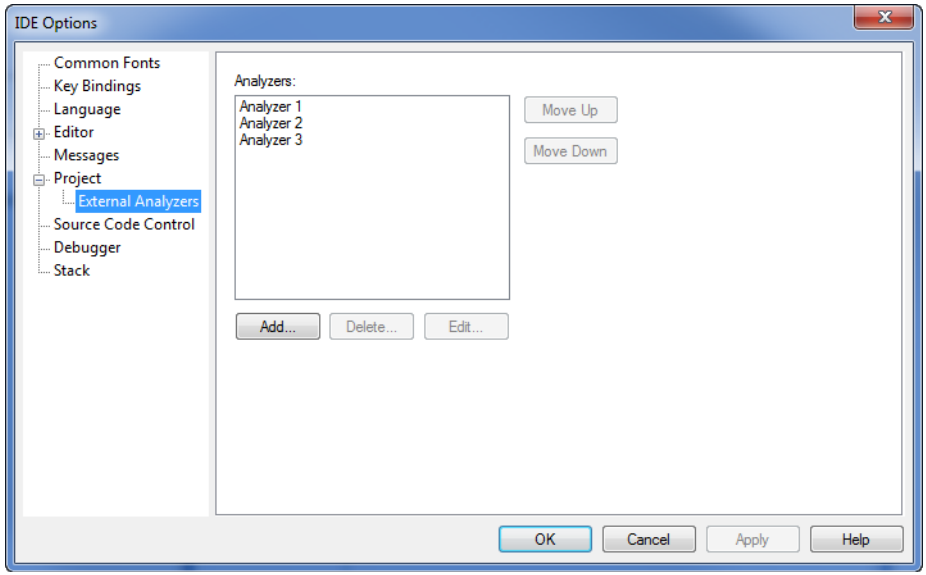
Enables the support for setting up live project connections, see *Add Project Connection dialog box*, page 105.

Enable parallel build

Enables the support for parallel build. The compiler runs in several parallel processes to better use the available cores in the CPU. In the **Processes** text box, specify the number of processes you want to use. Using all available cores might result in a less responsive IDE.

External Analyzers options

The **External Analyzers** options are available by choosing **Tools>Options**.



Use this page to add an external analyzer to the standard build toolchain. External analyzers operate on C/C++ source code in the user project. Header files or assembler source code files are not analyzed.

For more information, see *Getting started using external analyzers*, page 31.

Analyzers

Lists the external analyzers that you have added to the standard build toolchain.

Move Up

Moves the analyzer you have selected in the list one step up. This order is reflected on the **Project** menu.

Move Down

Moves the analyzer you have selected in the list one step down. This order is reflected on the **Project** menu.

Add

Displays the **External Analyzer** dialog box where you can add a new analyzer to the toolchain and configure the invocation of the analyzer.

Delete

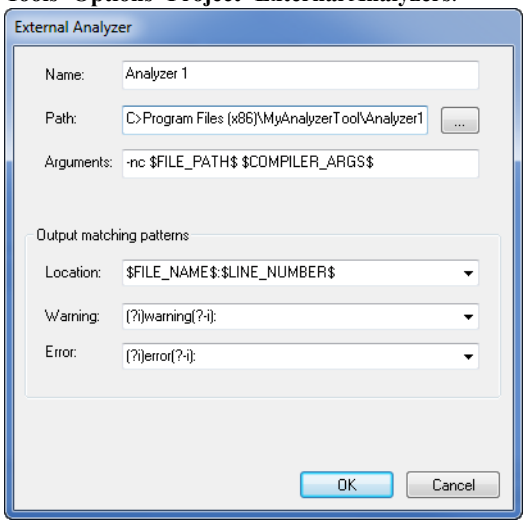
Deletes the selected analyzer from the list of analyzers.

Edit

Displays the **External Analyzer** dialog box where you can edit the invocation details of the selected analyzer.

External Analyzer dialog box

The **External Analyzer** dialog box is available by choosing **Tools>Options>Project>External Analyzers**.



Use this dialog box to configure the invocation of the external analyzer that you want to add to the standard build toolchain.

External analyzers operate on C/C++ source code in the user project. Header files or assembler source code files are not analyzed.

For more information, see *Getting started using external analyzers*, page 31.

Name

Specify the name of the external analyzer. Note that the name must be unique.

Path

Specify the path to the analyzer's executable file. A browse button is available.

Arguments

Specify the arguments that you want to pass to the analyzer.

Note that you can use argument variables for specifying the arguments, see *Argument variables*, page 81.

Location

Specify a regular expression used for finding source file locations. The regular expression is applied to each output line which will appear as text in the **Build Log** window. You can double-click a line that matches the regular expression you specify.

You can use the argument variables `$FILE_NAME$`, `$LINE_NUMBER$`, and `$COLUMN_NUMBER$` to identify a filename, line number, and column number, respectively. Choose one of the predefined expressions:

\"?$FILE_NAME$\"?:$LINE_NUMBER$

Will, for example, match a location of the form `file.c:17`.

\"?$FILE_NAME$\"? +$LINE_NUMBER$

Will, for example, match a location of the form `file.c17`.

\"?$FILE_NAME$\"?

Will, for example, match a location of the form `file.c`.

Alternatively, you can specify your own expression. For example, the regular expression `Msg: $FILE_NAME$ @ $LINE_NUMBER$`, when applied to the output string `Msg:MySourceFile.c @ 32`, will identify the file as `MySourceFile.c`, and the line number as 32.

Warning

Any output line that matches this expression is tagged with the warning symbol.

For example, the expression `(?i)warning(?-i):` will identify any line that contains the string `warning:` (regardless of case) as a warning.

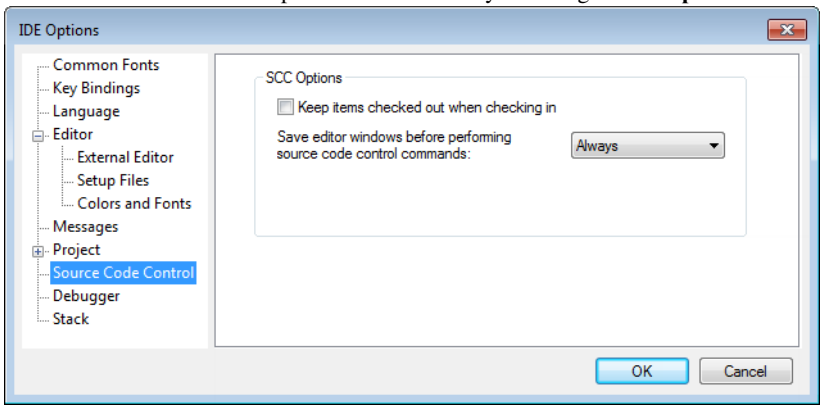
Error

Any output line that matches this expression is tagged with the error symbol. Errors have precedence over warnings.

For example, the expression `(?i)error(?-i):` will identify any line that contains the string `error:` (regardless of case) as an error.

Source Code Control options (deprecated)

The **Source Code Control** options are available by choosing **Tools>Options**.



Use this page to configure the interaction between an IAR Embedded Workbench project and an SCC project.

Note: This is a deprecated feature which is not supported for new projects.

Keep items checked out when checking in

Determines the default setting for the option **Keep Checked Out** in the **Check In Files** dialog box.

Save editor windows before performing source code control commands

Determines whether editor windows should be saved before you perform any source code control commands. Choose between:

Never

Never saves editor windows before performing any source code control commands.

Ask

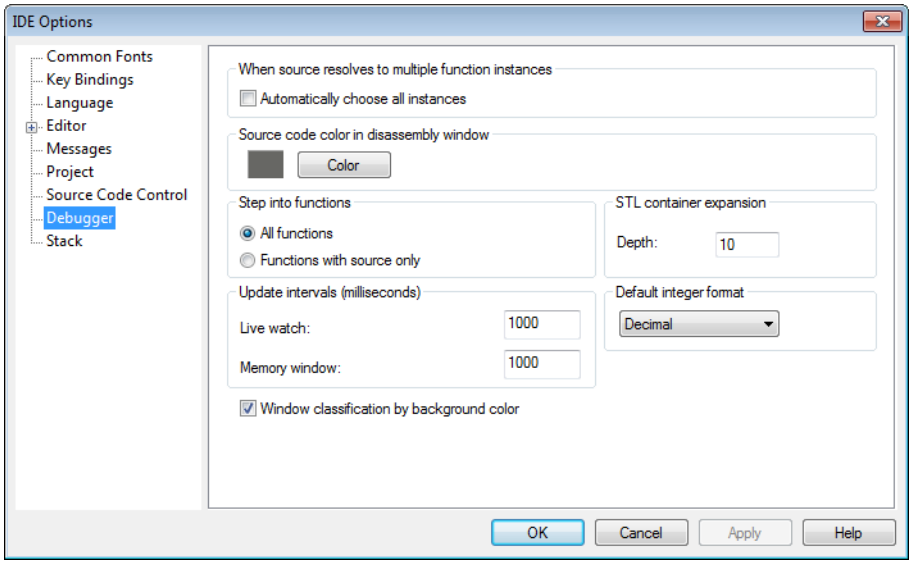
Prompts before performing any source code control commands.

Always

Always saves editor windows before performing any source code control commands.

Debugger options

The **Debugger** options are available by choosing **Tools>Options**.



Use this page to configure the debugger environment.

When source resolves to multiple function instances

Some source code corresponds to multiple code instances, for example template code. When specifying a source location in such code, for example when setting a source breakpoint, you can make C-SPY act on all instances or a subset of instances. Use the **Automatically choose all instances** option to let C-SPY act on all instances without asking first.

Source code color in disassembly window

Click the **Color** button to select the color for source code in the **Disassembly** window. To define your own color, choose **Custom** from the list.

Step into functions

Controls the behavior of the **Step Into** command. Choose between:

All functions

Makes the debugger step into all functions.

Functions with source only

Makes the debugger step only into functions for which the source code is known. This helps you avoid stepping into library functions or entering disassembly mode debugging.

STL container expansion

Specify how many elements that are shown initially when a container value is expanded in, for example, the **Watch** window.

Update intervals

Specify how often the contents of the **Live Watch** window and the **Memory** window are updated in milliseconds.

These text boxes are only available if the C-SPY driver you are using has access to the target system memory while executing your application.

Default integer format

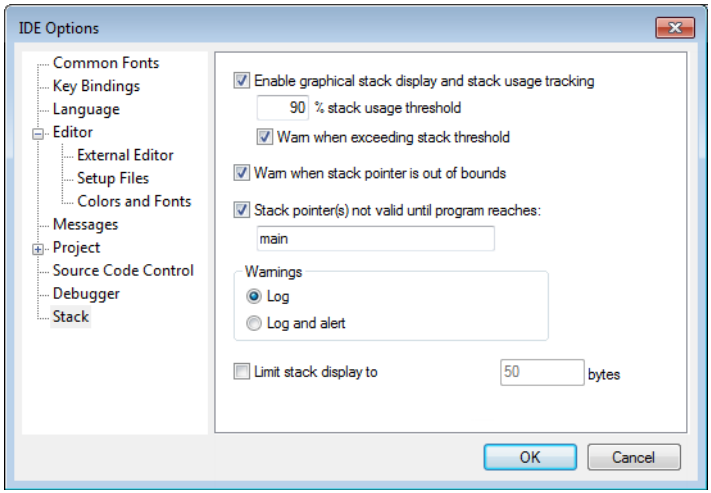
Selects the default integer format in the **Watch**, **Locals**, and related windows.

Window classification by background color

Toggles background colors in some C-SPY windows on or off. Colors are used for differentiating types of windows, for example, all interrupt-related windows have one background color, and all watch-related windows have another color, etc.

Stack options

The **Stack** options are available by choosing **Tools>Options** or from the context menu in the **Memory** window.



Use this page to set options specific to the **Stack** window.

Enable graphical stack display and stack usage tracking

Enables the graphical stack bar available at the top of the **Stack** window. It also enables detection of stack overflows. For more information about the stack bar and the information it provides, see the *C-SPY® Debugging Guide for RH850*.

% stack usage threshold

Specify the percentage of stack usage above which C-SPY should issue a warning for stack overflow.

Warn when exceeding stack threshold

Makes C-SPY issue a warning when the stack usage exceeds the threshold specified in the **% stack usage threshold** option.

Warn when stack pointer is out of bounds

Makes C-SPY issue a warning when the stack pointer is outside the stack memory range.

Stack pointer(s) not valid until program reaches

Specify a *location* in your application code from where you want the stack display and verification to occur. The **Stack** window will not display any information about stack usage until execution has reached this location.

By default, C-SPY will not track the stack usage before the `main` function. If your application does not have a `main` function, for example, if it is an assembler-only project, you should specify your own start label. If this option is selected, after each reset C-SPY keeps a breakpoint on the given location until it is reached.

Typically, the stack pointer is set up in the system initialization code `cstartup`, but not necessarily from the first instruction. Select this option to avoid incorrect warnings or misleading stack display for this part of the application.

Warnings

Selects where warnings should be issued. Choose between:

Log

Warnings are issued in the **Debug Log** window.

Log and alert

Warnings are issued in the **Debug Log** window and as alert dialog boxes.

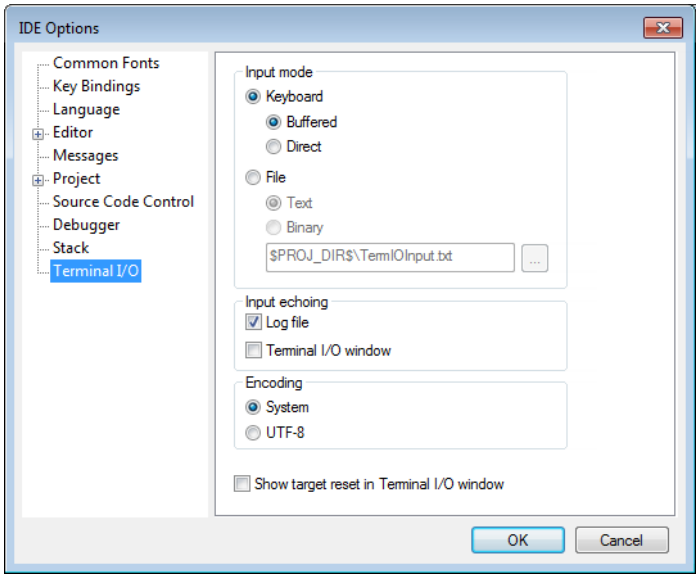
Limit stack display to

Limits the amount of memory displayed in the **Stack** window by specifying a number of bytes, counting from the stack pointer. This can be useful if you have a big stack or if you are only interested in the topmost part of the stack. Using this option can improve the **Stack** window performance, especially if reading memory from the target system is slow. By default, the **Stack** window shows the whole stack, or in other words, from the stack pointer to the bottom of the stack. If the debugger cannot determine the memory range for the stack, the byte limit is used even if the option is not selected.

Note: The **Stack** window does not affect the execution performance of your application, but it might read a large amount of data to update the displayed information when the execution stops.

Terminal I/O options

The **Terminal I/O** options are available by choosing **Tools>Options** when C-SPY is running.



Use this page to configure the C-SPY terminal I/O functionality.

Input mode

Controls how the terminal I/O input is read.

Keyboard Makes the input characters be read from the keyboard. Choose between:

Buffered: Buffers input characters.

Direct: Does not buffer input characters.

File Makes the input characters be read from a file. Choose between:

Text: Reads input characters from a text file.

Binary: Reads input characters from a binary file.

A browse button is available for locating the input file.

Input echoing

Determines whether to echo the input characters and where to echo them. Choose between:

Log file

Echoes the input characters in the Terminal I/O log file. Requires that you have enabled the option **Debug>Logging>Enable log file**.

Terminal I/O window

Echoes the input characters in the **Terminal I/O** window.

Encoding

Determines the encoding used for terminal input and output. Choose between:

System

Uses the Windows settings.

UTF-8

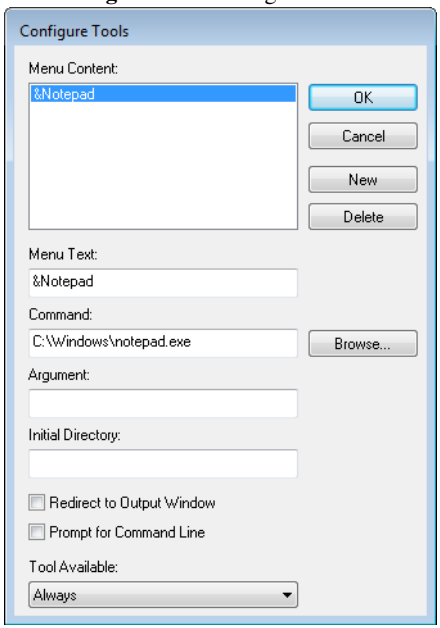
Uses the UTF-8 encoding.

Show target reset in Terminal I/O window

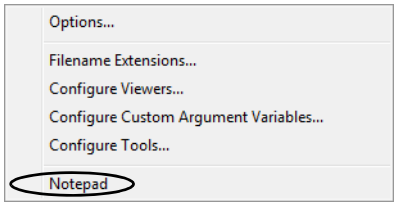
Displays a message in the C-SPY **Terminal I/O** window when the target resets.

Configure Tools dialog box

The **Configure Tools** dialog box is available from the **Tools** menu.



Use this dialog box to specify a tool of your choice to add to the **Tools** menu, for example Notepad:



Note: If you intend to add an external tool to the standard build toolchain, see *Extending the toolchain*, page 109.

You can use variables in the arguments, which allows you to set up useful tools such as interfacing to a command line revision control system, or running an external tool on the selected file.

To add a command line command or batch file to the Tools menu:

1. Type or browse to the `cmd.exe` command shell in the **Command** text box.

2 Type the command line command or batch file name in the **Argument** text box as:

`/C name`

where *name* is the name of the command or batch file you want to run.

The `/C` option terminates the shell after execution, to allow the IDE to detect when the tool has finished.

For an example, see *Adding command line commands to the Tools menu*, page 35.

New

Creates a stub for a new menu command for you to configure using this dialog box.

Delete

Removes the command selected in the **Menu Content** list.

Menu Content

Lists all menu commands that you have defined.

Menu Text

Specify the name of the menu command. If you add the `&` sign anywhere in the name, the following letter, `N` in this example, will appear as the mnemonic key for this command. The text you specify will be reflected in the **Menu Content** list.

Command

Specify the tool and its path, to be run when you choose the command from the menu. A browse button is available.

Argument

Optional: Specify an argument for the command.

Initial Directory

Specify an initial working directory for the tool.

Redirect to Output window

Makes the IDE send any console output from the tool to the **Tool Output** page in the message window. Tools that are launched with this option cannot receive any user input, for instance input from the keyboard.

Tools that require user input or make special assumptions regarding the console that they execute in, will *not* work at all if launched with this option.

Prompt for Command Line

Makes the IDE prompt for the command line argument when the command is chosen from the **Tools** menu.

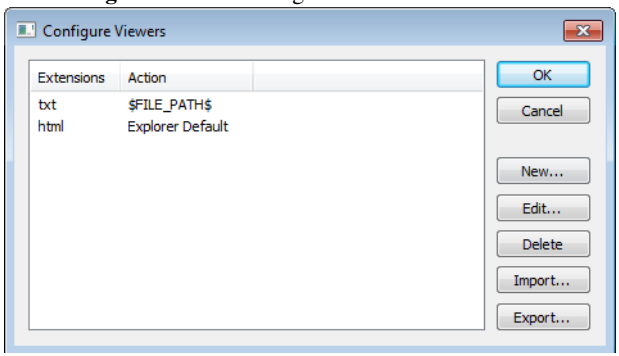
Tool Available

Specifies in which context the tool should be available. Choose between:

- **Always**
- **When debugging**
- **When not debugging**.

Configure Viewers dialog box

The **Configure Viewers** dialog box is available from the **Tools** menu.



This dialog box lists overrides to the default associations between the document formats that IAR Embedded Workbench can handle and viewer applications.

Display area

This area contains these columns:

Extensions

Explicitly defined filename extensions of document formats that IAR Embedded Workbench can handle.

Action

The viewer application that is used for opening the document type. **Explorer Default** means that the default application associated with the specified type in Windows Explorer is used.

New

Displays the **Edit Viewer Extensions** dialog box, see *Edit Viewer Extensions dialog box*, page 77.

Edit

Displays the **Edit Viewer Extensions** dialog box, see *Edit Viewer Extensions dialog box*, page 77.

Delete

Removes the association between the selected filename extensions and the viewer application.

Import

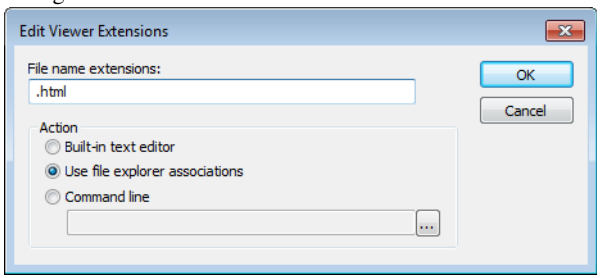
Opens a file browser where you can locate and import a File Viewer Association file in XML format. This file contains associations between document formats and viewer applications.

Export

Displays a standard **Save As** dialog box to let you save the current associations between document formats and viewer applications in the **Configure Viewers** dialog box to a file in XML format.

Edit Viewer Extensions dialog box

The **Edit Viewer Extensions** dialog box is available from the **Configure Viewers** dialog box.



Use this dialog box to specify how to open a new document type or edit the setting for an existing document type.

File name extensions

Specify the filename extension for the document type—including the separating period (.).

Action

Selects how to open documents with the filename extension specified in the **Filename extensions** text box. Choose between:

Built-in text editor

Opens all documents of the specified type with the IAR Embedded Workbench text editor.

Use file explorer associations

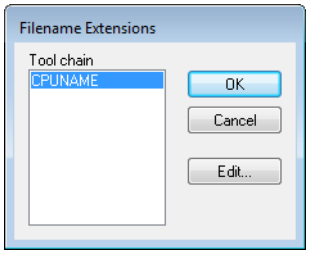
Opens all documents of the specified type with the default application associated with the specified type in Windows Explorer.

Command line

Opens all documents of the specified type with the viewer application you type or browse your way to. You can give any command line options you would like to the tool, for instance, type `$FILE_PATH$` after the path to the viewer application to start the viewer with the active file (in editor, project, or messages windows).

Filename Extensions dialog box

The **Filename Extensions** dialog box is available from the **Tools** menu.



Use this dialog box to customize the filename extensions recognized by the build tools. This is useful if you have many source files with different filename extensions.

Toolchain

Lists the toolchains for which you have an IAR Embedded Workbench installed on your host computer. Select the toolchain you want to customize filename extensions for.

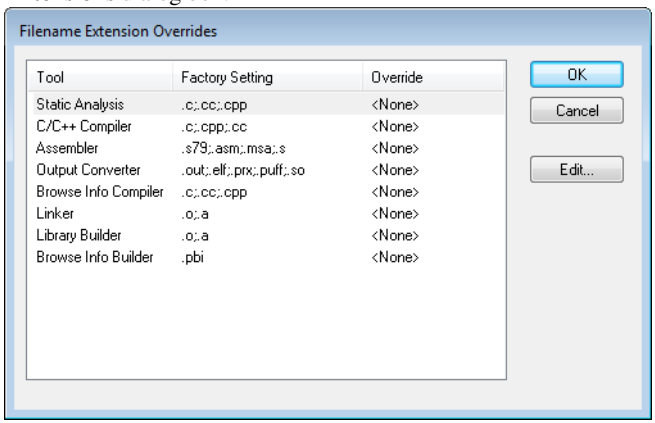
Note the * character indicates user-defined overrides. If there is no * character, factory settings are used.

Edit

Displays the **Filename Extension Overrides** dialog box, see *Filename Extension Overrides dialog box*, page 79.

Filename Extension Overrides dialog box

The **Filename Extension Overrides** dialog box is available from the **Filename Extensions** dialog box.



This dialog box lists filename extensions recognized by the build tools.

Display area

This area contains these columns:

Tool

The available tools in the build chain.

Factory Setting

The filename extensions recognized by default by the build tool.

Override

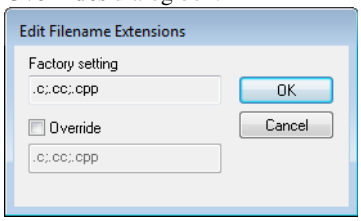
The filename extensions recognized by the build tool if there are overrides to the default setting.

Edit

Displays the **Edit Filename Extensions** dialog box for the selected tool.

Edit Filename Extensions dialog box

The **Edit File Extensions** dialog box is available from the **Filename Extension Overrides** dialog box.



This dialog box lists the filename extensions recognized by the IDE and lets you add new filename extensions.

Factory setting

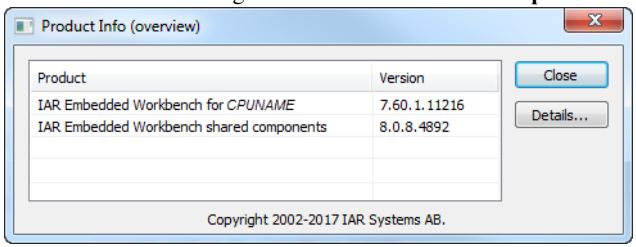
Lists the filename extensions recognized by default.

Override

Specify the filename extensions you want to be recognized. Extensions can be separated by commas or semicolons, and should include the leading period.

Product Info dialog box

The **Product Info** dialog box is available from the **Help** menu.



This dialog box lists the version number of your IAR Embedded Workbench product installation and the shared components.

Note: The initial digit of the version number of the shared components (in this screen shot `8`) is reflected by the default installation directory `x:\Program Files\IAR Systems\Embedded Workbench 8.n\`.

Details

Opens a dialog box which lists the version numbers of the various components part of your product installation.

Argument variables

You can use argument variables for paths and arguments, for example when you specify include paths in the **Options** dialog box or whenever there is a need for a macro-like expansion that depends on the current context, for example in arguments to tools. You can use a wide range of predefined argument variables as well as create your own, see *Configure Custom Argument Variables dialog box*, page 83. These are the predefined argument variables:

Variable	Description
`$COMPILER_ARGS$`	All compiler options except for the filename that is used when compiling using the compiler. Note that this argument variable is restricted to the **Arguments** text box in the **External Analyzer** dialog box.
`$CONFIG_NAME$`	The name of the current build configuration, for example Debug or Release.
`$CUR_DIR$`	Current directory
`$CUR_LINE$`	Current line
`$DATE$`	Today's date, formatted according to the current locale. Note that this might make the variable unsuited for use in file paths.
`$EW_DIR$`	Top directory of IAR Embedded Workbench, for example `c:\Program Files\IAR Systems\Embedded Workbench N.n`
`$EXE_DIR$`	Directory for executable output
`$FILE_BNAME$`	Filename without extension
`$FILE_BPATH$`	Full path without extension
`$FILE_DIR$`	Directory of active file, no filename
`$FILE_FNAME$`	Filename of active file without path
`$FILE_PATH$`	Full path of active file (in editor, project, or message window)
`$LIST_DIR$`	Directory for list output
`$OBJ_DIR$`	Directory for object output
`$PROJ_DIR$`	Project directory
`$PROJ_FNAME$`	Project filename without path

Table 3: Argument variables

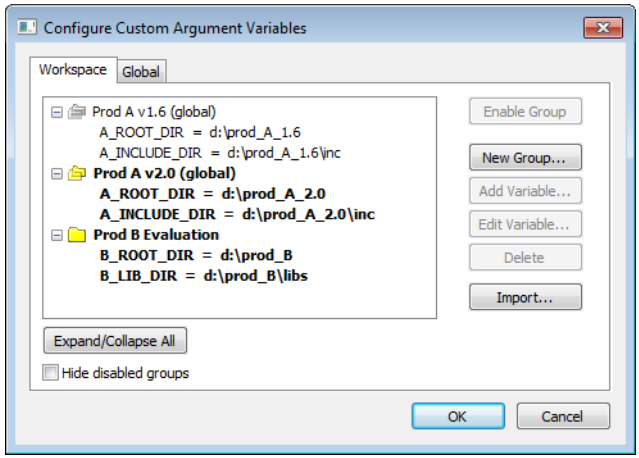
Variable	Description
$PROJ_PATH$	Full path of project file
$TARGET_DIR$	Directory of primary output file
$TARGET_BNAME$	Filename without path of primary output file and without extension
$TARGET_BPATH$	Full path of primary output file without extension
$TARGET_FNAME$	Filename without path of primary output file
$TARGET_PATH$	Full path of primary output file
$TOOLKIT_DIR$	Directory of the active product, for example c:\Program Files\IAR Systems\Embedded Workbench *N.n*\rh850
$USER_NAME$	Your host login name
WS_DIR	The active workspace directory (only available in the IDE, not when using iarbuild.exe or cspybat.exe)
$_*ENVVAR*_$	The Windows environment variable *ENVVAR*. Any name within $_ and _$ will be expanded to that system environment variable.
$*MY_CUSTOM_VAR*$	Your own argument variable, see *Configure Custom Argument Variables dialog box*, page 83. Any name within $ and $ will be expanded to the value you have defined.

Table 3: Argument variables (Continued)

Argument variables can also be used on some pages in the **IDE Options** dialog box, see *Tools menu*, page 194.

Configure Custom Argument Variables dialog box

The **Configure Custom Argument Variables** dialog box is available from the **Tools** menu.



Use this dialog box to define and edit your own custom argument variables. Typically, this can be useful if you install a third-party product and want to specify its include directory by using argument variables. Custom argument variables can also be used for simplifying references to files that you want to be part of your project.

Custom argument variables have one of two different scopes:

- *Workspace-local variables*, which are associated with a specific workspace and can only be seen by the workspace that was loaded when the variables were created.
- *Global variables*, which are available for use in all workspaces

You can organize your variables in named groups.

Workspace and Global tabs

Click the tab with the scope you want for your variable:

Workspace

- Both global and workspace-local variables are visible in the display area.
- Only workspace-local variables can be edited or removed.
- Groups of variables as well as individual variables can be added or imported to the local level.
- Workspace-local variables are stored in the file `Workspace.custom_argvars` in a specific directory, see *Files for local settings*, page 175.

Global

- Only variables that are defined as global are visible in the display area; all these variables can be edited or removed.
- Groups of variables as well as individual variables can be added or imported to the global level.
- Global variables are stored in the file `global.custom_argvars` in a specific directory, see *Files for global settings*, page 174.

Note that when you rely on custom argument variables in the build tool settings, some of the information needed for a project to build properly might now be in a `.custom_argvars` file. You should therefore consider version-controlling your custom argument file (workspace-local and global), and whether to document the need for using these variables.

Expand/Collapse All

Expands or collapses the view of the variables.

Hide disabled groups

Hides all groups of variables that you previously have disabled.

Enable Group / Disable Group

Enables or disables a group of variables that you have selected. The result differs depending on which tab you have open:

- **Workspace** tab: Enabling or disabling groups will only affect the current workspace.
- **Global** tab: Enabling will only affect newly created workspaces. These will inherit the current global state as the default for the workspace.

Note: You cannot use a variable that is part of a disabled group.

New Group

Opens the **New Group** dialog box where you can specify a name for a new group. When you click OK, the group is created and appears in the list of custom argument variables.

Add Variable

Opens the **Add Variables** dialog box where you can specify a name and value of a new variable to the group you have selected. When you click OK, the variable is created and appears in the list of custom argument variables.

Note that you can also add variables by importing previously defined variables. See **Import** below.

Edit Variable

Opens the **Edit Variables** dialog box where you can edit the name and value of a selected variable. When you click OK, the variable is created and appears in the list of custom argument variables.

Delete

Deletes the selected group or variable.

Import

Opens a file browser where you can locate a *Workspace*`.custom_argvars` file. The file can contain variables already defined and associated with another workspace or be a file created when installing a third-party product.

Project management

- Introduction to managing projects
- Managing projects
- Reference information on managing projects

Introduction to managing projects

These topics are covered:

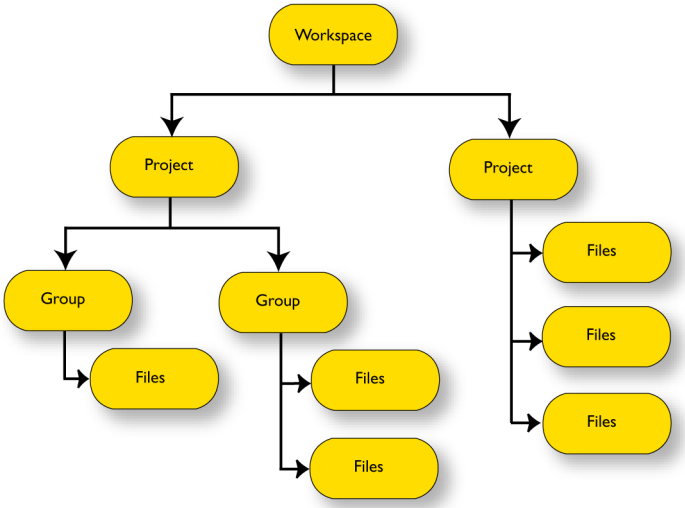
- Briefly about managing projects
- How projects are organized
- The IDE interacting with version control systems

BRIEFLY ABOUT MANAGING PROJECTS

In a large-scale development project, with hundreds of files, you must be able to organize the files in a structure that is easily navigated and maintained by several engineers.

The IDE comes with functions that will help you stay in control of all project modules, for example, C or C++ source code files, assembler files, include files, and other related

modules. You create *workspaces* and let them contain one or several *projects*. Files can be organized in *file groups*, and options can be set on all levels—project, group, or file.



Changes are tracked so that a request for rebuild will retranslate all required modules, making sure that no executable files contain out-of-date modules.

These are some additional features of the IDE:

- Project templates to create a project that can be built and executed for a smooth development startup
- Hierarchical project representation
- Source browser with an hierarchical symbol presentation
- Options can be set globally, on groups of source files, or on individual source files
- The Make command automatically detects changes and performs only the required operations
- Project connection to set up a connection between IAR Embedded Workbench and an external tool
- Text-based project files
- Custom Build utility to expand the standard toolchain in an easy way
- Command line build with the project file as input.

Navigating between project files

There are two main different ways to navigate your project files: using the **Workspace** window or the **Source Browser** window. The **Workspace** window displays an hierarchical view of the source files, dependency files, and output files and how they are logically grouped. The **Source Browser** window, on the other hand, displays information about the build configuration that is currently active in the **Workspace** window. For that configuration, the **Source Browser** window displays a hierarchical view of all globally defined symbols, such as variables, functions, and type definitions. For classes, information about any base classes is also displayed.

For more information about source browsing, see *Briefly about source browse information*, page 126.

HOW PROJECTS ARE ORGANIZED

The IDE allows you to organize projects in an hierarchical tree structure showing the logical structure at a glance.

The IDE has been designed to suit the way that software development projects are typically organized. For example, perhaps you need to develop related versions of an application for different versions of the target hardware, and you might also want to include debugging routines into the early versions, but not in the final application.

Versions of your applications for different target hardware will often have source files in common, and you might want to be able to maintain only one unique copy of these files, so that improvements are automatically carried through to each version of the application. Perhaps you also have source files that differ between different versions of the application, such as those dealing with hardware-dependent aspects of the application.

In the following sections, the various levels of the hierarchy are described.

Projects and workspaces

Typically you create one or several *projects*, where each project can contain either:

- Source code files, which you can use for producing your embedded application or a library. For an example where a library project has been combined with an application project, see the example about creating and using libraries in the tutorials.
- An externally built executable file that you want to load in C-SPY. For information about how to load executable files built outside of the IDE, see the *C-SPY® Debugging Guide for RH850*.

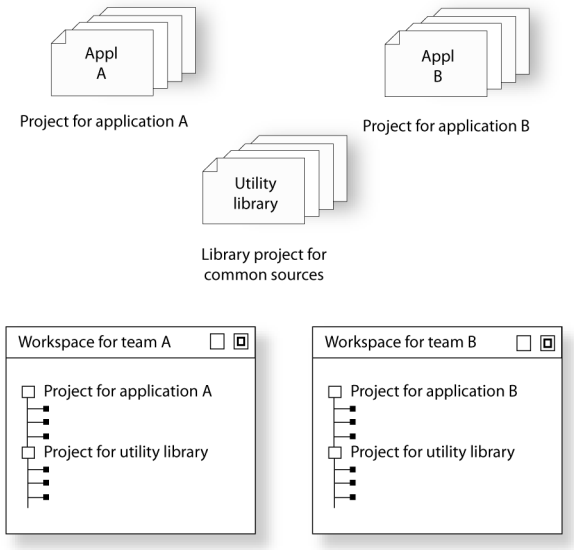
If you have several related projects, you can access and work with them simultaneously. To achieve this, you can organize related projects in *workspaces*.

Each workspace you define can contain one or more projects, and each project must be part of at least one workspace.

Consider this example: two related applications—for instance A and B—are developed, requiring one development team each (team A and B). Because the two applications are related, they can share parts of the source code between them. The following project model can be applied:

- *Three projects*—one for each application, and one for the common source code
- *Two workspaces*—one for team A and one for team B.

Collecting the common sources in a library project (compiled but not linked object code) is both convenient and efficient, to avoid having to compile it unnecessarily. This figure illustrates this example:



Projects and build configurations

Often, you need to build several versions of your project, for example, for different debug solutions that require different settings for the linker and debugger. Another example is when you need a separately built executable file with special debug output for execution trace, etc. IAR Embedded Workbench lets you define multiple build configurations for each project. In a simple case, you might need just two, called Debug and Release, where the only differences are the options used for optimization, debug information, and output format. In the Release configuration, the preprocessor symbol `NDEBUG` is defined, which means the application will not contain any asserts.

Additional build configurations might be useful, for instance, if you intend to use the application on different target devices. The application is the same, but hardware-related parts of the code differ. Thus, depending on which target device you intend to build for, you can exclude some source files from the build configuration. These build configurations might fulfill these requirements for Project A:

- Project A - Device 1:Release
- Project A - Device 1:Debug
- Project A - Device 2:Release
- Project A - Device 2:Debug

Groups

Normally, projects contain hundreds of files that are logically related. You can define each project to contain one or more groups, in which you can collect related source files. You can also define multiple levels of subgroups to achieve a logical hierarchy. By default, each group is present in all build configurations of the project, but you can also specify a group to be excluded from a particular build configuration.

Source files and their paths

Source files can be located directly under the project node or in a hierarchy of groups. The latter is convenient if the amount of files makes the project difficult to survey. By default, each file is present in all build configurations of the project, but you can also specify a file to be excluded from a particular build configuration.

Only the files that are part of a build configuration will actually be built and linked into the output code.

Once a project has been successfully built, all include files and output files are displayed in the structure below the source file that included or generated them.

Note: The settings for a build configuration can affect which include files that are used during the compilation of a source file. This means that the set of include files associated with the source file after compilation can differ between the build configurations.

The IDE supports relative source file paths to a certain degree, for:

- *Project files*

  Paths to files part of the project file are relative if they are located on the same drive. The path is relative either to `$PROJ_DIR$` or `$EW_DIR$`. The argument variable `$EW_DIR$` is only used if the path refers to a file located in a subdirectory of `$EW_DIR$` and the distance from `$EW_DIR$` is shorter than the distance from `$PROJ_DIR$`.

Paths to files that are part of the project file are absolute if the files are located on different drives.

- *Workspace files*

  For files located on the same drive as the workspace file, the path is relative to `$PROJ_DIR$`.

  For files located on another drive than the workspace file, the path is absolute.

- *Debug files*

  If your debug image file contains debug information, any paths in the file that refer to source files are absolute.

Drag and drop

You can easily drag individual source files and project files from Windows Explorer to the **Workspace** window. Source files dropped on a *group* are added to that group. Source files dropped outside the project tree—on the **Workspace** window background—are added to the active project.

THE IDE INTERACTING WITH VERSION CONTROL SYSTEMS

The IAR Embedded Workbench IDE can identify and access any files that are in a Subversion (SVN) working copy, see *Interacting with Subversion*, page 95.

From within the IDE you can connect an IAR Embedded Workbench project to an external SVN project, and perform some of the most commonly used operations.

To connect your IAR Embedded Workbench project to a version control system, you should be familiar with the version control *client application* you are using.

Note: Some of the windows and dialog boxes that appear when you work with version control in the IDE originate from the version control system and are not described in the documentation from IAR Systems. For information about details in the client application, refer to the documentation supplied with that application.

Note: Different version control systems use different terminology even for some of the most basic concepts involved. You must keep this in mind when you read the descriptions of the interaction between the IDE and the version control system.

Managing projects

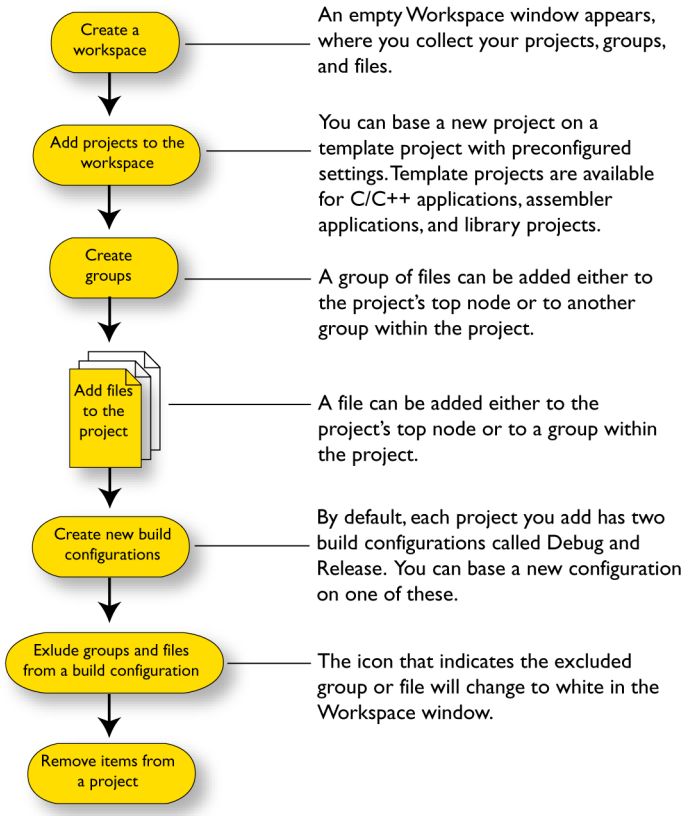
These tasks are covered:

- Creating and managing a workspace and its projects
- Viewing the workspace and its projects
- Interacting with Subversion

CREATING AND MANAGING A WORKSPACE AND ITS PROJECTS

This is a description of the overall procedure for creating the workspace, projects, groups, files, and build configurations. For a detailed step-by-step example, see *Creating an application project* in the tutorials.

The steps involved for creating and managing a workspace and its contents are:



Note: You do not have to use the same toolchain for the new build configuration as for other build configurations in the same project, and it might not be necessary for you to perform all of these steps and not in this order.

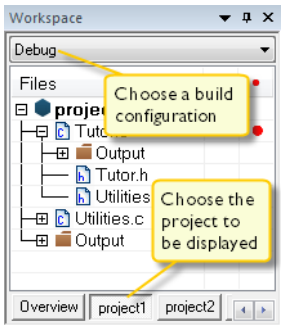
The **File** menu provides commands for creating workspaces. The **Project** menu provides commands for creating projects, adding files to a project, creating groups,

specifying project options, and running the IAR Systems development tools on the current projects.

VIEWING THE WORKSPACE AND ITS PROJECTS

The **Workspace** window is where you access your projects and files during the application development.

1 To choose which project you want to view, click its tab at the bottom of the **Workspace** window.



For each file that has been built, an `Output` folder icon appears, containing generated files, such as object files and list files. The latter is only generated if the list file option is enabled. The `Output` folder related to the project node contains generated files related to the whole project, such as the executable file and the linker map file (if the list file option is enabled).

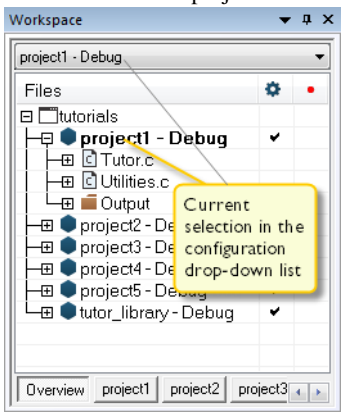
Also, any included header files will appear, showing dependencies at a glance.

2 To display the project with a different build configuration, choose that build configuration from the drop-down list at the top of the **Workspace** window**.**

The project and build configuration you have selected are displayed highlighted in the **Workspace** window. It is the project and build configuration that you select from the drop-down list that are built when you build your application.

3 To display an overview of all projects in the workspace, click the **Overview** tab at the bottom of the **Workspace** window.

An overview of all project members is displayed.



The current selection in the **Build Configuration** drop-down list is also highlighted when an overview of the workspace is displayed.

INTERACTING WITH SUBVERSION

The version control integration in IAR Embedded Workbench allows you to conveniently perform some of the most common Subversion operations directly from within the IDE, using the client applications `svn.exe` and `TortoiseProc.exe`.

To connect an IAR Embedded Workbench project to a Subversion system:

1. In the Subversion client application, set up a Subversion working copy.
2. In the IDE, connect your application project to the Subversion working copy.

To set up a Subversion working copy:

1. To use the Subversion integration in the IDE, make sure that `svn.exe` and `TortoiseProc.exe` are in your path.
2. Check out a working copy from a Subversion repository.

   The files that constitute your project do not have to come from the same working copy; all files in the project are treated individually. `However, note that TortoiseProc.exe` does not allow you to simultaneously, for example, check in files coming from different repositories.

To connect application projects to the Subversion working copy:

1. In the **Workspace** window, select the project for which you have created a Subversion working copy.

2 From the **Project** menu, choose **Version Control System>Connect Project to Subversion**. This command is also available from the context menu that appears when you right-click in the **Workspace** window.

For more information about the commands available for accessing the Subversion working copy, see *Version Control System menu for Subversion*, page 105.

Viewing the Subversion states

When your IAR Embedded Workbench project has been connected to the Subversion working copy, a column that contains status information for version control will appear in the **Workspace** window. Various icons are displayed, where each icon reflects the Subversion state, see *Subversion states*, page 107.

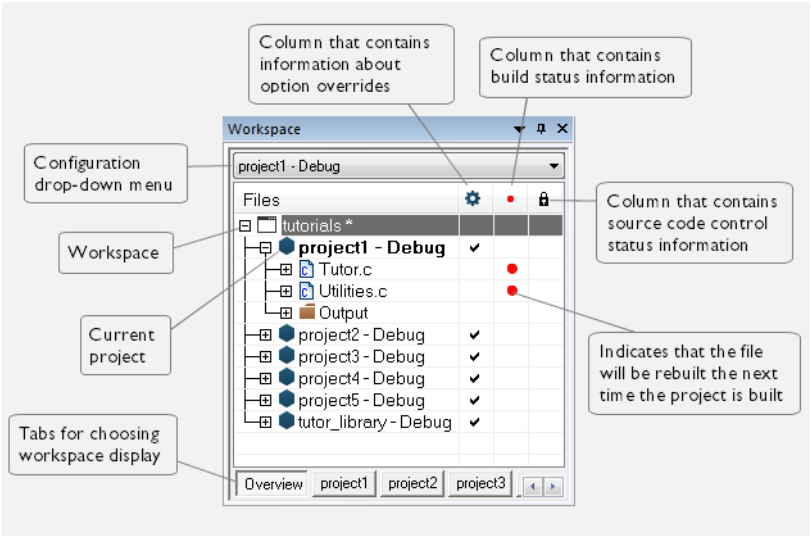
Reference information on managing projects

Reference information about:

- *Workspace window*, page 97
- *Create New Project dialog box*, page 102
- *Configurations for project dialog box*, page 103
- *New Configuration dialog box*, page 104
- *Add Project Connection dialog box*, page 105
- *Version Control System menu for Subversion*, page 105
- *Subversion states*, page 107

Workspace window

The **Workspace** window is available from the **View** menu.



Use this window to access your projects and files during the application development.

Drop-down list

At the top of the window there is a drop-down list where you can choose a build configuration to display in the window for a specific project.

The display area

This area contains four columns.

The **Files** column displays the name of the current workspace and a tree representation of the projects, groups and files included in the workspace. One or more of these icons are displayed:


Workspace


Project


Project with multi-file compilation


Group of files


Group excluded from the build


Group of files, part of multi-file compilation


Group of files, part of multi-file compilation, but excluded from the build


Object file or library


Assembler source file


C source file


C++ source file


Source file excluded from the build


Header file


Text file


HTML text file


Control file, for example the linker configuration file


IDE internal file


Other file

The column that contains status information about option overrides can have one of three icons for each level in the project:

Blank	There are no settings/overrides for this file/group.
Black check mark	There are local settings/overrides for this file/group.
Red check mark	There are local settings/overrides for this file/group, but they are either identical to the inherited settings or they are ignored because you use multi-file compilation, which means that the overrides are not needed.

The column that contains build status information can have one of three icons for each file in the project:

Blank	The file will not be rebuilt next time the project is built.
Red ball	The file will be rebuilt next time the project is built.
Small red dot	The file is being rebuilt.

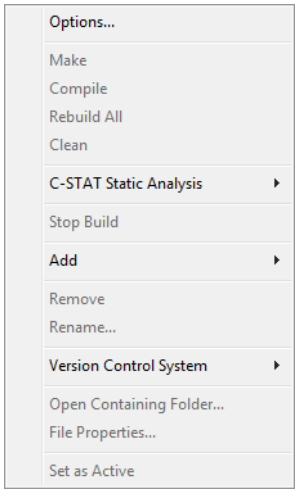
The column contains status information about version control. For information about the various icons, see *Subversion states*, page 107.

Use the tabs at the bottom of the window to choose which project to display. Alternatively, you can choose to display an overview of the entire workspace.

For more information about project management and using the **Workspace** window, see the *Introduction to managing projects*, page 87.

Context menu

This context menu is available:



These commands are available:

Options

Displays a dialog box where you can set options for each build tool for the selected item in the **Workspace** window, for example to exclude it from the build. You can set options for the entire project, for a group of files, or for an individual file. See *Setting project options using the Options dialog box*, page 111.

Make

Brings the current target up to date by compiling, assembling, and linking only the files that have changed since the last build.

Compile

Compiles or assembles the currently active file as appropriate. You can choose the file either by selecting it in the **Workspace** window, or by selecting the editor window containing the file you want to compile.

Rebuild All

Recompiles and relinks all files in the selected build configuration.

Clean

Deletes intermediate files.

C-STAT Static Analysis>Analyze Project

Makes C-STAT analyze the selected project. For more information about C-STAT, see the *C-STAT® Static Analysis Guide*.

C-STAT Static Analysis>Analyze File(s)

Makes C-STAT analyze the selected file(s). For more information about C-STAT, see the *C-STAT® Static Analysis Guide*.

C-STAT Static Analysis>Clear Analysis Results

Makes C-STAT clear the analysis information for previously performed analyses. For more information about C-STAT, see the *C-STAT® Static Analysis Guide*.

C-STAT Static Analysis>Generate HTML Summary

Shows a standard **Save As** dialog box where you can select the destination for a report summary in HTML and then create it. For more information about C-STAT, see the *C-STAT® Static Analysis Guide*.

C-STAT Static Analysis>Generate Full HTML Report

Shows a standard **Save As** dialog box where you can select the destination for a full report in HTML and create it. For more information about C-STAT, see the *C-STAT® Static Analysis Guide*.

Stop Build

Stops the current build operation.

Add>Add Files

Displays a dialog box where you can add files to the project.

Add>Add *filename*

Adds the indicated file to the project. This command is only available if there is an open file in the editor.

Add>Add Group

Displays the **Add Group** dialog box where you can add new groups to the project. For more information about groups, see *Groups*, page 91.

Remove

Removes selected items from the **Workspace** window.

Rename

Displays the **Rename Group** dialog box where you can rename a group. For more information about groups, see *Groups*, page 91.

Version Control System

Opens a submenu with commands for source code control, see *Version Control System menu for Subversion*, page 105.

Open Containing Folder

Opens the File Explorer that displays the directory where the selected file resides.

File Properties

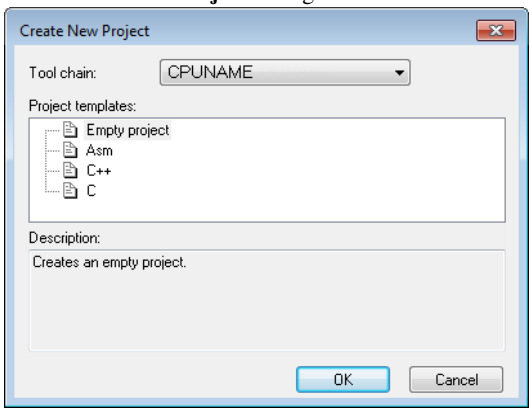
Displays a standard **File Properties** dialog box for the selected file.

Set as Active

Sets the selected project in the overview display to be the active project. It is the active project that will be built when the **Make** command is executed.

Create New Project dialog box

The **Create New Project** dialog box is available from the **Project** menu.



Use this dialog box to create a new project based on a template project. Template projects are available for C/C++ applications, assembler applications, and library projects. You can also create your own template projects.

Tool chain

Selects the target to build for. If you have several versions of IAR Embedded Workbench for different targets installed on your host computer, the drop-down list might contain some or all of these targets.

Project templates

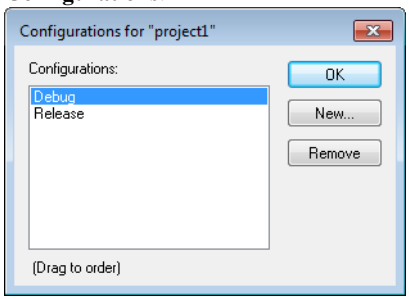
Select a template to base the new project on, from this list of available template projects.

Description

A description of the currently selected template.

Configurations for project dialog box

The **Configurations for project** dialog box is available by choosing **Project>Edit Configurations**.



Use this dialog box to define new build configurations for the selected project; either entirely new, or based on a previous project.

Configurations

Lists existing configurations, which can be used as templates for new configurations.

New

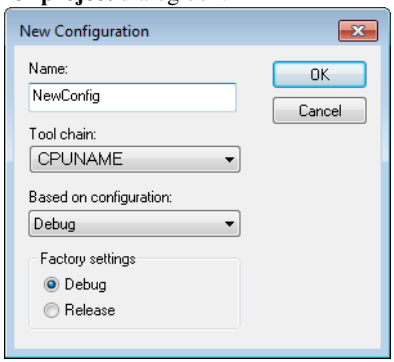
Displays a dialog box where you can define new build configurations, see *New Configuration dialog box*, page 104.

Remove

Removes the configuration that is selected in the **Configurations** list.

New Configuration dialog box

The **New Configuration** dialog box is available by clicking **New** in the **Configurations for project** dialog box.



Use this dialog box to define new build configurations; either entirely new, or based on any currently defined configuration.

Name

Type the name of the build configuration.

Tool chain

Specify the target to build for. If you have several versions of IAR Embedded Workbench for different targets installed on your host computer, the drop-down list might contain some or all of these targets.

Based on configuration

Selects a currently defined build configuration to base the new configuration on. The new configuration will inherit the project settings and information about the factory settings from the old configuration. If you select **None**, the new configuration will be based strictly on the factory settings.

Factory settings

Select the default factory settings that you want to apply to your new build configuration. These factory settings will be used by your project if you click the **Factory Settings** button in the **Options** dialog box.

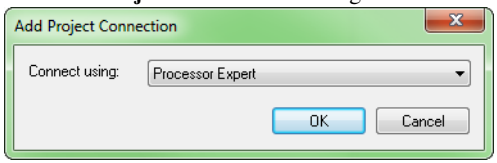
Choose between:

Debug, Factory settings suitable for a debug build configuration.

Release, Factory settings suitable for a release build configuration.

Add Project Connection dialog box

The **Add Project Connection** dialog box is available from the **Project** menu.



Use this dialog box to set up a project connection between IAR Embedded Workbench and an external tool. This can, for example, be useful if you want IAR Embedded Workbench to build source code files provided by the external tool. The source files will automatically be added to your project. If the set of files changes, the new set of files will automatically be used when the project is built in IAR Embedded Workbench.

To disable support for this, see *Project options*, page 61.

Connect using

Chooses the external tool that you want to set up a connection with.

OK

Displays a dialog box where you specify the connection.

Version Control System menu for Subversion

The **Version Control System** submenu is available from the **Project** menu and from the context menu in the **Workspace** window.

These commands are available:

For more information about interacting with an external version control system, see *The IDE interacting with version control systems*, page 92.

Menu commands

These commands are available for Subversion:

Commit

Displays Tortoise's **Commit** dialog box for the selected file(s).

Add

Displays Tortoise's **Add** dialog box for the selected file(s).

Revert

Displays Tortoise's **Revert** dialog box for the selected file(s).

Update

Opens Tortoise's **Update** window for the selected file(s).

Diff

Opens Tortoise's **Diff** window for the selected file(s).

Log

Opens Tortoise's **Log** window for the selected file(s).

Properties

Displays information available in the version control system for the selected file.

Refresh

Updates the version control system display status for all files that are part of the project. This command is always enabled for all projects under the version control system.

Connect Project to Subversion

Checks whether `svn.exe` and `TortoiseProc.exe` are in the path and then enables the connection between the IAR Embedded Workbench project and an existing checked-out working copy. After this connection has been created, a special column that contains status information appears in the **Workspace** window. Note that you must check out the source files from outside the IDE.

Disconnect Project from Subversion

Removes the connection between the selected IAR Embedded Workbench project and Subversion. The column in the **Workspace** window that contains SVN status information will no longer be visible for that project.

Subversion states

Each Subversion-controlled file can be in one of several states.

 (blue A)	Added.
 (red C)	Conflicted.
 (red D)	Deleted.
 (red I)	Ignored.
 (blank)	Not modified.
 (red M)	Modified.
 (red R)	Replaced.
 (gray X)	An unversioned directory created by an external definition.
 (gray question mark)	Item is not under version control.
 (black exclamation mark)	Item is missing—removed by a non-SVN command—or incomplete.
 (red tilde)	Item obstructed by an item of a different type.

Note: The version control system in the IAR Embedded Workbench IDE depends on the information provided by Subversion. If Subversion provides incorrect or incomplete information about the states, the IDE might display incorrect symbols.

Building projects

- Introduction to building projects
- Building a project
- Reference information on building

Introduction to building projects

These topics are covered:

- Briefly about building a project
- Extending the toolchain

BRIEFLY ABOUT BUILDING A PROJECT

The build process consists of these steps:

- Setting project options using the **Options** dialog box
- Building the project, either an application project or a library project
- Correcting any errors detected during the build procedure.

To make the build process more efficient, you can use the **Batch Build** command. This gives you the possibility to perform several builds in one operation. If necessary, you can also specify pre-build and post-build actions.

In addition to using the IAR Embedded Workbench IDE to build projects, you can also use the command line utility `iarbuild.exe`.

For examples of building application and library projects, see the tutorials in the Information Center. For more information about building library projects, see the *IAR C/C++ Development Guide for RH850*.

EXTENDING THE TOOLCHAIN

IAR Embedded Workbench provides a feature—Custom Build—which lets you extend the standard toolchain. This feature is used for executing external tools (not provided by IAR Systems). You can make these tools execute each time specific files in your project have changed.

If you specify custom build options on the **Custom tool configuration** page, the build commands treat the external tool and its associated files in the same way as the standard tools within the IAR Embedded Workbench IDE and their associated files. The relation

between the external tool and its input files and generated output files is similar to the relation between the C/C++ Compiler, `c` files, `h` files, and `o` files. For more information about custom build options, see *Custom build options*, page 235.

You specify filename extensions of the files used as input to the external tool. If the input file has changed since you last built your project, the external tool is executed; just as the compiler executes if a `c` file has changed. In the same way, any changes in additional input files (for instance, include files) are detected.

You must specify the name of the external tool. You can also specify any necessary command line options needed by the external tool, and the name of the output files generated by the external tool. Note that you can use argument variables for some of the file information.

You can specify custom build options to any level in the project tree. The options you specify are inherited by any sub-level in the project tree.

Tools that can be added to the toolchain

Some examples of external tools, or types of tools, that you can add to the IAR Embedded Workbench toolchain are:

- Tools that generate files from a specification, such as Lex and YACC
- Tools that convert binary files—for example files that contain bitmap images or audio data—to a table of data in an assembler or C source file. This data can then be compiled and linked together with the rest of your application.

For more information, see *Adding an external tool*, page 118.

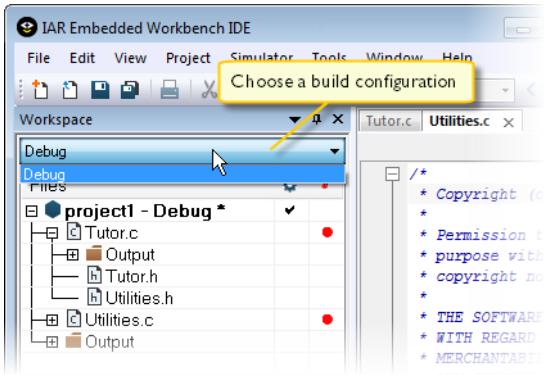
Building a project

These tasks are covered:

- Setting project options using the Options dialog box
- Building your project
- Correcting errors found during build
- Using pre- and post-build actions
- Building multiple configurations in a batch
- Building from the command line
- Adding an external tool

SETTING PROJECT OPTIONS USING THE OPTIONS DIALOG BOX

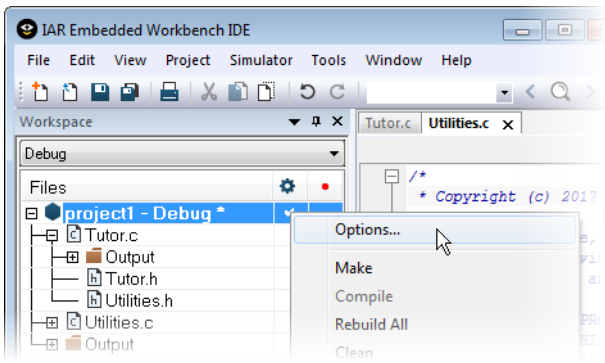
Before you can set project options, choose a build configuration.



By default, the IDE creates two build configurations when a project is created—**Debug** and **Release**. Every build configuration has its own project settings, which are independent of the other configurations.

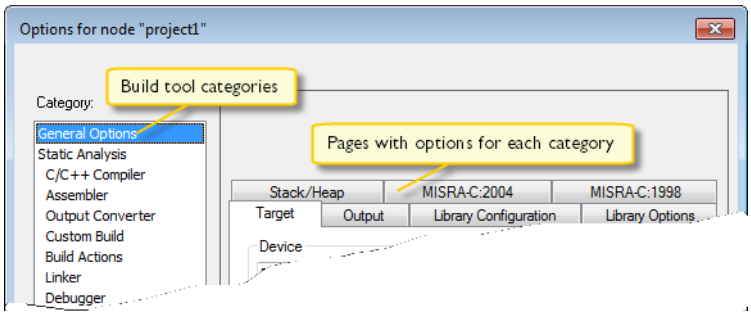
For example, a configuration that is used for debugging would not be highly optimized, and would produce output that suits the debugging. Conversely, a configuration for building the final application would be highly optimized, and produce output that suits a flash or PROM programmer.

2 Decide which *level* you want to set the options on: the entire project, groups of files, or for an individual file. Select that level in the **Workspace** window (in this example, the project level) and choose **Options** from the context menu to display the **Options** dialog box.



Note: There is one important restriction on setting options. If you set an option on group or file level (group or file level override), no options on higher levels that operate on files will affect that group or file.

3 The **Options** dialog box provides options for the build tools—a category for each build tool.

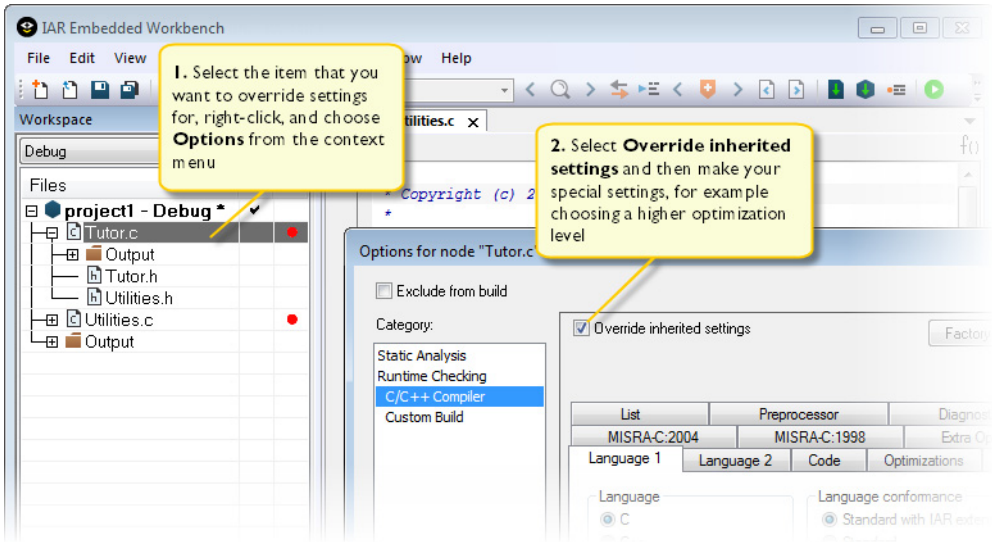


Options in the **General Options**, **Linker**, and **Debugger** categories can only be set on project level because they affect the entire build configuration, and cannot be set for individual groups and files. However, the options in the other categories can be set for the project, a group of files, or an individual file.

4 Select a category from the **Category** list to select which building tool to set options for. Which tools that are available in the **Category** list depends on which tools are included in your product. When you select a category, one or more pages containing options for that component are displayed.

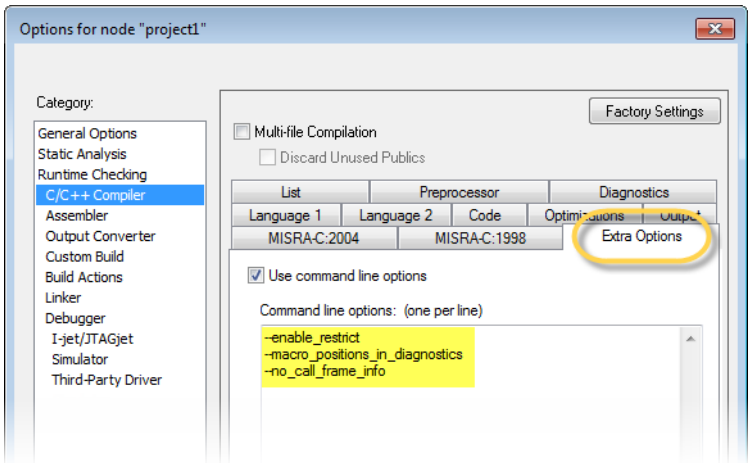
5 Click the tab that corresponds to the type of options you want to view or change. Make the appropriate settings. Some hints:

- To override project level settings, select the required item—for instance a specific group of files or an individual file—and select the option **Override inherited settings**.



The new settings will affect all members of that group, that is, files and any groups of files. Your local overrides are indicated with a checkmark in a separate column in the **Workspace** window.

- Use the **Extra Options** page to specify options that are only available as command line options and are not in the IDE.



- To restore all settings to the default factory settings, click the **Factory Settings** button, which is available for all categories except **General Options** and **Custom Build**. Note that two sets of factory settings are available: **Debug** and **Release**. Which one is used depends on your build configuration, see *New Configuration dialog box*, page 104.
- If you add a source file with a non-recognized filename extension to your project, you cannot set options on that source file. However, you can add support for additional filename extensions. For more information, see *Filename Extensions dialog box*, page 78.

BUILDING YOUR PROJECT

You can build your project either as an application project or a library project.

You have access to the build commands both from the **Project** menu and from the context menu that appears if you right-click an item in the **Workspace** window.

To build your project as an application project, choose one of the three build commands **Make**, **Compile**, and **Rebuild All**. They will run in the background, so you can continue editing or working with the IDE while your project is being built.

To build your project as a library project, choose **Project>Options>General Options>Output>Output file>Library** before you build your project. Then, **Linker** is replaced by **Library Builder** in the **Category** list in the **Options** dialog box, and the result of the build will be a library. For an example, see the tutorials.

For more information, see *Project menu*, page 189.

CORRECTING ERRORS FOUND DURING BUILD

Error messages are displayed in the **Build** message window.

To specify the level of output to the Build message window:

1. Right-click in the **Build** message window to open the context menu.
2. From the context menu, select the level of output you want: From **All**, which shows all messages, including compiler and linker information, to **Errors**, which only shows errors, but not warnings or other messages.

   If your source code contains errors, you can jump directly to the correct position in the appropriate source file by double-clicking the error message in the error listing in the **Build** window, or selecting the error and pressing Enter.

   After you have resolved any problems reported during the build process and rebuilt the project, you can directly start debugging the resulting code at the source level.

   For more information about the **Build** message window, see *Build window*, page 120.

USING PRE- AND POST-BUILD ACTIONS

If necessary, you can specify pre-build and post-build actions that you want to occur before or after the build. The Build Actions options in the **Options** dialog box—available from the **Project** menu—let you specify the actions required.

For more information about the Build Actions options, see *Build actions options*, page 237.

Using pre-build actions for time stamping

You can use pre-build actions to embed a time stamp for the build in the resulting binary file. Follow these steps:

1. Create a dedicated time stamp file, for example, `timestamp.c` and add it to your project.
2. In this source file, use the preprocessor macros `__TIME__` and `__DATE__` to initialize a string variable.
3. Choose **Project>Options>Build Actions** to open the **Build Actions** dialog box.
4. In the **Pre-build command line** text field, specify for example this pre-build action:

   ```
   cmd /c "del "$OBJ_DIR$\timestamp.o""
   ```

   This command removes the `timestamp.o` object file.

Alternatively, you can use the open source command line utility `touch` for this purpose or any other suitable utility that updates the modification time of the source file. For example:

```
"touch $PROJ_DIR$\timestamp.c"
```

5 If the project is not entirely up-to-date, the next time you use the **Make** command, the pre-build action will be invoked before the regular build process. Then the regular build process must always recompile `timestamp.c` and the correct timestamp will end up in the binary file.

If the project already is up-to-date, the pre-build action will not be invoked. This means that nothing is built, and the binary file still contains the timestamp for when it was last built.

BUILDING MULTIPLE CONFIGURATIONS IN A BATCH

Use the batch build feature when you want to build more than one configuration at once. A batch is an ordered list of build configurations. The **Batch Build** dialog box—available from the **Project** menu—lets you create, modify, and build batches of configurations.

For workspaces that contain several configurations, it is convenient to define one or more different batches. Instead of building the entire workspace, you can only build the appropriate build configurations, for instance Release or Debug configurations.

For more information about the **Batch Build** dialog box, see *Batch Build dialog box*, page 122.

BUILDING FROM THE COMMAND LINE

To build the project from the command line, use the IAR Command Line Build Utility (`iarbuild.exe`) located in the `common\bin` directory. Typically, this can be useful for automating your testing for continuous integration.

As input you use the project file, and the invocation syntax is:

```
iarbuild project.ewp [ -clean | -build | -make | -cstat_analyze |
-cstat_clean] config[,config1,config2,...]|*[-log
errors|warnings|info|all][-parallel number][-varfile filename]
```

These are the possible parameters:

Parameter	Description
project.ewp	Your IAR Embedded Workbench project file.
-clean	Removes any intermediate and output files.
-build	Rebuilds and relinks all files in the specified build configuration(s).
-make	Brings the specified build configuration(s) up to date by compiling, assembling, and linking only the files that have changed since the last build.
-cstat_analyze	Analyzes the project using C-STAT and generates information about the number of messages. For more information, see the *C-STAT® Static Analysis Guide*.
-cstat_clean	Cleans the C-STAT message database for the project. For more information, see the *C-STAT® Static Analysis Guide*.
config|*	*config*, the name of a configuration you want to build, which can be either one of the predefined configurations Debug or Release, or a name that you define yourself. For more information about build configurations, see *Projects and build configurations*, page 90. * (wild card character), the -clean, -build, and -make commands will process all configurations defined in the project.
-log errors	Displays build error messages.
-log warnings	Displays build warning and error messages.
-log info	Displays build warning and error messages, and messages issued by the #pragma message preprocessor directive.
-log all	Displays all messages generated from the build, for example compiler sign-on information and the full command line.
-parallel *number*	Specifies the number of parallel processes to run the compiler in to make better use of the cores in the CPU.
-varfile *filename*	Makes *custom-defined* argument variables become defined in a workspace scope available to the build engine by specifying the file to use. See *Configure Custom Argument Variables dialog box*, page 83.

Table 4: iarbuild.exe command line options

If you run the application from a command shell without specifying a project file, you will get a sign-on message describing available parameters and their syntax.

If the build process was successful, the IAR Command Line Build Utility returns 0. Otherwise it returns a non-zero number and a diagnostic message.

ADDING AN EXTERNAL TOOL

The following example demonstrates how to add the tool *Flex* to the toolchain. The same procedure can also be used for other tools.

In the example, Flex takes the file `myFile.lex` as input. The two files `myFile.c` and `myFile.h` are generated as output.

1. Add the file you want to work with to your project, for example `myFile.lex`.
2. Select this file in the **Workspace** window and choose **Project>Options**. Select **Custom Build** from the list of categories.
3. In the **Filename extensions** field, type the filename extension `.lex`. Remember to specify the leading period (`.`).
4. In the **Command line** field, type the command line for executing the external tool, for example:

   ```
   flex $FILE_PATH$ -o$FILE_BNAME$.c
   ```

   During the build process, this command line is expanded to:

   ```
   flex myFile.lex -omyFile.c
   ```

   Note the usage of *argument variables* and specifically the use of `$FILE_BNAME$` which gives the base name of the input file, in this example appended with the `c` extension to provide a C source file in the same directory as the input file `foo.lex`. For more information about these variables, see *Argument variables*, page 81.
5. In the **Output files** field, describe the output files that are relevant for the build. In this example, the tool Flex would generate two files—one source file and one header file. The text in the **Output files** text box for these two files would look like this:

   ```
   $FILE_BPATH$.c
   $FILE_BPATH$.h
   ```
6. If the external tool uses any additional files during the build, these should be added in the **Additional input files** field, for instance:

   ```
   $TOOLKIT_DIR$\inc\stdio.h
   ```

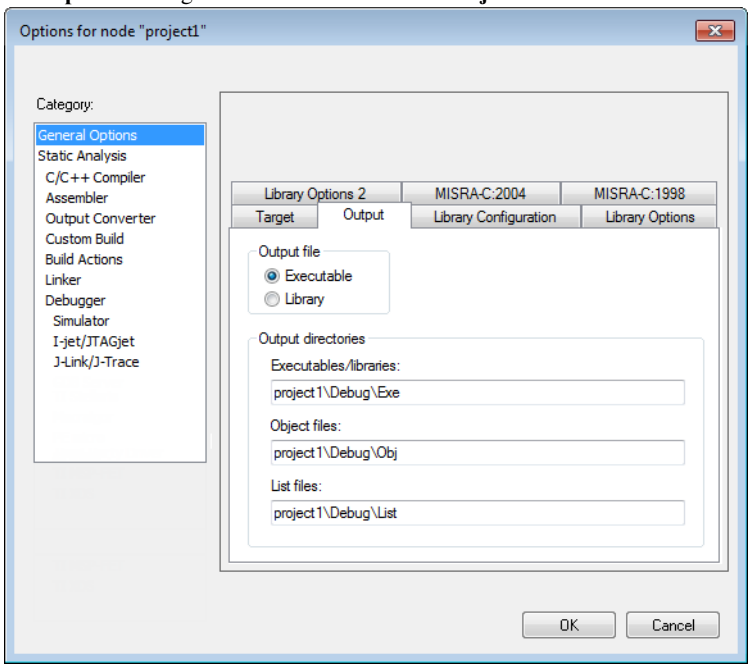
   This is important, because if the dependency files change, the conditions will no longer be the same and the need for a rebuild is detected.
7. Click **OK**.
8. To build your application, choose **Project>Make**.

Reference information on building

Reference information about:

Options dialog box

The **Options** dialog box is available from the **Project** menu.



Use this dialog box to specify your project settings.

See also *Setting project options using the Options dialog box*, page 111.

Category

Selects the build tool you want to set options for. The available categories will depend on the tools installed in your IAR Embedded Workbench IDE, and will typically include:

- General options
- Static Analysis, see the *C-STAT® Static Analysis Guide* for more information about these options
- C/C++ Compiler
- Assembler
- Output Converter, options for converting ELF output to Motorola, Intel-standard, or other simple formats, see *Output converter options*, page 233.
- Custom build, options for extending the toolchain
- Build Actions, options for pre-build and post-build actions
- Linker, available for application projects but not for library projects
- Library builder, available for library projects but not for application projects
- Debugger
- Simulator
- *C-SPY hardware drivers,* options specific to additional hardware debuggers.

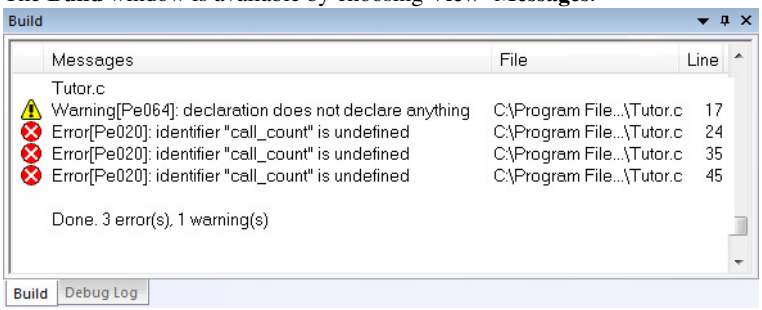
Selecting a category displays one or more pages of options for that component of the IDE.

Factory Settings

Restores all settings to the default factory settings. Note that this option is not available for all categories.

Build window

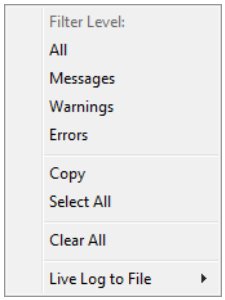
The **Build** window is available by choosing **View>Messages**.



This window displays the messages generated when building a build configuration. When opened, the window is, by default, grouped together with the other message windows. Double-click a message in the **Build** window to open the appropriate file for editing, with the insertion point at the correct position.

Context menu

This context menu is available:



These commands are available:

All

Shows all messages, including compiler and linker information.

Messages

Shows all messages.

Warnings

Shows warnings and errors.

Errors

Shows errors only.

Copy

Copies the contents of the window.

Select All

Selects the contents of the window.

Clear All

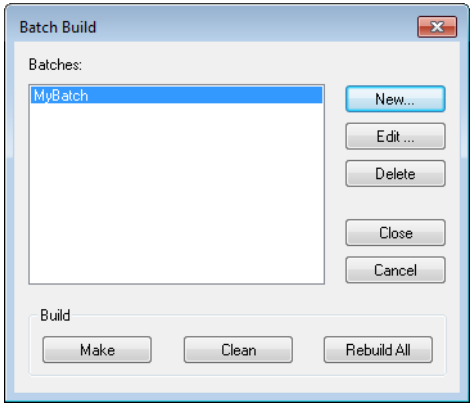
Deletes the contents of the window.

Live Log to File

Displays a submenu with commands for writing the build messages to a log file and setting filter levels for the log.

Batch Build dialog box

The **Batch Build** dialog box is available by choosing **Project>Batch build**.



This dialog box lists all defined batches of build configurations. For more information, see *Building multiple configurations in a batch*, page 116.

Batches

Select the batch you want to build from this list of currently defined batches of build configurations.

Build

Give the build command you want to execute:

- **Make**
- **Clean**
- **Rebuild All**.

New

Displays the **Edit Batch Build** dialog box, where you can define new batches of build configurations, see *Edit Batch Build dialog box*, page 123.

Remove

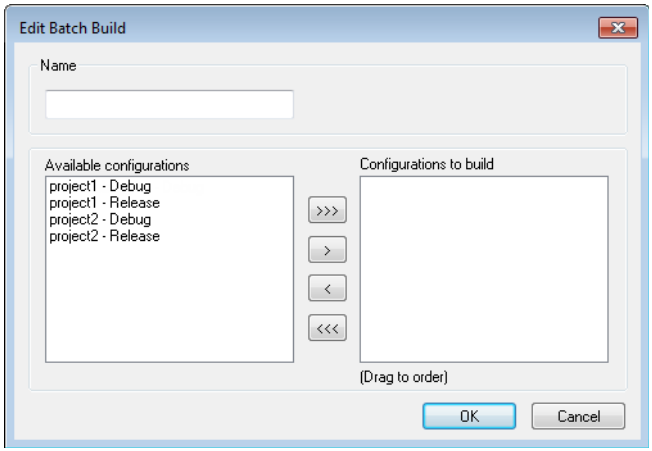
Removes the selected batch.

Edit

Displays the **Edit Batch Build** dialog box, where you can edit existing batches of build configurations.

Edit Batch Build dialog box

The **Edit Batch Build** dialog box is available from the **Batch Build** dialog box.



Use this dialog box to create new batches of build configurations, and edit already existing batches.

Name

Type a name for a batch that you are creating, or change the existing name (if you wish) for a batch that you are editing.

Available configurations

Select the configurations you want to move to be included in the batch you are creating or editing, from this list of all build configurations that belong to the workspace.

To move a build configuration from the **Available configurations** list to the **Configurations to build** list, use the arrow buttons.

Configurations to build

Lists the build configurations that will be included in the batch you are creating or editing. Drag the build configurations up and down to set the order between the configurations.

Editing

- Introduction to the IAR Embedded Workbench editor
- Editing a file
- Programming assistance
- Reference information on the editor

Introduction to the IAR Embedded Workbench editor

These topics are covered:

- Briefly about the editor
- Briefly about source browse information
- Customizing the editor environment

For information about how to use an external editor in the IAR Embedded Workbench IDE, see *Using an external editor*, page 35.

BRIEFLY ABOUT THE EDITOR

The integrated text editor allows you to edit multiple files in parallel, and provides both basic editing features and functions specific to software development, like:

- Automatic word and code completion
- Automatic line indentation and block indentation
- Parenthesis and bracket matching
- Function navigation within source files
- Context-sensitive help system that can display reference information for DLIB library functions and language extensions
- Text styles and color that identify the syntax of C or C++ programs and assembler directives
- Powerful search and replace commands, including multi-file search
- Direct jump to context from error listing
- Multibyte character support
- Parameter hints
- Bookmarks

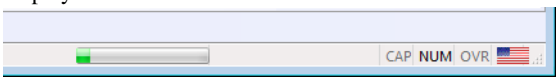
- Unlimited undo and redo for each window.

BRIEFLY ABOUT SOURCE BROWSE INFORMATION

Optionally, source browse information is continuously generated in the background. This information is used by many different features useful as programming assistance, for example:

- **Source Browser** window
- Go to definition or declaration
- Find all references
- Find all calls to or from a function, where the result is presented as a call graph.

The source browse information is updated when a file in the project is saved. When you save an edited source file, or when you open a new project, there will be a short delay before the information is up-to-date. During the update, progress information is displayed in the status bar.



Note: If you want the generation of source browse information to halt when you change focus from the IAR Embedded Workbench IDE to another program, make sure to enable the **No source browser and build status updates when the IDE is not the foreground process** option.

CUSTOMIZING THE EDITOR ENVIRONMENT

The IDE editor can be configured on the **IDE Options** pages **Editor** and **Editor>Colors and Fonts**. Choose **Tools>Options** to access the pages.

For information about these pages, see *Tools menu*, page 194.

Editing a file

The editor window is where you write, view, and modify your source code.

These tasks are covered:

- Indenting text automatically
- Matching brackets and parentheses
- Splitting the editor window into panes
- Dragging text
- Code folding

- Word completion
- Code completion
- Parameter hint
- Using and adding code templates
- Syntax coloring
- Adding bookmarks
- Using and customizing editor commands and shortcut keys
- Displaying status information

See also:

- *Programming assistance*, page 132
- *Using an external editor*, page 35

INDENTING TEXT AUTOMATICALLY

The text editor can perform various kinds of indentation. For assembler source files and plain text files, the editor automatically indents a line to match the previous line.

To indent several lines, select the lines and press the Tab key.

To move a whole block of lines back to the left again, press Shift+Tab.

For C/C++ source files, the editor indents lines according to the syntax of the C/C++ source code. This is performed whenever you:

- Press the Return key
- Type any of the special characters `{`, `}`, `:`, and `#`
- Have selected one or several lines, and choose the **Edit>Auto Indent** command.

To enable or disable the indentation:

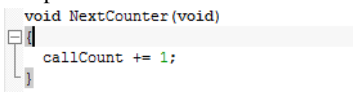
1. Choose **Tools>Options** and select **Editor**.
2. Select or deselect the **Auto indent** option.

To customize the C/C++ automatic indentation, click the **Configure** button.

For more information, see *Configure Auto Indent dialog box*, page 55.

MATCHING BRACKETS AND PARENTHESES

To highlight matching parentheses with a light gray color, place the insertion point next to a parenthesis:



The highlight remains in place as long as the insertion point is located next to the parenthesis.

To select all text between the brackets surrounding the insertion point, choose **Edit>Match Brackets**. Every time you choose **Match Brackets (grow)** or **Match Brackets (shrink)** after that, the selection will increase or shrink, respectively, to the next hierarchic pair of brackets.

Note: Both of these functions—automatic matching of corresponding parentheses and selection of text between brackets—apply to `()`, `[]`, `{}`, and <> (requires **Match All Brackets**).

SPLITTING THE EDITOR WINDOW INTO PANES

You can split the editor window horizontally or vertically into multiple panes, to look at different parts of the same source file at once, or to move text between two different panes.

To split a window into panes (horizontally or vertically), use the **Window>Split** command.

To revert to a single pane, double-click the splitter control or drag it to the edge of the window.

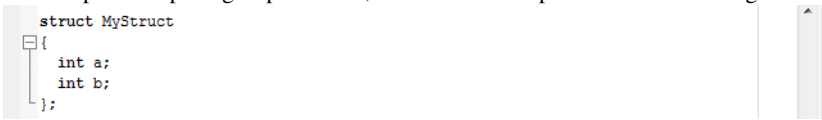
DRAGGING TEXT

To move text within an editor window or to copy between editor windows, select the text and drag it to the new location.

CODE FOLDING

Sections of code can be hidden and displayed using code folding.

To collapse or expand groups of lines, click on the fold points in the fold margin:



The fold point positions are based on the hierarchical structure of the document contents, for example, brace characters in C/C++ or the element hierarchy of an XML file. The **Toggle All Folds** command (Ctrl+Alt+F) can be used for expanding (or collapsing) all folds in the current document. The command is available from the **Edit** menu and from the context menu in the editor window. You can enable or disable the fold margin from **Tools>Options>Editor**.

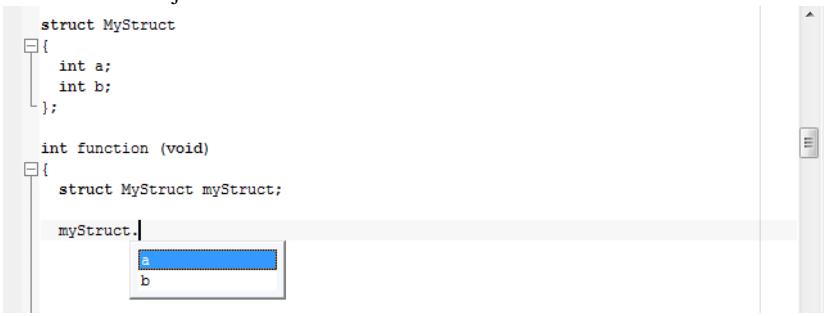
WORD COMPLETION

Word completion attempts to complete the word that you have started to type, basing the assumption on the contents of the rest of your document.

To make the editor complete the word that you have started to type, press Ctrl+Alt+Space or choose **Complete Word** from the context menu. If the suggestion is incorrect, repeat the command to get new suggestions.

CODE COMPLETION

To make the editor show a list of symbols that are available in a class, type `.`, `->`, or `::` after a class or object name:



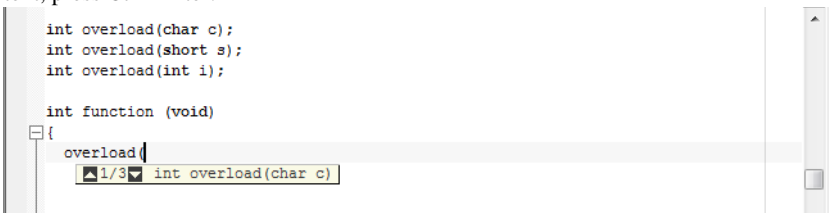
When you place the cursor anywhere else but after `.`, `->`, or `:`, the context menu lists all symbols available in the active translation unit.

Click on a symbol name in the list or choose it with the arrow keys and press Return to insert it at the current insertion point.

PARAMETER HINT

To make the editor suggest function parameters as tooltip information, start typing the first parenthesis after a function name.

When there are several overloaded versions of a function, you can choose which one to use by clicking the arrows in the tooltip (Ctrl+Up/Down). To insert the parameters as text, press Ctrl+Enter:



USING AND ADDING CODE TEMPLATES

Code templates are a method of conveniently inserting frequently used source code sequences, for example `for` loops and `if` statements. The code templates are defined in a plain text file. By default, a few example templates are provided. In addition, you can easily add your own code templates.

To set up the use of code templates:

1. Choose **Tools>Options>Editor>Setup Files**.
2. Select or deselect the **Use Code Templates** option. By default, code templates are enabled.
3. In the text field, specify which template file you want to use:
   - *The default template file*

     The original template file `CodeTemplates.txt` (alternatively`CodeTemplates.ENU.txt` or `CodeTemplates.JPN.txt` if you are using an IAR Embedded Workbench that is available in both English and Japanese) is located in a separate directory, see *Files for global settings*, page 174.

     Note that this is a local copy of the file, which means it is safe to modify it if you want.
   - *Your own template file*

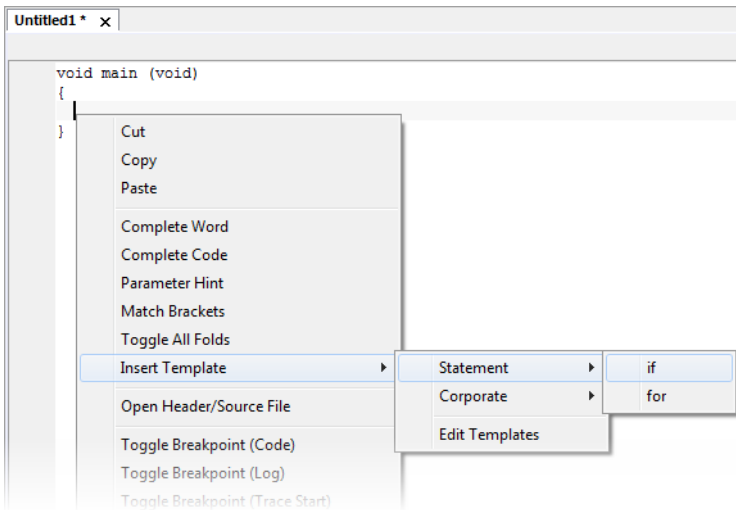
     Note that before you can choose your own template file, you must first have created one. To create your own template file, choose **Edit>Code Templates>Edit Templates**, add your code templates, and save the file with a new name. The syntax for defining templates is described in the default template file.

   A browse button is available for your convenience.
4. To use your new templates in your own template file, you must:
   - Delete the filename in the **Use Code Templates** text box.
   - Deselect the **Use Code Templates** option and click **OK**.
   - Restart the IAR Embedded Workbench IDE.
   - Choose **Tools>Options>Editor>Setup Files** again.

     The default code template file for the selected language version of the IDE should now be displayed in the **Use Code Templates** text box. Select the checkbox to enable the template.

To insert a code template into your source code:

1. In the editor window, right-click where you want the template to be inserted and choose **Insert Template** (Ctrl+Alt+V).
2. Choose a code template from the menu that appears.



If the code template requires any type of field input, as in the `for` loop example which needs an end value and a count variable, an input dialog box appears.

SYNTAX COLORING

If the **Tools>Options>Editor>Syntax highlighting** option is enabled, the IAR Embedded Workbench editor automatically recognizes the syntax of different parts of source code, for example:

- C and C++ keywords
- C and C++ comments
- Assembler directives and comments
- Preprocessor directives
- Strings.

The different parts of source code are displayed in different text styles.

To change these styles, choose **Tools>Options**, and use the **Editor>Colors and Fonts** options. For more information, see *Editor Colors and Fonts options*, page 59.

To define your own set of keywords that should be syntax-colored automatically:

1. In a text file, list all the keywords that you want to be automatically syntax-colored. Separate each keyword with either a space or a new line.
2. Choose **Tools>Options** and select **Editor>Setup Files**.

3 Select the **Use Custom Keyword File** option and specify your newly created text file. A browse button is available for your convenience.

4 Select **Editor>Colors and Fonts** and choose **User Keyword** from the **Syntax Coloring** list. Specify the font, color, and type style of your choice. For more information, see *Editor Colors and Fonts options*, page 59.

In the editor window, type any of the keywords you listed in your keyword file; see how the keyword is colored according to your specification.

ADDING BOOKMARKS

Use the **Edit>Navigate>Toggle Bookmark** command to add and remove bookmarks. To switch between the marked locations, choose **Edit>Navigate>Navigate Next Bookmark** or **Navigate Previous Bookmark**.

USING AND CUSTOMIZING EDITOR COMMANDS AND SHORTCUT KEYS

The **Edit** menu provides commands for editing and searching in editor windows, for instance, unlimited undo/redo. You can also find some of these commands on the context menu that appears when you right-click in the editor window. For more information about each command, see *Edit menu*, page 182.

There are also editor shortcut keys for:

- moving the insertion point
- scrolling text
- selecting text.

For more information about these shortcut keys, see *Editor shortcut key summary*, page 164.

To change the default shortcut key bindings, choose **Tools>Options**, and click the **Key Bindings** tab. For more information, see *Key Bindings options*, page 49.

DISPLAYING STATUS INFORMATION

The status bar is available by choosing **View>Status Bar**. For more information, see *IAR Embedded Workbench IDE window*, page 39.

Programming assistance

There are several features in the editor that assist you during your software development. This section describes various tasks related to using the editor.

These tasks are covered:

- Navigating in the insertion point history
- Navigating to a function
- Finding a definition or declaration of a symbol
- Finding references to a symbol
- Finding function calls for a selected function
- Switching between source and header files
- Displaying source browse information
- Text searching
- Accessing online help for reference information

NAVIGATING IN THE INSERTION POINT HISTORY

The current position of the insertion point is added to the insertion point history by actions like **Go to definition** and clicking on the result for the **Find in Files** command. You can jump in the history either forward or backward by using the **Navigate Forward**  and **Navigate Backward**  buttons (or by pressing Alt + Right Arrow or Alt + Left Arrow).

NAVIGATING TO A FUNCTION

Click the **Go to function** button in the top-right corner of the editor window to list all functions defined in the source file displayed in the window. You can then choose to navigate directly to one of the functions by clicking it in the list. Note that the list is refreshed when you save the file.

FINDING A DEFINITION OR DECLARATION OF A SYMBOL

To see the definition or declaration of a global symbol or a function, you can use these alternative methods:

- In the editor window, right-click on a symbol and choose the **Go to definition** or **Go to declaration** command from the context menu that appears. If more than one declaration is found, the declarations are listed in the **Declarations** window from where you can navigate to a specific declaration.
- In the **Source Browser** window, double-click on a symbol to view the definition
- In the **Source Browser** window, right-click on a symbol, or function, and choose the **Go to definition** command from the context menu that appears

The definition of the symbol or function is displayed in the editor window.

FINDING REFERENCES TO A SYMBOL

To find all references for a specific symbol, select the symbol in the editor window, right-click and choose **Find All References** from the context menu. All found references are displayed in the **References** window.

You can now navigate between the references.

FINDING FUNCTION CALLS FOR A SELECTED FUNCTION

To find all calls to or from a function, select the function in the editor window or in the **Source Browser** window, right-click and choose either **Find All Calls to** or **Find All Calls from** from the context menu. The result is displayed in the **Call Graph** window.

You can navigate between the function calls.

SWITCHING BETWEEN SOURCE AND HEADER FILES

If the insertion point is located on an `#include` line, you can choose the **Open "header.h"** command from the context menu, which opens the header file in an editor window. You can also choose the command **Open Header/Source File**, which opens the header or source file with a corresponding filename to the current file, or activates it if it is already open. This command is available if the insertion point is located on any line except an `#include` line.

DISPLAYING SOURCE BROWSE INFORMATION

1. To open the **Source Browser** window, choose **View>Source Browser>Source Browser**. Source browse information is displayed for the active build configuration.

   Note that you can choose a file filter and a type filter from the context menu that appears when you right-click in the window.

2. To display browse information in the **Source Browser** window, choose **Tools>Options>Project** and select the option **Generate browse information**.

TEXT SEARCHING

There are several standard search functions available in the editor:

- **Quick search** text box
- **Find** dialog box
- **Replace** dialog box
- **Find in Files** dialog box
- **Replace in Files** dialog box
- **Incremental Search** dialog box.

To use the Quick search text box on the toolbar:

1. Type the text you want to search for and press Enter.
2. Press Esc to stop the search. This is a quick method of searching for text in the active editor window.

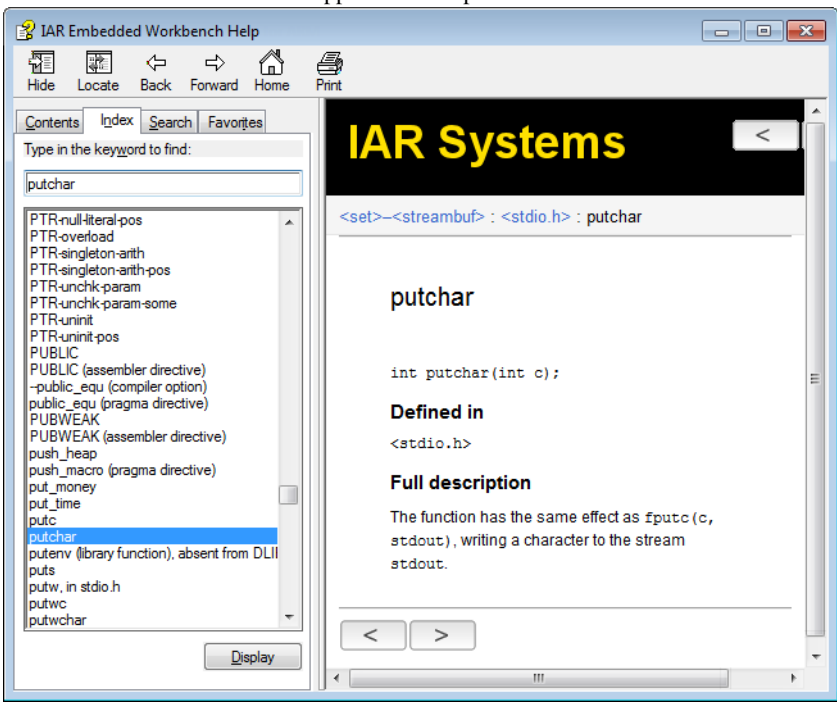
To use the Find, Replace, Find in Files, Replace in Files, and Incremental Search functions:

1. Before you use the search commands, choose **Tools>Options>Editor** and make sure the **Show bookmarks** option is selected.
2. Choose the appropriate search command from the **Edit** menu. For more information about each search function, see *Edit menu*, page 182.
3. To remove the blue flag icons that have appeared in the left-hand margin, right-click in the **Find in Files** window and choose **Clear All** from the context menu.

ACCESSING ONLINE HELP FOR REFERENCE INFORMATION

When you need to know the syntax of a library function, extended keyword, intrinsic function, etc, select it in the editor window and press F1.

The documentation for the item appears in a help window.



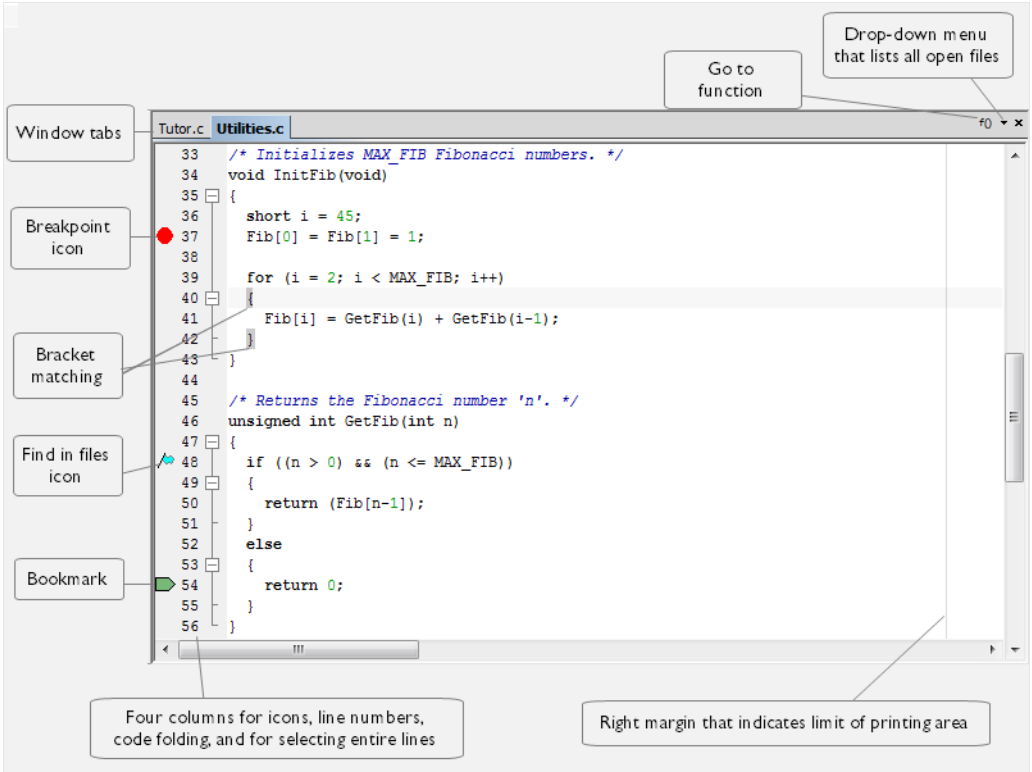
Reference information on the editor

Reference information about:

- *Editor window*, page 137
- *Find dialog box*, page 146
- *Find in Files window*, page 147
- *Replace dialog box*, page 148
- *Find in Files dialog box*, page 149
- *Replace in Files dialog box*, page 151
- *Incremental Search dialog box*, page 153
- *Declarations window*, page 154
- *Ambiguous Definitions window*, page 155
- *References window*, page 156
- *Source Browser window*, page 157

Editor window

The editor window is opened when you open or create a text file in the IDE.



You can open one or several text files, either from the **File** menu, or by double-clicking them in the **Workspace** window. All open files are available from the drop-down menu at the upper right corner of the editor window. Several editor windows can be open at the same time.

Source code files and HTML files are displayed in editor windows. From an open HTML document, hyperlinks to HTML files work like in an ordinary web browser. A link to an `eww` workspace file opens the workspace in the IDE, and closes any currently open workspace and the open HTML document.

When you want to print a source file, it can be useful to enable the option **Show line numbers**—available by choosing **Tools>Options>Editor**.

The editor window is always docked, and its size and position depend on other currently open windows.

For more information about using the editor, see *Editing a file*, page 126 and *Programming assistance*, page 132.

Relative source file paths

The IDE has partial support for relative source file paths.

If a source file is located in the project file directory or in any subdirectory of the project file directory, the IDE uses a path relative to the project file when accessing the source file.

Documentation comments

In addition to regular comments that start with `//` (in C++) or `/*` (in C and C++), the editor supports *documentation comments*, that start with `/**`, `/*!`, `///` or `//!`. The editor can distinguish these documentation comments from regular comments. By default, the editor assigns the two types of comments different colors.

Inside a documentation comment, the editor highlights doxygen-style keywords (keywords that begin with `\` or `@`) and by default uses a different color for them than for the rest of the comment. The color depends on whether the keyword is identified as an existing doxygen keyword or not. You can customize the editor's use of colors on the **Tools>Options>Editor>Colors and Fonts** page, see *Editor Colors and Fonts options*, page 59.

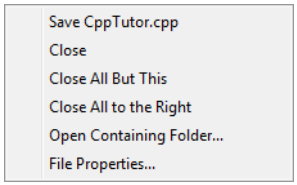
Lines inside documentation comment blocks can be shown in tooltips and parameter hints for variables and functions. A comment block with no doxygen-style keywords will be shown as a concatenated text string in tooltips and parameter hints. After the occurrence of a doxygen-style keyword, only text written after a `@brief` keyword will be shown in tooltips and parameter hints.

Window tabs, tab groups, and tab context menu

The name of the open file is displayed on the tab. If you open several files, they are organized in a *tab group*. Click the tab for the file that you want to display. If a file has been modified after it was last saved, an asterisk appears on the tab after the filename, for example `Utilities.c *`. If a file is read-only, a padlock icon is visible on the tab.

The tab's tooltip shows the full path and a remark if the file is not a member of the active project.

A context menu appears if you right-click on a tab in the editor window.



These commands are available:

Save *file*

Saves the file.

Close

Closes the file.

Close All But This

Closes all tabs except the current tab.

Close All to the Right

Closes all tabs to the right of the current tab.

Open Containing Folder

Opens the File Explorer that displays the directory where the selected file resides.

File Properties

Displays a standard **File Properties** dialog box.

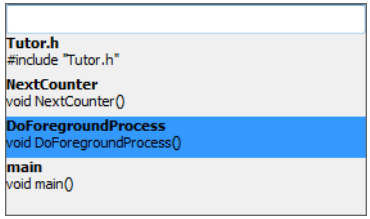
Multiple editor windows and splitter controls

You can have one or several editor windows open at the same time. The commands on the **Window** menu allow you to split the editor window into panes and to open multiple editor windows. There are also commands for moving files between editor windows.

For more information about each command on the **Window** menu, see *Window menu*, page 195.

Go to function

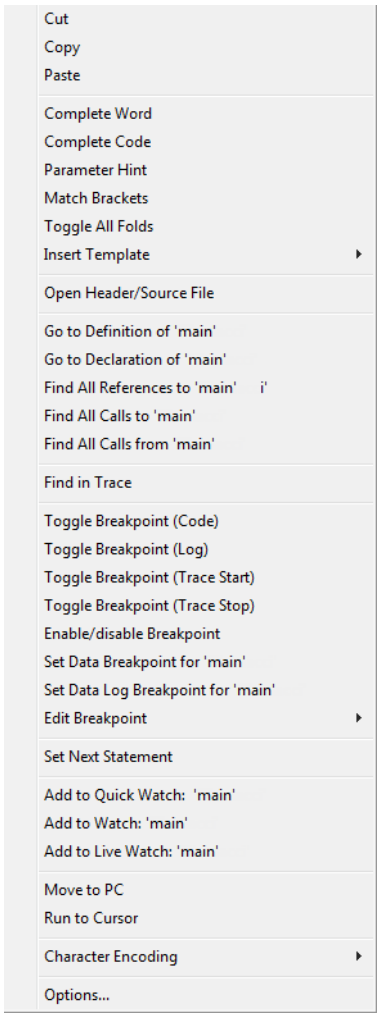
Click the **Go to function** button in the top right-hand corner of the editor window to list all functions of the C or C++ editor window.



Filter the list by typing the name of the function you are looking for. Then click the name of the function that you want to show in the editor window.

Context menu

This context menu is available:



The contents of this menu depend on whether the debugger is started or not, and on the C-SPY driver you are using. Typically, additional breakpoint types might be available on this menu. For information about available breakpoints, see the *C-SPY® Debugging Guide for RH850*.

These commands are available:

Cut, Copy, Paste

Standard window commands.

Complete Word

Attempts to complete the word you have begun to type, basing the guess on the contents of the rest of the editor document.

Complete Code

Shows a list of symbols that are available in a class, when you place the insertion point after `.`, `->`, or `::` and when these characters are preceded by a class or object name. For more information, see *Code completion*, page 129.

Parameter Hint

Suggests parameters as tooltip information for the function parameter list you have begun to type. When there are several overloaded versions of a function, you can choose which one to use by clicking the arrows in the tooltip. For more information, see *Parameter hint*, page 129.

Match Brackets

Selects all text between the brackets immediately surrounding the insertion point, increases the selection to the next hierarchic pair of brackets, or beeps if there is no higher bracket hierarchy.

Toggle All Folds

Expands/collapses all code folds in the active project.

Insert Template

Displays a list in the editor window from which you can choose a code template to be inserted at the location of the insertion point. If the code template you choose requires any field input, the **Template** dialog box appears. For more information about this dialog box, see *Template dialog box*, page 163. For information about using code templates, see *Using and adding code templates*, page 130.

Open "*header.h*"

Opens the header file *`header`*`.h` in an editor window. If more than one header file with the same name is found and the IDE does not have access to dependency information, the **Resolve File Ambiguity** dialog box is displayed, see *Resolve File Ambiguity dialog box*, page 162. This menu command is only available if the insertion point is located on an `#include` line when you open the context menu.

Open Header/Source File

Opens the header or source code file that has same base name as the current file. If the destination file is not open when you choose the command, the file will first be opened. This menu command is only available if the insertion point is located on any line except an `#include` line when you open the context menu. This command is also available from the **File>Open** menu.

Go to Definition of *symbol*

Places the insertion point at the definition of the symbol. If no definition is found in the source code, the first declaration will be used instead. If more than one possible definition is found, they are listed in the **Ambiguous Definitions** window. See *Ambiguous Definitions window*, page 155.

Go to Declaration of *symbol*

If only one declaration is found, the command puts the insertion point at the declaration of the symbol. If more than one declaration is found, these declarations are listed in the **Declarations** window.

Find All References to *symbol*

The references are listed in the **References** window.

Find All Calls to *symbol*

Opens the **Call Graph** window which displays all functions in the project that calls the selected function, see *Call Graph window*, page 162. If this command is disabled, make sure to select a function in the editor window.

Find All Calls from *symbol*

Opens the **Call Graph** window which displays all functions in the project that are called from the selected function, see *Call Graph window*, page 162. If this command is disabled, make sure to select a function in the editor window.

Find in Trace

Searches the contents of the **Trace** window for occurrences of the given location—the position of the insertion point in the source code—and reports the result in the **Find in Trace** window. This menu command requires support for Trace in the C-SPY driver you are using, see the *C-SPY® Debugging Guide for RH850*.

Toggle Breakpoint (Code)

Toggles a code breakpoint at the statement or instruction containing or close to the cursor in the source window. For information about code breakpoints, see the *C-SPY® Debugging Guide for RH850*.

Toggle Breakpoint (Log)

Toggles a log breakpoint at the statement or instruction containing or close to the cursor in the source window. For information about log breakpoints, see the *C-SPY® Debugging Guide for RH850*.

Toggle Breakpoint (Trace Start)

Toggles a Trace Start breakpoint. When the breakpoint is triggered, trace data collection starts. For information about Trace Start breakpoints, see the *C-SPY® Debugging Guide for RH850*. Note that this menu command is only available if the C-SPY driver you are using supports trace.

Toggle Breakpoint (Trace Stop)

Toggles a Trace Stop breakpoint. When the breakpoint is triggered, trace data collection stops. For information about Trace Stop breakpoints, see the *C-SPY® Debugging Guide for RH850*. Note that this menu command is only available if the C-SPY driver you are using supports trace.

Enable/disable Breakpoint

Toggles a breakpoint between being disabled, but not actually removed—making it available for future use—and being enabled again.

Set Data Breakpoint for '*variable*'

Toggles a data log breakpoint on variables with static storage duration. Requires support in the C-SPY driver you are using. For more information about data breakpoints, see the *C-SPY® Debugging Guide for RH850*.

Set Data Log Breakpoint for '*variable*'

Toggles a data log breakpoint on variables with static storage duration. Requires support in the C-SPY driver you are using. The breakpoints you set in this window will be triggered by both read and write accesses; to change this, use the **Breakpoints** window. For more information about data logging and data log breakpoints, see the *C-SPY® Debugging Guide for RH850*.

Edit Breakpoint

Displays the **Edit Breakpoint** dialog box to let you edit the breakpoint available on the source code line where the insertion point is located. If there is more than one breakpoint on the line, a submenu is displayed that lists all available breakpoints on that line.

Set Next Statement

Sets the Program Counter directly to the selected statement or instruction without executing any code. This menu command is only available when you are using the debugger. For more information, see the *C-SPY® Debugging Guide for RH850*.

Add to Quick Watch: *symbol*

Opens the **Quick Watch** window and adds the symbol, see the *C-SPY® Debugging Guide for RH850*. This menu command is only available when you are using the debugger.

Add to Watch: *symbol*

Opens the symbol to the **Watch** window and adds the symbol. This menu command is only available when you are using the debugger.

Add to Live Watch: *symbol*

Opens the **Live Watch** window and adds the symbol, see the *C-SPY® Debugging Guide for RH850*. This menu command is only available when you are using the debugger.

Move to PC

Moves the insertion point to the current PC position in the editor window. This menu command is only available when you are using the debugger.

Run to Cursor

Executes from the current statement or instruction up to the statement or instruction where the insertion point is located. This menu command is only available when you are using the debugger.

Character Encoding

Interprets the source file according to the specified character encoding. Choose between:

System (uses the Windows settings)
Western European
UTF-8
Japanese (Shift-JIS)
Chinese Simplified (GB2312)
Chinese Traditional (Big5)
Korean (Unified Hangul Code)
Arabic
Baltic
Central European
Greek
Hebrew
Russian
Thai
Vietnamese
Convert to UTF-8 (converts the document to UTF-8)

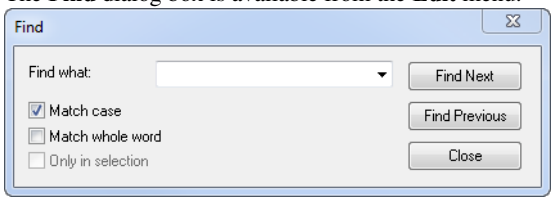
Use one of these settings if the **Auto-detect character encoding** option could not determine the correct encoding or if the option is deselected. For more information about file encoding, see *Editor options*, page 52.

Options

Displays the **IDE Options** dialog box, see *Tools menu*, page 194.

Find dialog box

The **Find** dialog box is available from the **Edit** menu.



Note that the contents of the dialog box might be different if you search in an editor window compared to if you search in the **Memory** window. This screen shot reflects the dialog box when you search in an editor window.

Find what

Specify the text to search for. Use the drop-down list to use old search strings.

When you search in the **Memory** window, the value you search for must be a multiple of the display unit size. For example, when using the **2 units** size in the **Memory** window, the search value must be a multiple of two bytes.

Match case

Searches only for occurrences that exactly match the case of the specified text. Otherwise, specifying `int` will also find `INT` and `Int`. This option is only available when you perform the search in an editor window.

Match whole word

Searches for the specified text only if it occurs as a separate word. Otherwise, specifying `int` will also find `print`, `sprintf` etc. This option is only available when you perform the search in an editor window.

Search as hex

Searches for the specified hexadecimal value. This option is only available when you perform the search in the **Memory** window.

Only in selection

Limits the search operation to the selected lines (when searching in an editor window) or to the selected memory area (when searching in the **Memory** window). The option is only enabled when a selection has been made before you open the dialog box.

Find Next

Searches for the next occurrence of the specified text.

Find Previous

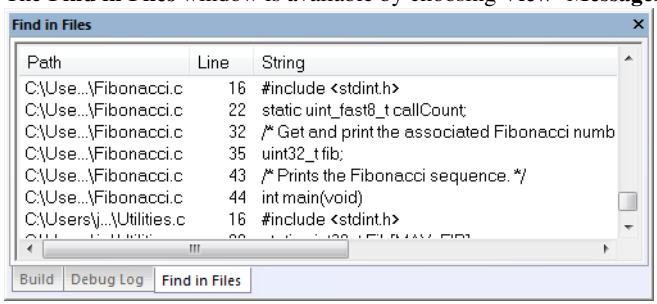
Searches for the previous occurrence of the specified text.

Stop

Stops an ongoing search. This button is only available during a search in the **Memory** window.

Find in Files window

The **Find in Files** window is available by choosing **View>Messages**.

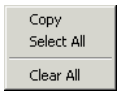


This window displays the output from the **Edit>Find and Replace>Find in Files** command. When opened, this window is, by default, grouped together with the other message windows.

Double-click an entry in the window to open the corresponding file with the insertion point positioned at the correct location. That source location is highlighted with a blue flag icon. Choose **Edit>Next Error/Tag** or press F4 to jump to the next in sequence.

Context menu

This context menu is available:



These commands are available:

Copy

Copies the selected content of the window.

Select All

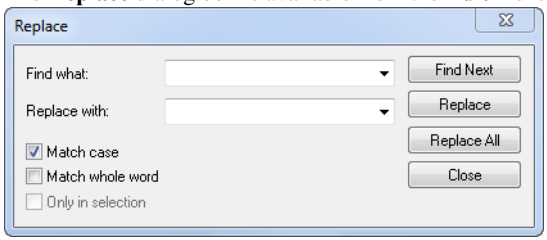
Selects the contents of the window.

Clear All

Deletes the contents of the window and any blue flag icons in the left-side margin of the editor window.

Replace dialog box

The **Replace** dialog box is available from the **Edit** menu.



Note that the contents of the dialog box are different if you search in an editor window compared to if you search in the **Memory** window.

Find what

Specify the text to search for. Use the drop-down list to use old search strings.

Replace with

Specify the text to replace each found occurrence with. Use the drop-down list to use old search strings.

Match case

Searches only for occurrences that exactly match the case of the specified text. Otherwise, specifying `int` will also find `INT` and `Int`. This option is only available when you perform the search in an editor window.

Match whole word

Searches for the specified text only if it occurs as a separate word. Otherwise, `int` will also find `print`, `sprintf` etc. This option is only available when you search in an editor window.

Search as hex

Searches for the specified hexadecimal value. This option is only available when you perform the search in the **Memory** window.

Only in selection

Limits the search operation to the selected lines (when searching in an editor window) or to the selected memory area (when searching in the **Memory** window). The option is only enabled when a selection has been made before you open the dialog box.

Find next

Searches for the next occurrence of the specified text.

Replace

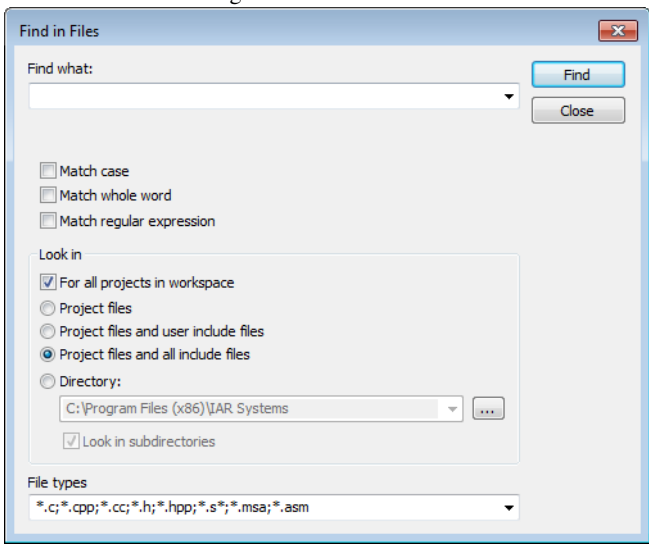
Replaces the searched text with the specified text.

Replace all

Replaces all occurrences of the searched text in the current editor window.

Find in Files dialog box

The **Find in Files** dialog box is available from the **Edit** menu.



Use this dialog box to search for a string in files.

The result of the search appears in the **Find in Files** message window—available from the **View** menu. You can then go to each occurrence by choosing the **Edit>Next Error/Tag** command, alternatively by double-clicking the messages in the **Find in Files** message window. This opens the corresponding file in an editor window with the insertion point positioned at the start of the specified text. A blue flag in the left-hand margin indicates the line with the string you searched for.

Find what

Specify the string you want to search for, or a regular expression. Use the drop-down list to use old search strings/expressions. You can narrow the search down with one or more of these conditions:

Match case

Searches only for occurrences that exactly match the case of the specified text. Otherwise, specifying `int` will also find `INT` and `Int`.

Match whole word

Searches only for the string when it occurs as a separate word (mnemonic `&w`). Otherwise, `int` will also find `print`, `sprintf` and so on.

Match regular expression

Interprets the search string as a the regular expression, which must follow the standard for the Perl programming language.

Look in

Specify which files you want to search in. Choose between:

For all projects in workspace

Searches all projects in the workspace, not just the active project.

Project files

Searches all files that you have explicitly added to your project.

Project files and user include files

Searches all files that you have explicitly added to your project and all files that they include, except the include files in the IAR Embedded Workbench installation directory.

Project files and all include files

Searches all project files that you have explicitly added to your project and all files that they include.

Directory

Searches the directory that you specify. Recent search locations are saved in the drop-down list. A browse button is available for your convenience.

Look in subdirectories

Searches the directory that you have specified and all its subdirectories.

File types

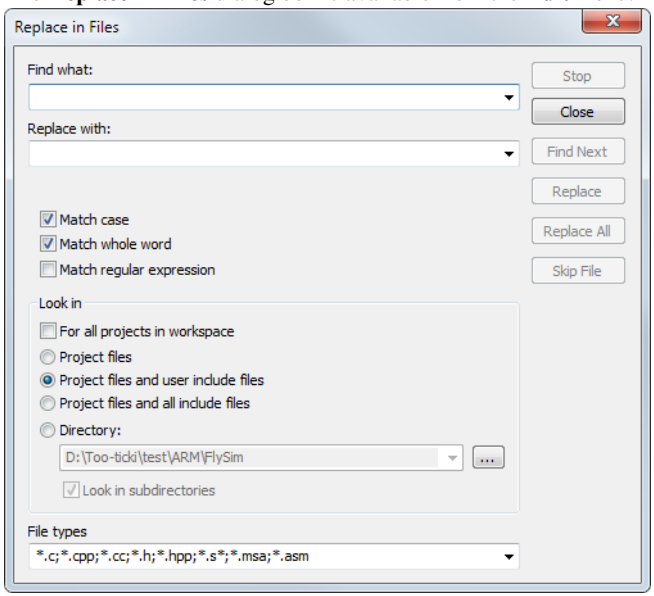
A filter for choosing which type of files to search; the filter applies to all **Look in** settings. Choose the appropriate filter from the drop-down list. The text field is editable, to let you add your own filters. Use the `*` character to indicate zero or more unknown characters of the filters, and the `?` character to indicate one unknown character.

Stop

Stops an ongoing search. This button is only available during an ongoing search.

Replace in Files dialog box

The **Replace in Files** dialog box is available from the **Edit** menu.



Use this dialog box to search for a specified string in multiple text files and replace it with another string.

The result of the replacement appears in the **Find in Files** message window—available from the **View** menu. You can then go to each occurrence by choosing the **Edit>Next Error/Tag** command, alternatively by double-clicking the messages in the **Find in Files** message window. This opens the corresponding file in an editor window with the

insertion point positioned at the start of the specified text. A blue flag in the left-hand margin indicates the line containing the string you searched for.

Find what

Specify the string you want to search for and replace, or a regular expression. Use the drop-down list to use old search strings/expressions. You can narrow the search down with one or more of these conditions:

Match case

Searches only for occurrences that exactly match the case of the specified text. Otherwise, specifying `int` will also find `INT` and `Int`.

Match whole word

Searches only for the string when it occurs as a separate word (mnemonic `&w`). Otherwise, `int` will also find `print`, `sprintf`, and so on.

Match regular expression

Interprets the search string as a regular expression, which must follow the standard for the Perl programming language.

Replace with

Specify the string you want to replace the original string with. Use the drop-down list to use old replace strings.

Look in

Specify which files you want to search in. Choose between:

For all projects in workspace

Searches all projects in the workspace, not just the active project.

Project files

Searches all files that you have explicitly added to your project.

Project files and user include files

Searches all files that you have explicitly added to your project and all files that they include, except the include files in the IAR Embedded Workbench installation directory.

Project files and all include files

Searches all project files that you have explicitly added to your project and all files that they include.

Directory

Searches the directory that you specify. Recent search locations are saved in the drop-down list. A browse button is available for your convenience.

Look in subdirectories

Searches the directory that you have specified and all its subdirectories.

File types

A filter for choosing which type of files to search; the filter applies to all **Look in** settings. Choose the appropriate filter from the drop-down list. The text field is editable, to let you add your own filters. Use the `*` character to indicate zero or more unknown characters of the filters, and the `?` character to indicate one unknown character.

Stop

Stops an ongoing search. This button is only available during an ongoing search.

Close

Closes the dialog box. An ongoing search must be stopped first.

Find Next

Finds the next occurrence of the specified search string.

Replace

Replaces the found string and finds the next occurrence of the specified search string.

Replace All

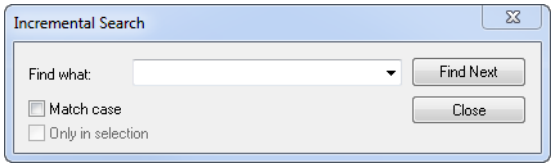
Saves all files and replaces all found strings that match the search string.

Skip file

Skips the occurrences in the current file.

Incremental Search dialog box

The **Incremental Search** dialog box is available from the **Edit** menu.



Use this dialog box to gradually fine-tune or expand the search string.

Find what

Type the string to search for. The search is performed from the location of the insertion point—the *start point*. Every character you add to or remove from the search string instantly changes the search accordingly. If you remove a character, the search starts over again from the start point.

If a word in the editor window is selected when you open the **Incremental Search** dialog box, this word will be displayed in the **Find What** text box.

Use the drop-down list to use old search strings.

Match case

Searches for occurrences that exactly match the case of the specified text. Otherwise, searching for `int` will also find `INT` and `Int`.

Find Next

Searches for the next occurrence of the current search string. If the **Find What** text box is empty when you click the **Find Next** button, a string to search for will automatically be selected from the drop-down list. To search for this string, click **Find Next**.

Close

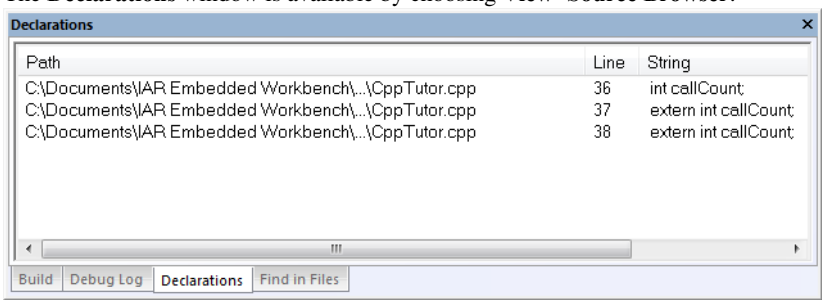
Closes the dialog box.

Only in selection

Limits the search operation to the selected lines. The option is only available when more than one line has been selected before you open the dialog box.

Declarations window

The **Declarations** window is available by choosing **View>Source Browser**.



This window displays the result from the **Go to Declaration** command on the editor window context menu.

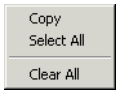
When opened, this window is by default grouped together with the other message windows.

To find and list declarations for a specific symbol, select a symbol in the editor window, right-click and choose **Go to Declaration** from the context menu. All declarations are listed in the **Declarations** window.

Double-click an entry in the window to open the corresponding file with the insertion point positioned at the correct location. Choose **Edit>Next Error/Tag** or press F4 to jump to the next in sequence.

Context menu

This context menu is available:



These commands are available:

Copy

Copies the contents of the window.

Select All

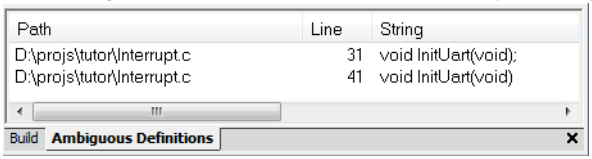
Selects the contents of the window.

Clear All

Deletes the contents of the window.

Ambiguous Definitions window

The **Ambiguous Definitions** window is available by choosing **View>Source Browser**.



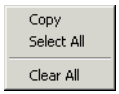
This window displays the result from the **Go to Definition** command on the editor window context menu, if the source browser finds more than one possible definition.

When opened, this window is by default grouped together with the other message windows.

Double-click an entry in the window to open the corresponding file with the insertion point positioned at the correct location. Choose **Edit>Next Error/Tag** or press F4 to jump to the next entry in sequence.

Context menu

This context menu is available:



These commands are available:

Copy

Copies the contents of the window.

Select All

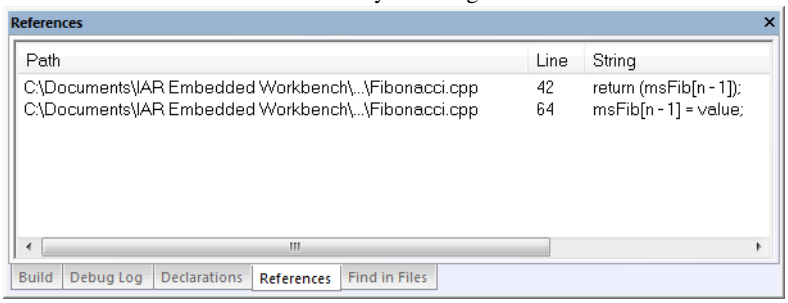
Selects the contents of the window.

Clear All

Deletes the contents of the window.

References window

The **References** window is available by choosing **View>Source Browser**.



This window displays the result from the **Find All References** commands on the editor window context menu.

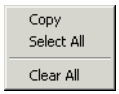
When opened, this window is by default grouped together with the other message windows.

To find and list references for a specific symbol, select a symbol in the editor window, right-click and choose **Find All References** from the context menu. All references are listed in the **References** window.

Double-click an entry in the window to open the corresponding file with the insertion point positioned at the correct location. Choose **Edit>Next Error/Tag** or press F4 to jump to the next in sequence.

Context menu

This context menu is available:



These commands are available:

Copy

Copies the contents of the window.

Select All

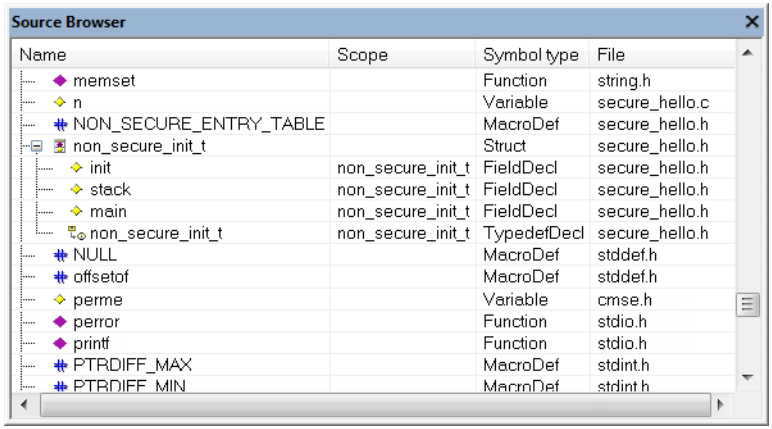
Selects the contents of the window.

Clear All

Deletes the contents of the window.

Source Browser window

The **Source Browser** window is available from the **View** menu.



This window displays an hierarchical view in alphabetical order of all symbols defined in the active build configuration. This means that source browse information is available for symbols in source files and include files part of that configuration. Source browse information is not available for symbols in linked libraries.

For more information about how to use this window, see *Displaying source browse information*, page 134.

The display area

The display area contains four columns:

Name	The names of global symbols and functions defined in the project. Note that an unnamed type, for example a `struct` or a `union` without a name, will get a name based on the filename and line number where it is defined. These pseudonames are enclosed in angle brackets.
Scope	The scope (namespaces and classes/structs) that the entry belongs to.
Symbol type	Displays the symbol type for each element.
File	The file name (without path) that contains the definition of the entry.

To sort each column, click its header.

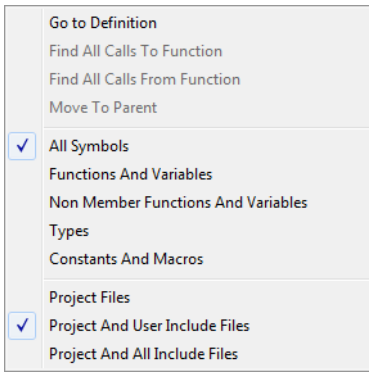
Icons used for the symbol types

These are the icons used:

		Base class
		Class
		Configuration
		Enumeration
		Enumeration constant
	(Yellow rhomb)	Field of a struct
	(Purple rhomb)	Function
		Macro
		Namespace
		Template class
		Template function
		Type definition
		Union
	(Yellow rhomb)	Variable

Context menu

This context menu is available in the display area:



These commands are available:

Go to Definition

The editor window will display the definition of the selected item.

Find All Calls to

Opens the **Call Graph** window which displays all functions in the project that calls the selected function, see *Call Graph window*, page 162. If this command is disabled, make sure to select a function in the **Source Browser** window.

Find All Calls from

Opens the **Call Graph** window which displays all functions in the project that are called from the selected function, see *Call Graph window*, page 162. If this command is disabled, make sure to select a function in the **Source Browser** window.

Move to Parent

If the selected element is a member of a class, struct, union, enumeration, or namespace, this menu command can be used for moving the insertion point to the enclosing element.

All Symbols

Type filter; displays all global symbols and functions defined in the project.

Functions and Variables

Type filter; displays all functions and variables defined in the project.

Non-Member Functions and Variables

Type filter; displays all functions and variables that are not members of a class.

Types

Type filter; displays all types such as structures and classes defined in the project.

Constants and Macros

Type filter; displays all constants and macros defined in the project.

Project Files

File filter; displays symbols from all files that you have explicitly added to your project, but no include files.

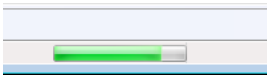
Project and User Include Files

File filter; displays symbols from all files that you have explicitly added to your project and all files included by them, except the include files in the IAR Embedded Workbench installation directory.

Project and All Include Files

File filter; displays symbols from all files that you have explicitly added to your project and all files included by them.

Progress bar

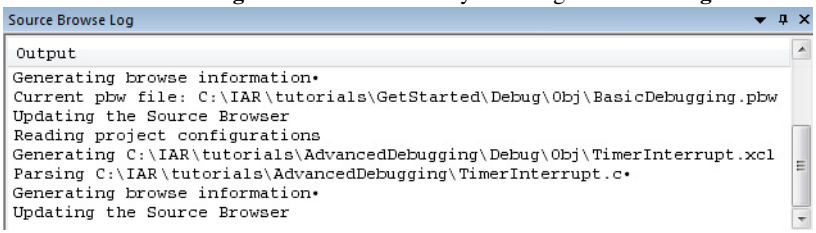


While the source browse information is generated for a project, a green progress bar is displayed in the status bar of the IDE window. Clicking on this progress bar opens a context menu with a command to open the **Source Browse Log** window, see *Source Browse Log window*, page 160.

If the source browser encounters a fatal error, the progress bar turns red.

Source Browse Log window

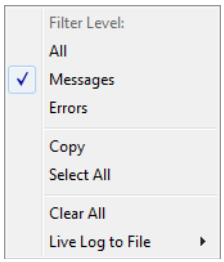
The **Source Browse Log** window is available by choosing **View>Messages**.



This window displays the output from the operation of the source browser.

Context menu

This context menu is available:



These commands are available:

All

Shows all messages sent by the source browser. This is mainly useful as input to IAR Systems technical support.

Messages

Gives information about what the source browser is doing and any errors that occur during parsing.

Errors

Shows only errors received during the source browsing.

Copy

Copies the contents of the window.

Select All

Selects the contents of the window.

Clear All

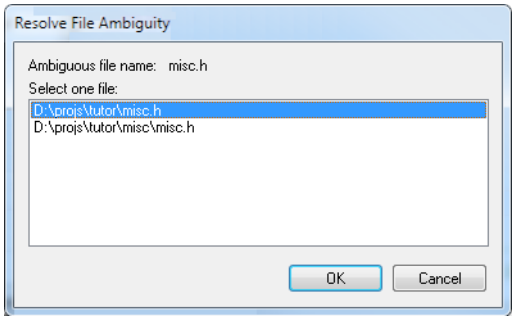
Clears the contents of the window.

Live Log to File

Displays a submenu with commands for writing the source browse messages to a log file, and setting filter levels for the log.

Resolve File Ambiguity dialog box

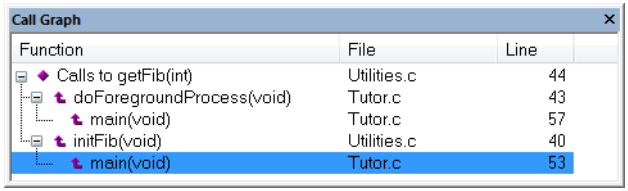
The **Resolve File Ambiguity** dialog box is displayed when the editor finds more than one header file with the same name.



This dialog box lists the header files if more than one header file is found when you choose the **Open "*header*.h"** command on the editor window context menu and the IDE does not have access to dependency information.

Call Graph window

The **Call Graph** window is available by choosing **View>Source Browser>Call Graph**.



This window displays calls to or calls from a function. The window is useful for navigating between the function calls.

To display a call graph, select a function name in the editor window or in the **Source Browser** window, right-click and select either **Find All Calls to** or **Find All Calls from** from the context menu.

Double-click an entry in the window to place the insertion point at the location of the function call (or definition, if a call is not applicable for the entry). The editor will open the file that contains the call if necessary.

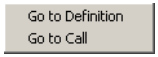
Display area

The display area shows the call graph for the selected function, where each line lists a function. These columns are available:

Function	Displays the call graph for the selected function; first the selected function, followed by a list of all called or calling functions. The functions calling the selected function are indicated with left arrow and the functions called by the selected function are indicated with a right arrow.
File	The name of the source file.
Line	The line number for the call.

Context menu

This context menu is available:



These commands are available:

Go to Definition

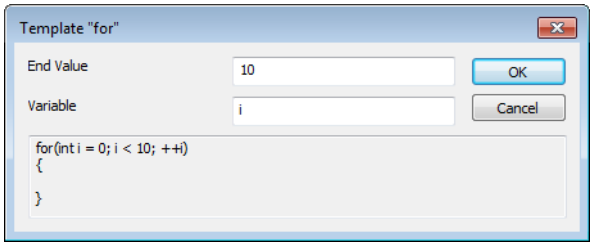
Places the insertion point at the location of the function definition.

Go to Call

Places the insertion point at the location of the function call.

Template dialog box

The **Template** dialog box appears when you insert a code template that requires any field input.



Use this dialog box to specify any field input that is required by the source code template you insert.

Note: The figure reflects the default code template that can be used for automatically inserting code for a `for` loop.

Text fields

Specify the required input in the text fields. Which fields that appear depends on how the code template is defined.

Display area

The display area shows the code that would result from the code template, using the values you submit. For more information about using code templates, see *Using and adding code templates*, page 130.

Editor shortcut key summary

There are three types of shortcut keys that you can use in the editor:

- Predefined shortcut keys, which you can edit using the **IDE Options** dialog box
- Shortcut keys provided by the Scintilla editor
- Custom shortcut keys that you can add using the **IDE Options** dialog box.

The following tables summarize the editor's predefined shortcut keys.

Moving the insertion point

To move the insertion point	Press
One character to the left	Left arrow
One character to the right	Right arrow
One word to the left	Ctrl + Left arrow
One word to the right	Ctrl + Right arrow
One word part to the left; when using mixed cases, for example mixedCaseName	Ctrl + Alt + Left arrow
One word part to the right; when using mixed cases, for example mixedCaseName	Ctrl + Alt + Right arrow
One line up	Up arrow
One line down	Down arrow
To the previous paragraph	Ctrl + Alt + Up arrow
To the next paragraph	Ctrl + Alt + Down arrow
To the start of the line	Home
To the end of the line	End
To the beginning of the file	Ctrl + Home
To the end of the file	Ctrl + End

Table 5: Editor shortcut keys for insertion point navigation

Selecting text

To select text, press Shift and the corresponding command for moving the insertion point. In addition, this command is available:

To select	Press
A column-based block	Shift + Alt + Arrow key

Table 6: Editor shortcut keys for selecting text

Scrolling text

To scroll	Press
Up one line. When used in the parameter hints text box, this shortcut steps up one line through the alternatives.	Ctrl + Up arrow
Down one line, When used in the parameter hints text box, this shortcut steps down one line through the alternatives.	Ctrl + Down arrow
Up one page	Page Up
Down one page	Page Down

Table 7: Editor shortcut keys for scrolling

Miscellaneous shortcut keys

Description	Press
When used in the parameter hints text box, this shortcut inserts parameters as text in the source code.	Ctrl + Enter
Bracket matching: Expand selection to next level of matching of {}, [], or ().	Ctrl + B
Bracket matching: Expand selection to next level of matching of {}, [], (), or <>.	Ctrl + Alt + B
Bracket matching: Shrink selection to next level of matching of {}, [], or ().	Ctrl + Shift + B
Bracket matching: Shrink selection to next level of matching of {}, [], (), or <>.	Ctrl + Alt + Shift + B
Change case for selected text to lower	Ctrl + u
Change case for selected text to upper	Ctrl + U

Table 8: Miscellaneous editor shortcut keys

Description	Press
Complete code	Ctrl + Space
Complete word	Ctrl + Alt + Space
Insert template	Ctrl + Alt + V
Parameter hint	Ctrl + Shift + Space
Zooming	Mouse wheel
Zoom in	Ctrl + numeric keypad '+'
Zoom out	Ctrl + numeric keypad '-'
Zoom normal	Ctrl + numeric keypad '/'

Table 8: Miscellaneous editor shortcut keys (Continued)

Additional Scintilla shortcut keys

Description	Press
Scroll window line up or down	Ctrl + Up Ctrl + Down
Select a rectangular block and change its size a line up or down, or a column left or right	Shift + Alt + arrow key
Move insertion point one paragraph up or down	Ctrl + Alt + Up Ctrl + Alt + Down
Grow selection one paragraph up or down	Ctrl + Shift + Alt + Up Ctrl + Shift + Alt + Down
Move insertion point one word left or right	Ctrl + Left Ctrl + Right
Grow selection one word left or right	Ctrl + Shift + Left Ctrl + Shift + Right
Grow selection to next start or end of a word	Ctrl + Shift + Alt + Left Ctrl + Shift + Alt + Right
Move to first non-blank character of the line	Home
Move to start of line	Alt + Home
Select to start of the line	Shift + Alt + Home
Select a rectangular block to the start or end of page	Shift + Alt + Page Up Shift + Alt + Page Down
Delete to start of next word	Ctrl + Delete
Delete to start of previous word	Ctrl + Backspace
Delete forward to end of line	Ctrl + Shift + Delete

Table 9: Additional Scintilla shortcut keys

Description	Press
Delete backward to start of line	Ctrl + Shift + Backspace
Zoom in	Ctrl + Add (numeric +)
Zoom out	Ctrl + Subtract (numeric –)
Restore zoom to 100%	Ctrl + Divide (numeric /)
Cut current line	Ctrl + L
Copy current line	Ctrl + Shift + T
Delete current line	Ctrl + Shift + L
Change selection to lower case	Ctrl + U
Change selection to upper case	Ctrl + Shift + U

Table 9: Additional Scintilla shortcut keys (Continued)

Part 2. Reference information

This part contains these chapters:

- Product files
- Menu reference
- General options
- Compiler options
- Assembler options
- Output converter options
- Custom build options
- Build actions options
- Linker options
- Library builder options

Product files

- Installation directory structure
- Project directory structure
- Various settings files
- File types

Installation directory structure

These topics are covered:

- Root directory
- The rh850 directory
- The common directory
- The install-info directory

The installation procedure creates several directories to contain the various types of files used with the IAR Systems development tools. The following sections give a description of the files contained by default in each directory.

ROOT DIRECTORY

The default installation root directory is typically *x*:\Program Files\IAR Systems\Embedded Workbench *N.n*\, where *x* is the drive where Microsoft Windows is installed, and the first digit in *N.n* reflects the first digit in the version number of the IAR Embedded Workbench shared components.

Note that this version number is not the same as the version number of your IAR Embedded Workbench product. To find the version number of the IDE and the product, see *Product Info dialog box*, page 80.

THE RH850 DIRECTORY

The `rh850` directory contains all product-specific subdirectories.

Directory	Description
`rh850\bin`	Contains executable files for RH850-specific components, such as the compiler, the assembler, the linker and the library tools, and the C-SPY® drivers.
`rh850\config`	Contains files used for configuring the development environment and projects, for example: • Linker configuration files (`*.icf`) • Special function register description files (`*.sfr`) • C-SPY device description files (`*.ddf`) • Device selection files (`*.menu`) • Device files (`*.dvf`) • Syntax coloring configuration files (`*.cfg`) • Project templates for both application and library projects (`*.ewp`), and for the library projects, the corresponding library configuration files.
`rh850\cstat`	Contains files related to C-STAT.
`rh850\doc`	Contains online versions in hypertext PDF format of this user guide, and of the RH850 reference guides, as well as online help files (`*.chm`). The directory also contains release notes with recent additional information about the RH850 tools.
`rh850\drivers`	Contains low-level device drivers, typically USB drivers required by the C-SPY drivers.
`rh850\examples`	Contains files related to example projects, which can be opened from the Information Center.
`rh850\inc`	Contains include files, such as the header files for the standard C or C++ library. There are also specific header files that define special function registers (SFRs); these files are used by both the compiler and the assembler.
`rh850\lib`	Contains prebuilt libraries and the corresponding library configuration files, used by the compiler.
`rh850\plugins`	Contains executable files and description files for components that can be loaded as plugin modules.
`rh850\rtos`	Contains product information, evaluation versions, and example projects for third-party RTOS and middleware solutions integrated into IAR Embedded Workbench.

Table 10: The rh850 directory

Directory	Description
rh850\src	Contains source files for some configurable library functions and the library source code. For the ILINK linker, the directory also contains the source code for ELF utilities.
rh850\tutorials	Contains the files used for the tutorials in the Information Center.

Table 10: The rh850 directory (Continued)

THE COMMON DIRECTORY

The common directory contains subdirectories for components shared by all IAR Embedded Workbench products.

Directory	Description
common\bin	Contains executable files for components common to all IAR Embedded Workbench products, such as the editor and the graphical user interface components. The executable file for the IDE is also located here.
common\config	Contains files used by the IDE for settings in the development environment.
common\doc	Contains release notes with recent additional information about the components common to all IAR Embedded Workbench products. We recommend that you read these files. The directory also contains documentation related to installation and licensing.
common\plugins	Contains executable files and description files for components that can be loaded as plugin modules, for example modules for code coverage.

Table 11: The common directory

THE INSTALL-INFO DIRECTORY

The install-info directory contains metadata (version number, name, etc.) about the installed product components. Do not modify these files.

Project directory structure

When you build your project, the IDE creates new directories in your project directory. A subdirectory is created; the name of this directory reflects the build configuration you are using, typically Debug or Release. This directory in turn contains these subdirectories:

List The destination directory for various list files.

`Obj`	The destination directory for the object files from the compiler and assembler. The object files have the extension `o` and are used as input to the linker.
`Exe`	The destination directory for: • The executable file, which has the extension `out` and is used as input to the IAR C-SPY® Debugger. • Library object files, which have the extension `a`.

Various settings files

When you work in the IDE, the IDE creates files for various types of settings. These files are stored in different directories depending on whether the files contain global or local settings.

FILES FOR GLOBAL SETTINGS

Files for *global* settings are stored in `C:\Users\User\AppData\Local\IAR Embedded Workbench`. These are the global settings files:

`CodeTemplates.txt`
`CodeTemplates.ENU.txt`
`CodeTemplates.JPN.txt`
	A file that holds predefined code templates.
	Note that if you are using an IDE that is available in languages other than English, you are asked to select a language version when you start the IAR Embedded Workbench for the first time. In this case, the filename is extended with `ENU` or `JPN`, depending on your choice of language (English or Japanese).
	See also *Using and adding code templates*, page 130.

`global.custom_argvars`	A file that holds any custom argument variables that are defined for a global scope. See also *Configure Custom Argument Variables dialog box*, page 83.
`IarIde.xml`	A file that holds IDE and project settings global to your installed IAR Embedded Workbench product(s).

FILES FOR LOCAL SETTINGS

Files for *local* settings are stored in the directory `settings`, which is created in your project directory. These are the local settings files:

Project.dbgdt	A file for debugger desktop settings.
Project.Buildconfig.cspy.bat	A batch file that C-SPY creates every time it is invoked.
Project.Buildconfig.driver.xcl	A file that C-SPY creates every time it is invoked, and which contains the command line options used that are specific to the C-SPY driver you are using.
Project.Buildconfig.general.xcl	A file that C-SPY creates every time it is invoked, and which contains the command line options used that are specific to `cspybat`.
Project.dnx	A file for debugger initialization information.
Workspace.wsdt	A file for workspace desktop settings.
Workspace.wspos	A file for placement information for the main IDE window.
Workspace.custom_argvars	A file for any custom argument variables that are defined for a workspace-local scope. See also *Configure Custom Argument Variables dialog box*, page 83.

File types

The IAR Systems development tools use the following default filename extensions to identify the produced files and other recognized file types:

Ext.	Type of file	Output from	Input to
a	Library	iarchive	ILINK
asm	Assembler source code	Text editor	Assembler
bat	Windows command batch file	C-SPY	Windows
c	C source code	Text editor	Compiler

Table 12: File types

Ext.	Type of file	Output from	Input to
cfg	Syntax coloring configuration	Text editor	IDE
cgx	Call graph file	ILINK	–
chm	Online help system file	--	IDE
cpp	C++ source code	Text editor	Compiler
cspy.bat	Invocation file for cspybat	C-SPY	–
dat	Macros for formatting of STL containers	IDE	IDE
dbgdt	Debugger desktop settings	C-SPY	C-SPY
ddf	Device description file	Text editor	C-SPY
dep	Dependency information	IDE	IDE
dnx	Debugger initialization file	C-SPY	C-SPY
dvf	Device file	--	C-SPY
ewd	Project settings for C-SPY	IDE	IDE
ewp	IAR Embedded Workbench project (current version)	IDE	IDE
ewplugin	IDE description file for plugin modules	--	IDE
ewt	Project settings for C-STAT and C-RUN	IDE	IDE
eww	Workspace file	IDE	IDE
fmt	Formatting information for the **Locals** and **Watch** windows	IDE	IDE
h	C/C++ or assembler header source	Text editor	Compiler or assembler #include
helpfiles	Help menu configuration file	Text editor	IDE
html, htm	HTML document	Text editor	IDE
i	Preprocessed source	Compiler	Compiler
icf	Linker configuration file	Text editor	ILINK
inc	Assembler header source	Text editor	Assembler #include
ini	Project configuration	IDE	–
log	Log information	IDE	–
lst	List output	Compiler and assembler	–
mac	C-SPY macro definition	Text editor	C-SPY

Table 12: File types (Continued)

Ext.	Type of file	Output from	Input to
menu	Device selection file	Text editor	IDE
o	Object module	Compiler and assembler	ILINK
out	Target application	ILINK	EPROM, C-SPY, etc.
out	Target application with debug information	ILINK	C-SPY and other symbolic debuggers
pbd	Source browse information	IDE	IDE
pbi	Source browse information	IDE	IDE
pew	IAR Embedded Workbench project (old project format)	IDE	IDE
prj	IAR Embedded Workbench project (old project format)	IDE	IDE
reggroups	User-defined register group configuration	IDE	IDE
s	Assembler source code	Text editor	Assembler
sfr	Special function register definitions	Text editor	C-SPY
sim	Simple code formatted input for the flash loader	C-SPY	C-SPY
suc	Stack usage control file	Text editor	ILINK
vsp	visualSTATE project files	IAR visualSTATE Designer	IAR visualSTATE Designer and IAR Embedded Workbench IDE
wsdt	Workspace desktop settings	IDE	IDE
wspos	Main IDE window placement information	IDE	IDE
xcl	Extended command line	Text editor	Assembler, compiler, linker, cspybat, source browser

Table 12: File types (Continued)

When you run the IDE, some files are created and located in dedicated directories under your project directory, by default `$PROJ_DIR$\Debug`, `$PROJ_DIR$\Release`, `$PROJ_DIR$\settings`. None of these directories or files affect the execution of the IDE, which means you can safely remove them if required.

Menu reference

- Menus

Menus

Reference information about:

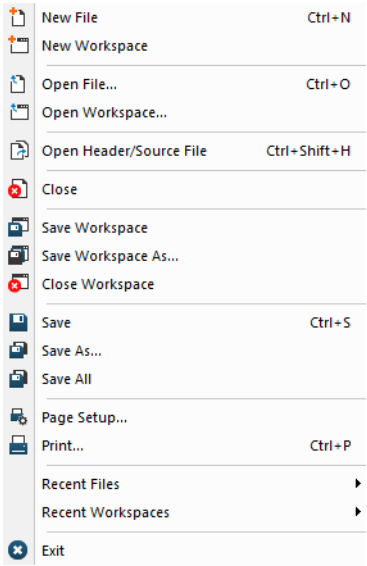
- File menu
- Edit menu
- View menu
- Project menu
- Tools menu
- Window menu
- Help menu

In addition, a set of C-SPY-specific menus become available when you start the debugger. For more information about these menus, see the *C-SPY® Debugging Guide for RH850*.

File menu

The **File** menu provides commands for opening workspaces and source files, saving and printing, and exiting from the IDE.

The menu also includes a numbered list of the most recently opened files and workspaces. To open one of them, choose it from the menu.



Menu commands

These commands are available:

New File (Ctrl+N)

Creates a new text file.

New Workspace

Creates a new workspace.

Open File (Ctrl+O)

Displays an **Open** dialog box for selecting a text file or an HTML document to open. See *Editor window*, page 137.

Open Workspace

Displays an **Open Workspace** dialog box for selecting a workspace file to open. Before a new workspace is opened you will be prompted to save and close any currently open workspaces.

Open Header/Source File (Ctrl+Shift+H)

Opens the header file or source file that corresponds to the current file, and shifts focus from the current file to the newly opened file. This command is also available on the context menu in the editor window.

Close

Closes the active window. You will be given the opportunity to save any files that have been modified before closing.

Save Workspace

Saves the current workspace file.

Save Workspace As

Displays a **Save Workspace As** dialog box for saving the workspace with a new name.

Close Workspace

Closes the current workspace file.

Save (Ctrl+S)

Saves the current text file or workspace file.

Save As

Displays a **Save As** dialog box where you can save the current file with a new name.

Save All

Saves all open text documents and workspace files.

Page Setup

Displays a **Page Setup** dialog box where you can set printer options.

Print (Ctrl+P)

Displays a **Print** dialog box where you can print a text document.

Recent Files

Displays a submenu from where you can quickly open the most recently opened text documents.

Recent Workspaces

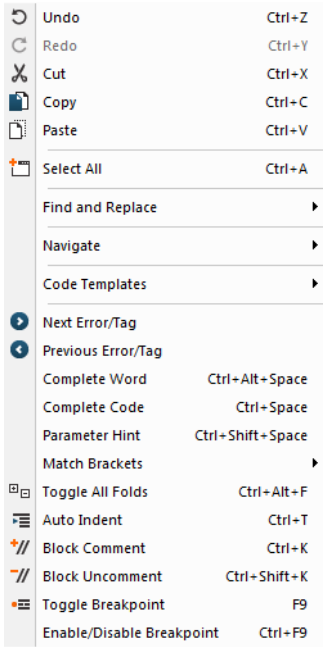
Displays a submenu from where you can quickly open the most recently opened workspace files.

Exit

Exits from the IDE. You will be asked whether to save any changes to text files before closing them. Changes to the project are saved automatically.

Edit menu

The **Edit** menu provides commands for editing and searching.



Menu commands

These commands are available:

Undo (Ctrl+Z)

Undoes the last edit made to the current editor window.

Redo (Ctrl+Y)

Redoes the last Undo in the current editor window. You can undo and redo an unlimited number of edits independently in each editor window.

Cut (Ctrl+X)

The standard Windows command for cutting text in editor windows and text boxes.

Copy (Ctrl+C)

The standard Windows command for copying text in editor windows and text boxes.

Paste (Ctrl+V)

The standard Windows command for pasting text in editor windows and text boxes.

Select All (Ctrl+A)

Selects all text in the active editor window.

Find and Replace>Find (Ctrl+F)

Displays the **Find** dialog box where you can search for for text within the current editor window, see *Find dialog box*, page 146. Note that if the insertion point is located in the **Memory** window when you choose the **Find** command, the dialog box will contain a different set of options than otherwise. If the insertion point is located in the **Trace** window when you choose the **Find** command, the **Find in Trace** dialog box is opened; the contents of this dialog box depend on the C-SPY driver you are using, see the *C-SPY® Debugging Guide for RH850* for more information.

Find and Replace>Find Next (F3)

Finds the next occurrence of the specified string.

Find and Replace>Find Previous (Shift+F3)

Finds the previous occurrence of the specified string.

Find and Replace>Find Next (Selected) (Ctrl+F3)

Searches for the next occurrence of the currently selected text or the word currently surrounding the insertion point.

Find and Replace>Find Previous (Selected) (Ctrl+Shift+F3)

Searches for the previous occurrence of the currently selected text or the word currently surrounding the insertion point.

Find and Replace>Replace (Ctrl+H)

Displays a dialog box where you can search for a specified string and replace each occurrence with another string, see *Replace dialog box*, page 148.

Note that if the insertion point is located in the **Memory** window when you choose the **Replace** command, the dialog box will contain a different set of options than otherwise.

Find and Replace>Find in Files

Displays a dialog box where you can search for a specified string in multiple text files, see *Find in Files window*, page 147.

Find and Replace>Replace in Files

Displays a dialog box where you can search for a specified string in multiple text files and replace it with another string, see *Replace in Files dialog box*, page 151.

Find and Replace>Incremental Search (Ctrl+I)

Displays a dialog box where you can gradually fine-tune or expand the search by continuously changing the search string, see *Incremental Search dialog box*, page 153.

Navigate>Go To (Ctrl+G)

Displays the **Go to Line** dialog box where you can move the insertion point to a specified line and column in the current editor window.

Navigate>Toggle Bookmark (Ctrl+F2)

Toggles a bookmark at the line where the insertion point is located in the active editor window.

Navigate>Previous Bookmark (Shift+F2)

Moves the insertion point to the previous bookmark that has been defined with the **Toggle Bookmark** command.

Navigate>Next Bookmark (F2)

Moves the insertion point to the next bookmark that has been defined with the **Toggle Bookmark** command.

Navigate>Navigate Backward (Alt+Left Arrow)

Navigates backward in the insertion point history. The current position of the insertion point is added to the history by actions like **Go to definition** and clicking on a result from the **Find in Files** command.

Navigate>Navigate Forward (Alt+Right Arrow)

Navigates forward in the insertion point history. The current position of the insertion point is added to the history by actions like **Go to definition** and clicking on a result from the **Find in Files** command.

Navigate>Go to Definition (F12)

Shows the declaration of the selected symbol or the symbol where the insertion point is placed. This menu command is available when browse information has been enabled, see *Project options*, page 61.

Code Templates>Insert Template (Ctrl+Alt+V)

Displays a list in the editor window from which you can choose a code template to be inserted at the location of the insertion point. If the code template you choose requires any field input, the **Template** dialog box appears, see *Template dialog box*, page 163. For information about using code templates, see *Using and adding code templates*, page 130.

Code Templates>Edit Templates

Opens the current code template file, where you can modify existing code templates and add your own code templates. For information about using code templates, see *Using and adding code templates*, page 130.

Next Error/Tag (F4)

If the message window contains a list of error messages or the results from a **Find in Files** search, this command displays the next item from that list in the editor window.

Previous Error/Tag (Shift+F4)

If the message window contains a list of error messages or the results from a **Find in Files** search, this command displays the previous item from that list in the editor window.

Complete Word (Ctrl+Alt+Space)

Attempts to complete the word you have begun to type, basing the guess on the contents of the rest of the editor document.

Complete Code (Ctrl+Space)

Shows a list of symbols that are available in a class, when you place the insertion point after `.`, `->`, or `::` and when these characters are preceded by a class or object name. For more information, see *Code completion*, page 129.

Parameter Hint (Ctrl+Shift+Space)

Suggests parameters as tooltip information for the function parameter list you have begun to type. When there are several overloaded versions of a function, you can choose which one to use by clicking the arrows in the tooltip. For more information, see *Parameter hint*, page 129.

Match Brackets

Selects all text between the brackets immediately surrounding the insertion point, increases the selection to the next hierarchic pair of brackets, or beeps if there is no higher bracket hierarchy.

Toggle All Folds (Ctrl+Alt+F)

Expands/collapses all code folds in the active project.

Auto Indent (Ctrl+T)

Indents one or several lines you have selected in a C/C++ source file. To configure the indentation, see *Configure Auto Indent dialog box*, page 55.

Block Comment (Ctrl+K)

Places the C++ comment character sequence `//` at the beginning of the selected lines.

Block Uncomment (Ctrl+Shift+K)

Removes the C++ comment character sequence `//` from the beginning of the selected lines.

Toggle Breakpoint (F9)

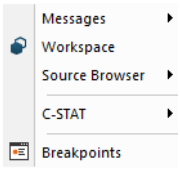
Toggles a breakpoint at the statement or instruction that contains or is located near the cursor in the source window. This command is also available as an icon button on the debug toolbar.

Enable/Disable Breakpoint (Ctrl+F9)

Toggles a breakpoint between being disabled, but not actually removed—making it available for future use—and being enabled again.

View menu

The **View** menu provides several commands for opening windows in the IDE. When C-SPY is running you can also open debugger-specific windows from this menu. See the *C-SPY® Debugging Guide for RH850* for information about these.



Menu commands

These commands are available:

Messages

Displays a submenu which gives access to the message windows—**Build**, **Find in Files**, **Source Browse Log**, **Tool Output**, **Debug Log**—that display messages and text output from the IAR Embedded Workbench commands. If the window you choose from the menu is already open, it becomes the active window.

Workspace

Opens the current **Workspace** window, see *Workspace window*, page 97.

Source Browser>Source Browser

Opens the **Source Browser** window, see *Source Browser window*, page 157.

Source Browser>References

Opens the **References** window, see *References window*, page 156.

Source Browser>Declarations

Opens the **Declarations** window, see *Declarations window*, page 154.

Source Browser>Ambiguous Definitions

Opens the **Ambiguous Definitions** window, see *Ambiguous Definitions window*, page 155.

Source Browser>Call Graph

Opens the **Call Graph** window, see *Call Graph window*, page 162.

C-STAT>C-STAT Messages

Opens the **C-STAT Messages** window, see the *C-STAT® Static Analysis Guide*.

Breakpoints

Opens the **Breakpoints** window, see the *C-SPY® Debugging Guide for RH850*.

Call Stack

Opens the **Call Stack** window. Only available when C-SPY is running.

Watch

Opens an instance of the **Watch** window from a submenu. Only available when C-SPY is running.

Live Watch

Opens the **Live Watch** window. Only available when C-SPY is running.

Quick Watch

Opens the **Quick Watch** window. Only available when C-SPY is running.

Auto

Opens the **Auto** window. Only available when C-SPY is running.

Locals

Opens the **Locals** window. Only available when C-SPY is running.

Statics

Opens the **Statics** window. Only available when C-SPY is running.

Memory

Opens an instance of the **Memory** window from a submenu. Only available when C-SPY is running.

Registers

Displays a submenu which gives access to the Registers windows—**Registers** and **Register User Groups Setup**. Only available when C-SPY is running.

Disassembly

Opens the **Disassembly** window. Only available when C-SPY is running.

Stack

Opens an instance of the **Stack** window from a submenu. Only available when C-SPY is running.

Symbolic Memory

Opens the **Symbolic Memory** window. Only available when C-SPY is running.

Terminal I/O

Opens the **Terminal I/O** window. Only available when C-SPY is running.

Macros>Macro Quicklaunch

Opens the **Macro Quicklaunch** window. Only available when C-SPY is running.

Macros>Macro Registration

Opens the **Macro Registration** window. Only available when C-SPY is running.

Macros>Debugger Macros

Opens the **Debugger Macros** window. Only available when C-SPY is running.

Symbols

Opens the **Symbols** window. Only available when C-SPY is running.

Code Coverage

Opens the **Code Coverage** window. Only available when C-SPY is running.

Images

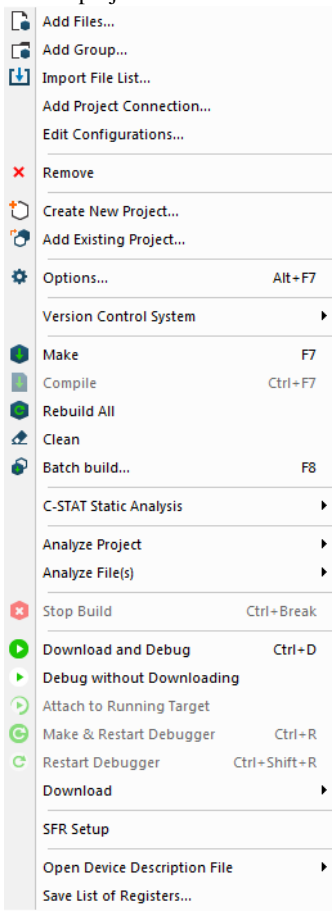
Opens the **Images** window. Only available when C-SPY is running.

Cores

Opens the **Cores** window. Only available when C-SPY is running.

Project menu

The **Project** menu provides commands for working with workspaces, projects, groups, and files, and for specifying options for the build tools, and running the tools on the current project.



Menu commands

These commands are available:

Add Files

Displays a dialog box where you can select which files to include in the current project.

Add Group

Displays a dialog box where you can create a new group. In the **Group Name** text box, specify the name of the new group. For more information about groups, see *Groups*, page 91.

Import File List

Displays a standard **Open** dialog box where you can import information about files and groups from projects created using another IAR Systems toolchain.

To import information from project files which have one of the older filename extensions `pew` or `prj` you must first have exported the information using the context menu command **Export File List** available in your current IAR Embedded Workbench.

Add Project Connection

Displays the **Add Project Connection** dialog box, see *Add Project Connection dialog box*, page 105.

Edit Configurations

Displays the **Configurations for project** dialog box, where you can define new or remove existing build configurations. See *Configurations for project dialog box*, page 103.

Remove

In the **Workspace** window, removes the selected item from the workspace.

Create New Project

Displays the **Create New Project** dialog box where you can create a new project and add it to the workspace, see *Create New Project dialog box*, page 102.

Add Existing Project

Displays a standard **Open** dialog box where you can add an existing project to the workspace.

Options (Alt+F7)

Displays the **Options** dialog box, where you can set options for the build tools, for the selected item in the **Workspace** window, see *Options dialog box*, page 119. You can set options for the entire project, for a group of files, or for an individual file.

Version Control System

Displays a submenu with commands for version control, see *Version Control System menu for Subversion*, page 105.

Make (F7)

Brings the current build configuration up to date by compiling, assembling, and linking only the files that have changed since the last build.

Compile (Ctrl+F7)

Compiles or assembles the currently selected file, files, or group.

One or more files can be selected in the **Workspace** window—all files in the same project, but not necessarily in the same group. You can also select the editor window containing the file you want to compile. The **Compile** command is only enabled if *all* files in the selection can be compiled or assembled.

You can also select a *group*, in which case the command is applied to each file in the group (also inside nested groups) that can be compiled, even if the group contains files that cannot be compiled, such as header files.

If the selected file is part of a multi-file compilation group, the command will still only affect the selected file.

Rebuild All

Rebuilds and relinks all files in the current target.

Clean

Removes any intermediate files.

Batch Build (F8)

Displays the **Batch Build** dialog box where you can configure named batch build configurations, and build a named batch. See *Batch Build dialog box*, page 122.

C-STAT Static Analysis>Analyze Project

Makes C-STAT analyze the selected project. For more information about C-STAT, see the *C-STAT® Static Analysis Guide*.

C-STAT Static Analysis>Analyze File(s)

Makes C-STAT analyze the selected file(s). For more information about C-STAT, see the *C-STAT® Static Analysis Guide*.

C-STAT Static Analysis>Clear Analysis Results

Makes C-STAT clear the analysis information for previously performed analyses. For more information about C-STAT, see the *C-STAT® Static Analysis Guide*.

C-STAT Static Analysis>Generate HTML Summary

Shows a standard save dialog box where you can select the destination for a report summary in HTML and create it. For more information about C-STAT, see the *C-STAT® Static Analysis Guide*.

C-STAT Static Analysis>Generate Full HTML Report

Shows a standard save dialog box where you can select the destination for a full report in HTML and create it. For more information about C-STAT, see the *C-STAT® Static Analysis Guide*.

Analyze Project

Runs the external analyzer that you select and performs an analysis on all source files of your project. The list of analyzers is populated with analyzers you specify on the **External Analyzers** page in the **IDE Options** dialog box.

Note that this menu command is only available if you have added an external analyzer. For more information, see *Getting started using external analyzers*, page 31.

Analyze File(s)

Runs the external analyzer that you select and performs an analysis on a group of files or on an individual file. The list of analyzers is populated with analyzers you specify on the **External Analyzers** page in the **IDE Options** dialog box.

Note that this menu command is only available if you have added an external analyzer. For more information, see *Getting started using external analyzers*, page 31.

Stop Build (Ctrl+Break)

Stops the current build operation.

Download and Debug (Ctrl+D)

Downloads the application and starts C-SPY so that you can debug the project object file. If necessary, a make will be performed before running C-SPY to ensure the project is up to date. This command is not available during a debug session.

Debug without Downloading

Starts C-SPY so that you can debug the project object file. This menu command is a shortcut for the **Suppress Download** option available on the **Download** page. The **Debug without Downloading** command is not available during a debug session.

Attach to Running Target

Makes the debugger attach to a running application at its current location, without resetting the target system. If you have defined any breakpoints in your project, the C-SPY driver will set them during attachment. If the C-SPY driver cannot set them without stopping the target system, the breakpoints will be disabled. The option also suppresses download and the **Run to** option.

If the option is not available, it is not supported by the combination of C-SPY driver and device you are using.

Make & Restart Debugger

Stops C-SPY, makes the active build configuration, and starts the debugger again; all in a single command. This command is only available during a debug session.

Restart Debugger

Stops C-SPY and starts the debugger again; all in a single command. This command is only available during a debug session.

Download

Commands for flash download and erase. Choose between these commands:

Download active application downloads the active application to the target without launching a full debug session. The result is roughly equivalent to launching a debug session but exiting it again before the execution starts.

Download file opens a standard **Open** dialog box where you can specify a file to be downloaded to the target system without launching a full debug session.

Erase memory.

If your `.board` file specifies only one flash memory, a simple confirmation dialog box is displayed where you confirm the erasure. However, if your `.board` file specifies two or more flash memories, the **Erase Memory** dialog box is displayed. For information about this dialog box, see the *C-SPY® Debugging Guide for RH850*.

SFR Setup

Opens the **SFR Setup** window which displays the currently defined SFRs that C-SPY has information about. For more information about this window, see the *C-SPY® Debugging Guide for RH850*.

Open Device Description File

Opens a submenu where you can choose to open a file from a list of all device files and SFR definitions files that are in use.

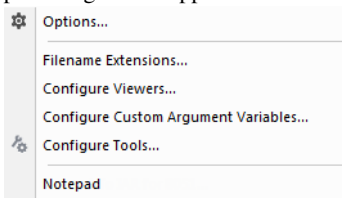
Save List of Registers

Generates a list of all defined registers, including SFRs, with information about the size, location, and access type of each register. If you are in a debug session, the list also includes the current value of the register. This menu command is only available when a project is loaded in the IDE.

Tools menu

The **Tools** menu provides commands for customizing the environment, such as changing common fonts and shortcut keys.

It is a user-configurable menu to which you can add tools for use with IAR Embedded Workbench. Therefore, it might look different depending on which tools you have preconfigured to appear as menu items.



Menu Commands

These commands are available:

Options

Displays the **IDE Options** dialog box where you can customize the IDE. See:

Common Fonts options, page 48

Key Bindings options, page 49

Language options, page 51

Editor options, page 52

Configure Auto Indent dialog box, page 55

External Editor options, page 56

Editor Setup Files options, page 58

Editor Colors and Fonts options, page 59

Messages options, page 60

Project options, page 61

Source Code Control options (deprecated), page 67

Debugger options, page 68

Stack options, page 70

Terminal I/O options, page 72

Filename Extensions

Displays the **Filename Extensions** dialog box where you can define the filename extensions to be accepted by the build tools, see *Filename Extensions dialog box*, page 78.

Configure Viewers

Displays the **Configure Viewers** dialog box where you can configure viewer applications to open documents with, see *Configure Viewers dialog box*, page 76.

Configure Custom Argument Variables

Displays the **Configure Custom Argument Variables** dialog box where you can define and edit your own custom argument variables, see *Configure Custom Argument Variables dialog box*, page 83.

Configure Tools

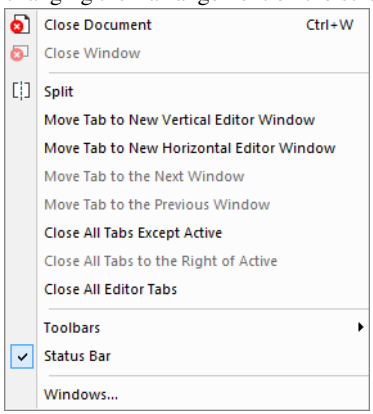
Displays the **Configure Tools** dialog box where you can set up the interface to use external tools, see *Configure Tools dialog box*, page 74.

Notepad

User-configured. This is an example of a user-configured addition to the Tools menu.

Window menu

The **Window** menu provides commands for manipulating the IDE windows and changing their arrangement on the screen.



The last section of the **Window** menu lists the currently open windows. Choose the window you want to switch to.

Menu commands

These commands are available:

Close Document (Ctrl+W)

Closes the active editor document.

Close Window

Closes the active IDE window.

Split

Splits an editor window horizontally or vertically into two or four panes, which means that you can see more parts of a file simultaneously.

Move Tab to New Vertical Editor Window

Opens a new empty window next to the current editor window and moves the active document to the new window.

Move Tab to New Horizontal Editor Window

Opens a new empty window under the current editor window and moves the active document to the new window.

Move Tab to the Next Window

Moves the active document in the current window to the next window.

Move Tab to the Previous Window

Moves the active document in the current window to the previous window.

Close All Tabs Except Active

Closes all the tabs except the current tab.

Close All Tabs to the Right of Active

Closes all tabs to the right of the current tab.

Close All Editor Tabs

Closes all tabs currently available in editor windows.

Toolbars

The options on this submenu toggle the toolbars on or off. There might be toolbars that are only available for certain C-SPY debug drivers, and only during a debug session.

Status bar

Toggles the status bar on or off.

Help menu

The **Help** menu provides help about IAR Embedded Workbench. From this menu you can also find the version numbers of the user interface and of the IDE, see *Product Info dialog box*, page 80.

You can also access the Information Center from the **Help** menu. The Information Center is an integrated navigation system that gives easy access to the information resources you need to get started and during your project development: tutorials, example projects, user guides, support information, and release notes. It also provides shortcuts to useful sections on the IAR Systems web site.

General options

- Description of general options

Description of general options

Reference information about:

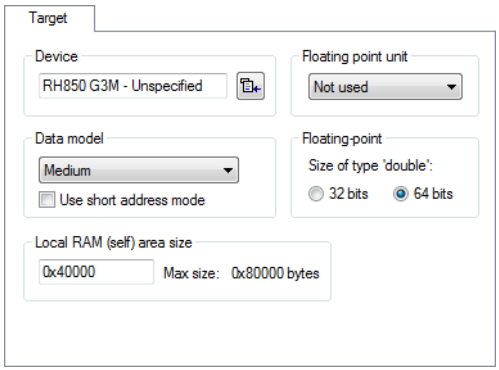
- Target options
- Output
- Library Configuration
- Library Options 1
- Library Options 2
- Stack/Heap
- MISRA C

To set general options in the IDE:

1. Choose **Project>Options** to display the **Options** dialog box.
2. Select **General Options** in the **Category** list.
3. To restore all settings to the default factory settings, click the **Factory Settings** button.

Target options

The **Target** options specify target-specific features for the IAR C/C++ Compiler and Assembler.



Device

Use this option to specify the processor core family and device your are developing for. For a description of the available core variants, see the *IAR C/C++ Development Guide for RH850*.

The choice of device will automatically determine the default linker configuration file and C-SPY® device description file. For information about how to override the default files, see the *C-SPY® Debugging Guide for RH850*.

Data model

Specifies the default memory type for data. Use the Data model option to choose the data model for your project.

Your choice of data model determines the availability of data model options.

Tiny

Places data in the low 32 Kbytes or high 32 Kbytes.

Small

Places data in a 64-Kbytes area anywhere in memory.

Medium

Places data in 8 Mbytes RAM and 8 Mbytes in ROM. Only available for the V850E2M core and above.

Large

Places data in the entire 4 Gbytes of memory.

For more information about the data models, see the *IAR C/C++ Development Guide for RH850.*

Use short address mode

Uses the efficient short addressing mode for your project; to place the data in the saddr area, a 256-byte section of the default memory area. See the *IAR C/C++ Development Guide for RH850* for more information.

Floating-point unit

Enables the floating-point unit. Choose between:

Not used

Does not use the floating-point unit.

Single precision

Uses the floating-point unit for 32-bit operations.

Double precision

Uses the floating-point unit for all operations.

The drop-down list looks slightly different for different drivers. If the selected device does not have a floating-point unit, the setting will be **Not available**.

For more details about the floating-point unit, see the *IAR C/C++ Development Guide for RH850.*

Floating-point

The compiler represents floating-point values by 32- and 64-bit numbers in standard IEEE 754 format. **Size of type 'double'** selects the size of the type `double`; choose between:

32 bits

The data type `double` is represented by the 32-bit floating-point format.

64 bits

The data type `double` is represented by the 64-bit floating-point format.

For more information about the floating-point format, see the *IAR C/C++ Development Guide for RH850.*

Local RAM (self) area size

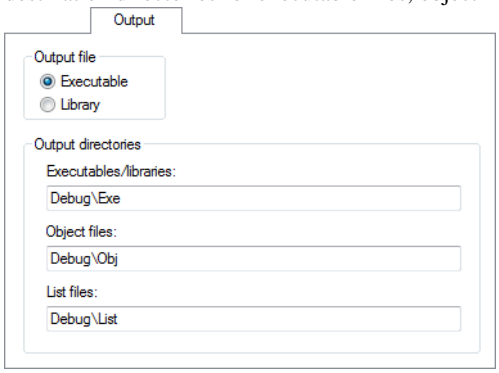
For multicore devices, this is the amount of RAM allocated to Local RAM (self) and the remaining RAM will be allocated to Local RAM (CPU *n*). For single-core devices with two logical areas for the same physical RAM, this is the amount of RAM allocated to Local RAM (mirror) and the remaining RAM will be allocated to Local RAM.

Note: The stack section is located in the Local RAM (self/mirror) area; make sure to allocate enough space for it.

This option overrides the size of the Local RAM (self/mirror) area specified in the linker configuration file. For more details about the Local RAM areas, see the *IAR C/C++ Development Guide for RH850.*

Output

The **Output** options determine the type of output file. You can also specify the destination directories for executable files, object files, and list files.



Output file

Selects the type of the output file:

Executable (default)

As a result of the build process, the linker will create an *application* (an executable output file). When this setting is used, linker options will be available in the **Options** dialog box. Before you create the output you should set the appropriate linker options.

Library

As a result of the build process, the library builder will create a *library file*. When this setting is used, library builder options will be available in the **Options** dialog box, and **Linker** will disappear from the list of categories. Before you create the library you can set the options.

Output directories

Specify the paths to the destination directories. Note that incomplete paths are relative to your project directory. You can specify:

Executables/libraries

Overrides the default directory for executable or library files. Type the name of the directory where you want to save executable files for the project.

Object files

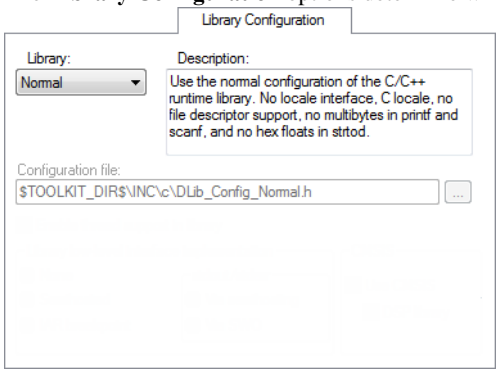
Overrides the default directory for object files. Type the name of the directory where you want to save object files for the project.

List files

Overrides the default directory for list files. Type the name of the directory where you want to save list files for the project.

Library Configuration

The **Library Configuration** options determine which library to use.



For information about the runtime library, library configurations, the runtime environment they provide, and the possible customizations, see *IAR C/C++ Development Guide for RH850*.

Library

Selects which runtime library to use. For information about available libraries, see the *IAR C/C++ Development Guide for RH850*.

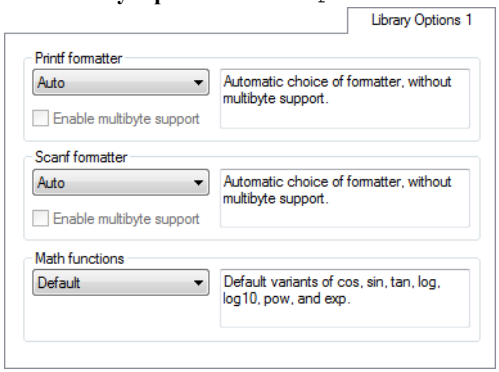
The names of the library object file and library configuration file that actually will be used are displayed in the **Library file** and **Configuration file** text boxes, respectively.

Configuration file

Displays the library configuration file that will be used. A library configuration file is chosen automatically depending on the project settings. If you have chosen **Custom** in the **Library** drop-down list, you must specify your own library configuration file.

Library Options 1

The **Library Options** select the `printf` and `scanf` formatters.



For information about the capabilities of the formatters, see the *IAR C/C++ Development Guide for RH850*.

Printf formatter

If you select **Auto**, the linker automatically chooses the appropriate formatter for `printf`-related functions based on information from the compiler.

To override the default formatter for all `printf`-related functions, except for `wprintf` variants, choose between:

- Printf formatters in the IAR DLIB Library: **Full**, **Large**, **Small**, and **Tiny**

Choose a formatter that suits the requirements of your application.

Select **Enable multibyte support** to make the printf formatter support multibytes.

Scanf formatter

If you select **Auto**, the linker automatically chooses the appropriate formatter for `scanf`-related functions based on information from the compiler.

To override the default formatter for all `scanf`-related functions, except for `wscanf` variants, choose between:

- Scanf formatters in the IAR DLIB Library: **Full**, **Large**, and **Small**

Choose a formatter that suits the requirements of your application.

Select **Enable multibyte support** to make the scanf formatter support multibytes.

Math functions

Some library math functions are also available in size-optimized versions, and in more accurate versions. Choose between:

Default

The default versions of the functions `cos`, `exp`, `log`, `log10`, `pow`, `sin`, `tan`, and `__iar_Sin`.

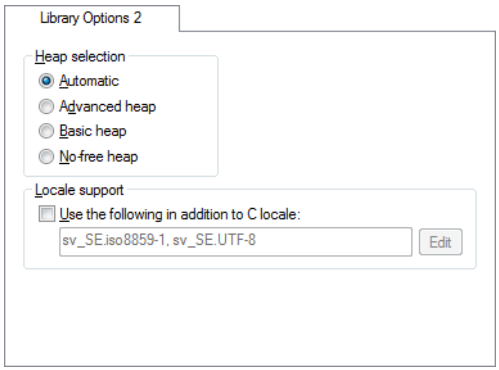
Smaller

Versions of the functions `cos`, `exp`, `log`, `log10`, `pow`, `sin`, `tan`, and `__iar_Sin` that are about 20% smaller and about 20% faster than the default versions.

More accurate

Versions of the functions `cos`, `exp`, `log`, `log10`, `pow`, `sin`, `tan`, and `__iar_Sin` that are more exact and can handle larger argument ranges than the default versions. The drawback is that they are larger and slower than the default versions.

Library Options 2



Heap selection

Select the heap to use. For more information about heaps, see the *IAR C/C++ Development Guide for RH850*.

Automatic

Automatically selects the heap to use for your application.

The no-free heap is selected if your code does not contain any calls to `free` or `realloc`. The advanced heap is selected if there are calls to memory allocation routines in your code. Otherwise, the basic heap is selected.

Advanced heap

Selects the advanced heap.

Basic heap

Selects the basic heap.

No-free heap

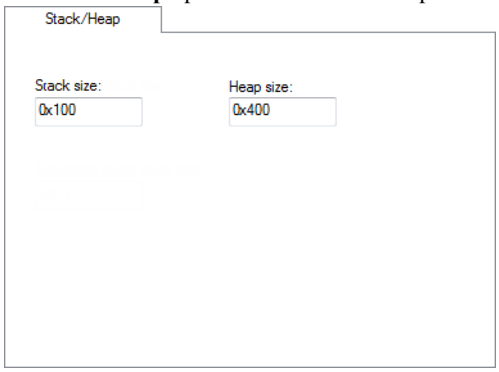
Uses the smallest possible heap implementation. Because this heap does not support `free` or `realloc`, it is only suitable for applications that in the startup phase allocate heap memory for various buffers etc. This heap memory is never deallocated.

Locale support

Select the locales that the linker will use in addition to the C locale. (Requires that you have selected a library configuration that includes the C locale.)

Stack/Heap

The **Stack/Heap** options determine the heap and stack sizes.



For more information about using the stacks and heaps, s*ee the IAR C/C++ Development Guide for RH850*.

Stack size

Specify the stack size.

Heap size

Specify the heap size.

MISRA C

The **MISRA-C:1998** and **MISRA-C:2004** options control how the IDE checks the source code for deviations from the MISRA C rules. The settings are used for both the compiler and the linker.

For details about specific options, see the *IAR Embedded Workbench® MISRA C:2004 Reference Guide* or the *IAR Embedded Workbench® MISRA C:1998 Reference Guide* available from the **Help** menu.

Compiler options

- Description of compiler options

Description of compiler options

Reference information about:

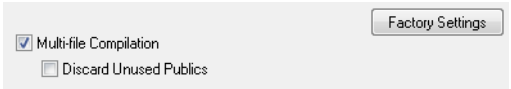
- Multi-file Compilation
- Language 1
- Language 2
- Optimizations
- Output
- List
- Preprocessor
- Diagnostics
- MISRA C
- Encodings
- Extra Options
- Edit Include Directories dialog box

To set compiler options in the IDE:

1. Choose **Project>Options** to display the **Options** dialog box.
2. Select **C/C++ Compiler** in the **Category** list.
3. To restore all settings to the default factory settings, click the **Factory Settings** button.

Multi-file Compilation

Before you set specific compiler options, you can decide whether you want to use multi-file compilation, which is an optimization technique.



Multi-file Compilation

Enables multi-file compilation from the group of project files that you have selected in the **Workspace** window.

You can use this option for the entire project or for individual groups of files. All C/C++ source files in such a group are compiled together using one invocation of the compiler.

This means that all files included in the selected group are compiled using the compiler options which have been set on the group or nearest higher enclosing node which has any options set. Any overriding compiler options on one or more files are ignored when building, because a group compilation must use exactly one set of options.

For information about how multi-file compilation is displayed in the **Workspace** window, see *Workspace window*, page 97.

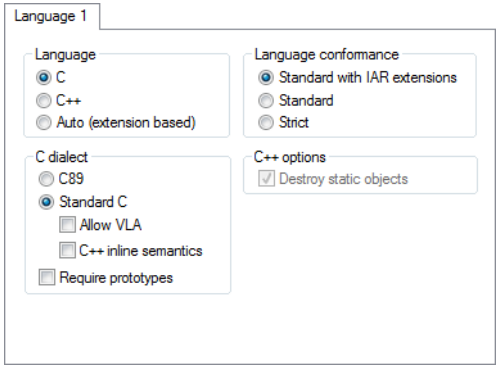
Discard Unused Publics

Discards any unused public functions and variables from the compilation unit.

For more information about multi-file compilation and discarding unused public functions, see the *IAR C/C++ Development Guide for RH850*.

Language 1

The **Language 1** options determine which programming language to use and which extensions to enable.



For more information about the supported languages, their dialects, and their extensions, see the *IAR C/C++ Development Guide for RH850*.

Language

Determines the compiler support for either C or C++:

C (default)

Makes the compiler treat the source code as C, which means that features specific to C++ cannot be used.

C++

Makes the compiler treat the source code as C++.

Auto

Language support is decided automatically depending on the filename extension of the file being compiled:

`c`, files with this filename extension are treated as C source files.

`cpp`, files with this filename extension will be treated as C++ source files.

Language conformance

Controls how strictly the compiler adheres to the standard C or C++ language:

Standard with IAR extensions

Accepts RH850-specific keywords as extensions to the standard C or C++ language. In the IDE, this setting is enabled by default.

Standard

Disables IAR Systems extensions, but does not adhere strictly to the C or C++ dialect you have selected. Some very useful relaxations to C or C++ are still available.

Strict

Adheres strictly to the C or C++ dialect you have selected. This setting disables a great number of useful extensions and relaxations to C or C++.

C dialect

Selects the dialect if **C** is the supported language:

C89

Enables the C89 standard instead of Standard C. Note that this setting is mandatory when the MISRA C checking is enabled.

Standard C

Enables the C18 standard, also known as Standard C. This is the default standard used in the compiler, and it is stricter than C89. Features specific to C89 cannot be used. In addition, choose between:

Allow VLA, allows the use of C11 variable length arrays.

C++ inline semantics, enables C++ inline semantics when compiling a Standard C source code file.

Require prototypes

Forces the compiler to verify that all functions have proper prototypes, which means that source code containing any of the following will generate an error:

- A function call of a function with no declaration, or with a Kernighan & Ritchie C declaration.
- A function definition of a public function with no previous prototype declaration.
- An indirect function call through a function pointer with a type that does not include a prototype.

C++ options

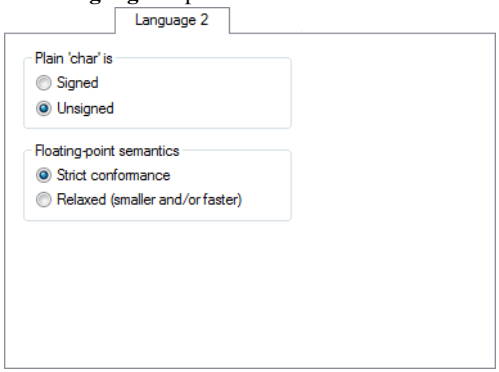
Selects **C++** language options:

Destroy static objects

Makes the compiler generate code to destroy C++ static variables that require destruction at program exit.

Language 2

The **Language 2** options control the use of some language extensions.



Plain 'char' is

Normally, the compiler interprets the plain `char` type as `unsigned char`. **Plain 'char' is Signed** makes the compiler interpret the `char` type as `signed char` instead, for example for compatibility with another compiler.

Note: The runtime library is compiled with unsigned plain characters. If you select the **Signed** option, references to library functionality that uses unsigned plain characters will not work.

Floating-point semantics

Controls floating-point semantics. Choose between:

Strict conformance

Makes the compiler conform strictly to the C and floating-point standards for floating-point expressions.

Relaxed

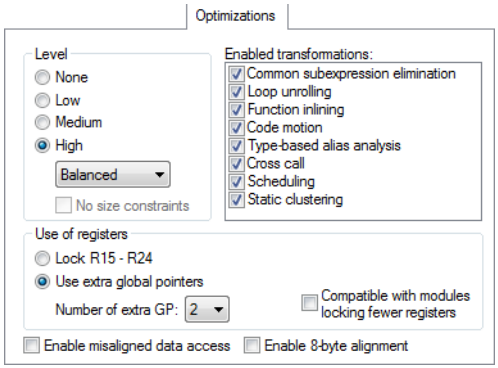
Makes the compiler relax the language rules and perform more aggressive optimization of floating-point expressions. This option improves performance for floating-point expressions that fulfill these conditions:

- The expression consists of both single- and double-precision values
- The double-precision values can be converted to single precision without loss of accuracy
- The result of the expression is converted to single precision.

Note that performing the calculation in single precision instead of double precision might cause a loss of accuracy.

Optimizations

The Optimizations options determine the type and level of optimization for the generation of object code.



Level

Selects the optimization level:

None

No optimization; provides best debug support.

Low

The lowest level of optimization.

Medium

The medium level of optimization.

High, balanced

The highest level of optimization, balancing between speed and size.

High, size

The highest level of optimization, favors size.

High, speed

The highest level of optimization, favors speed.

No size constraints

Optimizes for speed, but relaxes the normal restrictions for code size expansion. This option is only available at the level **High, speed**.

By default, a debug project will have a size optimization that is fully debuggable, while a release project will have a high balanced optimization that generates small code without sacrificing speed.

For a list of optimizations performed at each optimization level, see the *IAR C/C++ Development Guide for RH850*.

Enabled transformations

Selects which transformations that are available at different optimization levels. When a transformation is available, you can enable or disable it by selecting its check box. Choose between:

- Common subexpression elimination
- Loop unrolling
- Function inlining
- Code motion
- Type-based alias analysis
- Static clustering
- Instruction scheduling
- Cross call (subroutine abstraction)

In a debug project the transformations are, by default, disabled. In a release project the transformations are, by default, enabled.

For a brief description of the transformations that can be individually disabled, see the *IAR C/C++ Development Guide for RH850*.

Use of registers

These options determine how the compiler treats registers.

Lock R15 – R24

Locks registers `R15–R24`, so that they are left untouched by the compiler. Note that register `R2` is always free to use by an operating system, because it is never used by the compiler at all.

Use extra global pointers

Reserves some of the registers `R20–R24` for use by the compiler as extra global pointer (GP) registers. Use the **Number of extra GP** option to specify the number of extra global pointer registers. The reserved registers always start with `R20`, so if (for example) three registers are used as extra global pointers, it will be registers `R20–R22`.

Compatible with modules locking fewer registers

Links the module being compiled with object files that lock fewer registers than the module. These object files may also use register constants even if they are not used for this module.

This option does not allow definitions or the use of functions that are not compatible when different register locking levels are used. In practice this means an upper limit to the number of parameters to functions.

In order to use this feature at least one register must be locked from use.

Enable misaligned data access

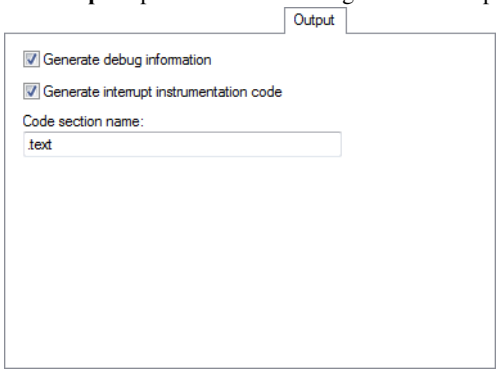
Generates misaligned data accesses for packed structures.

Enable 8-byte alignments

Use this option to enable 8-byte alignment for data types `long long` and `double`.

Output

The **Output** options determine the generated compiler output.



Generate debug information

Makes the compiler include additional information in the object modules that is required by C-SPY® and other symbolic debuggers.

Generate debug information is selected by default. Deselect it if you do not want the compiler to generate debug information.

Note: The included debug information increases the size of the object files.

Generate interrupt instrumentation code

Select this option to generate the instrumentation code needed for the C-SPY Debugger to log interrupts. See the *C-SPY® Debugging Guide for RH850* for information about interrupt logging.

Code section name

The compiler places functions into named sections which are referred to by the IAR ILINK Linker. **Code section name** specifies a different name than the default name to place any part of your application source code into separate non-default sections. This is useful if you want to control placement of your code to different address ranges and you find the `@` notation, alternatively the `#pragma location` directive, insufficient.

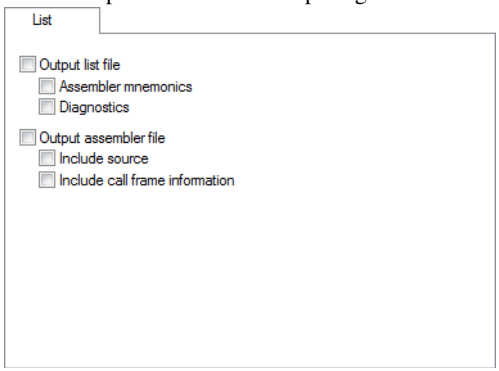
Note: Take care when you explicitly place a function in a predefined section other than the one used by default. This is useful in some situations, but incorrect placement can result in anything from error messages during compilation and linking to a malfunctioning application. Carefully consider the circumstances; there might be strict requirements on the declaration and use of the function or variable.

Note that any changes to the section names require a corresponding modification in the linker configuration file.

For detailed information about sections and the various methods for controlling the placement of code, see the *IAR C/C++ Development Guide for RH850.*

List

The **List** options make the compiler generate a list file and determine its contents.



By default, the compiler does not generate a list file. Select any of the following options to generate a list file or an assembler file. The list file will be saved in the `List` directory, and its filename will consist of the source filename, plus the filename extension `lst`.

If you want to save the list file in another directory than the default directory for list files, use the **Output Directories** option in the **General Options** category, see *Output*, page 202.

You can open the output files directly from the **Output** folder which is available in the **Workspace** window.

Output list file

Makes the compiler generate a list file. You can open the output files directly from the **Output** folder which is available in the **Workspace** window. By default, the compiler does not generate a list file. For the list file content, choose between:

Assembler mnemonics

Includes assembler mnemonics in the list file.

Diagnostics

Includes diagnostic information in the list file.

Output assembler file

Makes the compiler generate an assembler list file. For the list file content, choose between:

Include source

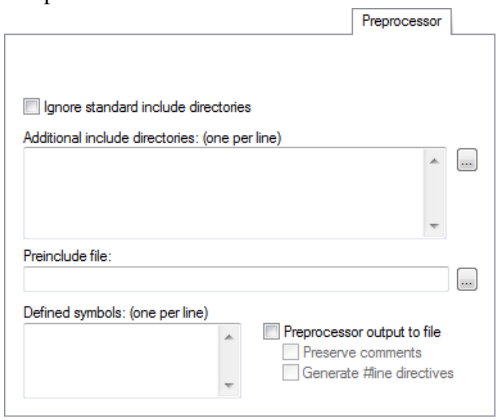
Includes source code in the assembler file.

Include call frame information

Includes compiler-generated information for runtime model attributes, call frame information, and frame size information.

Preprocessor

The **Preprocessor** options allow you to define symbols and include paths for use by the compiler and assembler.



Ignore standard include directories

Normally, the compiler and assembler automatically look for include files in the standard include directories. Use this option to turn off this behavior.

Additional include directories

Specify the full paths of directories to search for include files, one per line. Any directories specified here are searched before the standard include directories, in the order specified.

Use the browse button to display the **Edit Include Directories** dialog box, where you can specify directories using a file browser. For more information, see *Edit Include Directories dialog box*, page 224.

To avoid being dependent on absolute paths, and to make the project more easily portable between different machines and file system locations, you can use argument variables like `$TOOLKIT_DIR$` and `$PROJ_DIR$`, see *Argument variables*, page 81.

Preinclude file

Specify a file to include before the first line of the source file.

Defined symbols

Define a macro symbol (one per line), including its value, for example like this:

```
TESTVER=1
```

This has the same effect as if a line like this appeared before the start of the source file:

```
#define  TESTVER 1
```

A line with no value has the same effect as if `=1` was specified.

Preprocessor output to file

Makes the compiler and assembler output the result of the preprocessing to a file with the filename extension `i`, located in the `lst` directory. Choose between:

Preserve comments

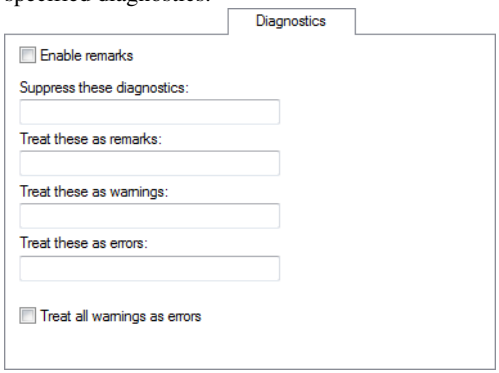
Includes comments in the output. Normally, comments are treated as whitespace, and their contents are not included in the preprocessor output.

Generate #line directives

Generates `#line` directives in the output to indicate where each line originated from.

Diagnostics

The **Diagnostics** options determine how diagnostic messages are classified and displayed. Use the diagnostics options to override the default classification of the specified diagnostics.



Note: The diagnostic messages cannot be suppressed for fatal errors, and fatal errors cannot be reclassified.

Enable remarks

Enables the generation of remarks. By default, remarks are not issued.

The least severe diagnostic messages are called remarks. A remark indicates a source code construct that might cause strange behavior in the generated code.

Suppress these diagnostics

Suppresses the output of diagnostic messages for the tags that you specify.

For example, to suppress the warnings *Xx*`117` and *Xx*`177`, type:

```
Xx117,Xx177
```

Treat these as remarks

Classifies diagnostic messages as remarks. A remark is the least severe type of diagnostic message. It indicates a source code construct that might cause strange behavior in the generated code.

For example, to classify the warning `Xx177` as a remark, type:

```
Xx177
```

Treat these as warnings

Classifies diagnostic messages as warnings. A warning indicates an error or omission that is of concern, but which will not cause the compiler to stop before compilation is completed.

For example, to classify the remark `Xx826` as a warning, type:

```
Xx826
```

Treat these as errors

Classifies diagnostic messages as errors. An error indicates a violation of the language rules, of such severity that object code will not be generated, and the exit code will be non-zero.

For example, to classify the warning `Xx117` as an error, type:

```
Xx117
```

Treat all warnings as errors

Classifies all warnings as errors. If the compiler encounters an error, object code is not generated.

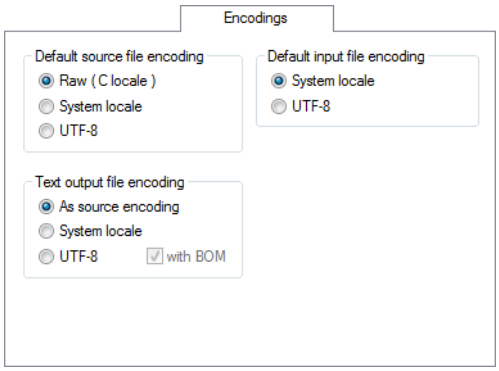
MISRA C

The **MISRA-C:1998** and **MISRA-C:2004** options override the corresponding options in the **General Options** category.

For details about specific options, see the *IAR Embedded Workbench® MISRA C:2004 Reference Guide* or the *IAR Embedded Workbench® MISRA C:1998 Reference Guide* available from the **Help** menu.

Encodings

The **Encodings** options determine the encodings for source files, output files, and input files.



Default source file encoding

Specifies the encoding that the compiler shall use when reading a source file with no Byte Order Mark (BOM).

Raw (C locale)

Sets the Raw encoding (C locale) as the default source file encoding.

System locale

Sets the system locale encoding as the default source file encoding.

UTF-8

Sets the UTF-8 encoding as the default source file encoding

Text output file encoding

Specifies the encoding to be used when generating a text output file.

As source encoding

Uses the same encoding as in the source file.

System locale

Uses the system locale encoding.

UTF-8

Uses the UTF-8 encoding.

UTF-16 little-endian

Uses the UTF-16 little-endian encoding.

UTF-16 big-endian

Uses the UTF-16 big-endian encoding.

With BOM

Adds a Byte Order Mark (BOM) to the output file.

This option is only available when you have selected one of the UTF encodings for your output file.

Default input file encoding

Specifies the encoding that the compiler shall use when reading a text input file with no Byte Order Mark (BOM).

System locale

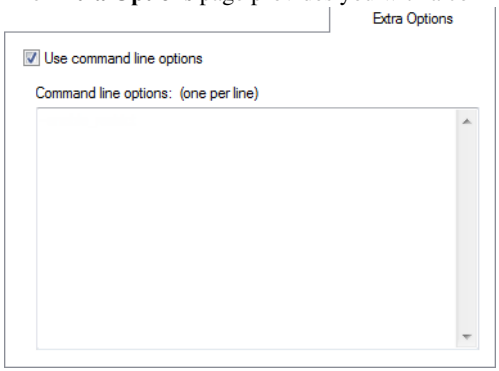
Sets the system locale encoding as the default encoding.

UTF-8

Sets the UTF-8 encoding as the default encoding.

Extra Options

The **Extra Options** page provides you with a command line interface to the tool.

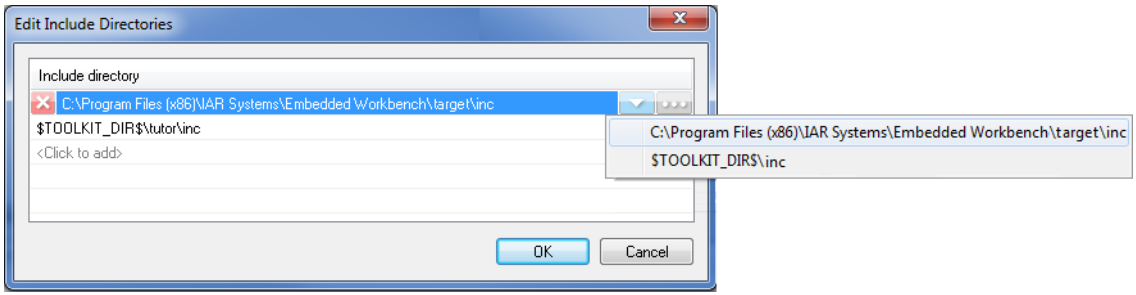


Use command line options

Specify additional command line arguments to be passed to the tool (not supported by the GUI).

Edit Include Directories dialog box

The **Edit Include Directories** dialog box is available from the **Preprocessor** page in the **Options** dialog box for the compiler and assembler categories.



Use this dialog box to specify or delete include paths, or to make a path relative or absolute.

To add a path to an include directory:

1. Click the text **<Click to add>**. A browse dialog box is displayed.
2. Browse to the appropriate include directory and click **Select**. The include path appears. To add yet another one, click **<Click to add>**.

To make the path relative or absolute:

1. Click the drop-down arrow. A context menu is displayed. which shows the absolute path and paths relative to the argument variables `$PROJ_DIR$` and `$TOOLKIT_DIR$`, when possible.
2. Choose one of the alternatives.

To change the order of the paths:

1. Use the shortcut key combinations Ctrl+Up/Down.
2. The list will be sorted accordingly.

To delete an include path:

1. Select the include path and click the red cross at the beginning of the line, alternatively press the **Delete** key.
2. The selected path will disappear.

Assembler options

- Description of assembler options

Description of assembler options

Reference information about:

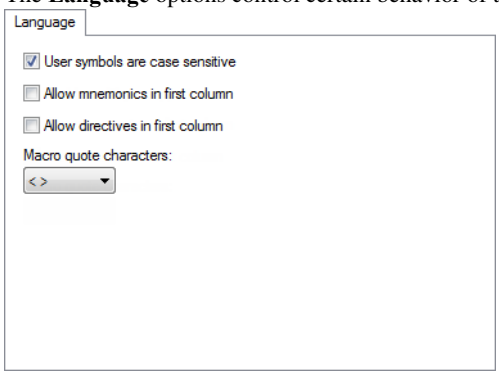
- Language
- Output
- List
- Preprocessor
- Diagnostics
- Extra Options

To set assembler options in the IDE:

1. Choose **Project>Options** to display the **Options** dialog box.
2. Select **Assembler** in the **Category** list.
3. To restore all settings to the default factory settings, click the **Factory Settings** button.

Language

The **Language** options control certain behavior of the assembler language.



User symbols are case sensitive

Toggles case sensitivity on and off. By default, case sensitivity is on. This means that, for example, `LABEL` and `label` refer to different symbols. When case sensitivity is off, `LABEL` and `label` will refer to the same symbol.

Allow mnemonics in first column

Makes mnemonics names (without a trailing colon) that start in the first column to be recognized as mnemonics. By default, the assembler treats all identifiers starting in the first column as labels.

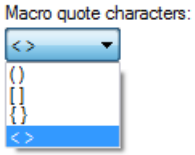
Allow directives in first column

Makes directive names (without a trailing colon) that start in the first column to be recognized as directives. By default, the assembler treats all identifiers starting in the first column as labels.

Macro quote characters

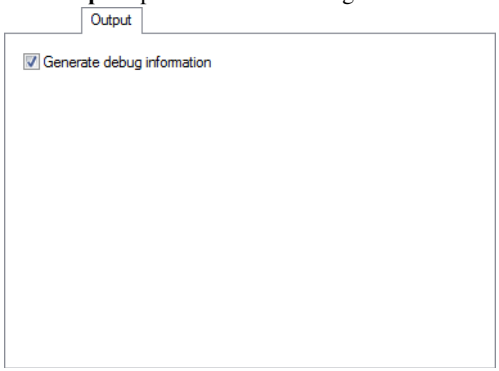
Selects the characters used for the left and right quotes of each macro argument. By default, the characters are < and >.

Macro quote characters changes the quote characters to suit an alternative convention or simply to allow a macro argument to contain < or >.



Output

The **Output** options determine the generated assembler output.

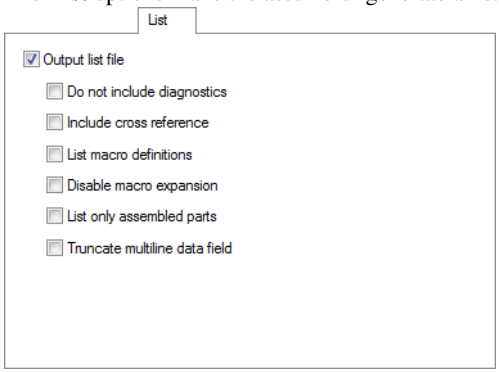


Generate debug information

Makes the assembler generate debug information. Use this option if you want to use a debugger with your application. By default, this option is selected in a Debug project, but not in a Release project.

List

The **List** options make the assembler generate a list file and determine its contents.



Output list file

Makes the assembler generate a list file and send it to the file `sourcename.lst`. By default, the assembler does not generate a list file.

If you want to save the list file in another directory than the default directory for list files, use the **Output Directories** option in the **General Options** category. For more

information, see *Output*, page 202. You can open the output files directly from the **Output** folder which is available in the **Workspace** window.

For the list file content, choose between:

Do not include diagnostics

Excludes diagnostic information from the list file.

Include cross-reference

Generates a cross-reference table at the end of the list file.

List macro definitions

Includes macro definitions in the list file.

Disable macro expansion

Excludes macro expansions from the list file.

List only assembled parts

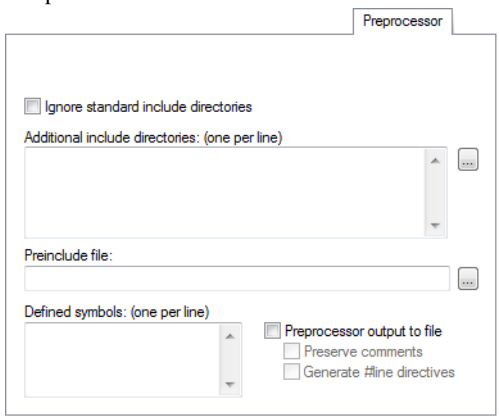
Excludes lines in false conditional assembly sections from the list file.

Truncate multiline data field

Lists only the first line of a generated multiline construction. If the option is deselected, all lines are listed.

Preprocessor

The **Preprocessor** options allow you to define symbols and include paths for use by the compiler and assembler.



Ignore standard include directories

Normally, the compiler and assembler automatically look for include files in the standard include directories. Use this option to turn off this behavior.

Additional include directories

Specify the full paths of directories to search for include files, one per line. Any directories specified here are searched before the standard include directories, in the order specified.

Use the browse button to display the **Edit Include Directories** dialog box, where you can specify directories using a file browser. For more information, see *Edit Include Directories dialog box*, page 224.

To avoid being dependent on absolute paths, and to make the project more easily portable between different machines and file system locations, you can use argument variables like `$TOOLKIT_DIR$` and `$PROJ_DIR$`, see *Argument variables*, page 81.

Preinclude file

Specify a file to include before the first line of the source file.

Defined symbols

Define a macro symbol (one per line), including its value, for example like this:

```
TESTVER=1
```

This has the same effect as if a line like this appeared before the start of the source file:

```
#define  TESTVER 1
```

A line with no value has the same effect as if `=1` was specified.

Preprocessor output to file

Makes the compiler and assembler output the result of the preprocessing to a file with the filename extension `i`, located in the `lst` directory. Choose between:

Preserve comments

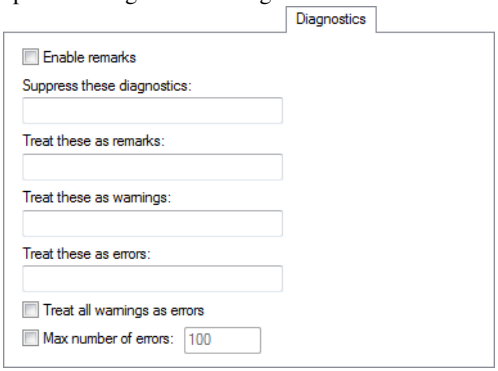
Includes comments in the output. Normally, comments are treated as whitespace, and their contents are not included in the preprocessor output.

Generate #line directives

Generates `#line` directives in the output to indicate where each line originated from.

Diagnostics

The **Diagnostics** options determine how diagnostic messages are classified and displayed. Use the diagnostics options to override the default classification of the specified diagnostic messages.



Note: The diagnostic messages cannot be suppressed for fatal errors, and fatal errors cannot be reclassified.

Enable remarks

Enables the generation of remarks. By default, remarks are not issued.

The least severe diagnostic messages are called remarks. A remark indicates a source code construct that might cause strange behavior in the generated code.

Suppress these diagnostics

Suppresses the output of diagnostic messages for the tags that you specify.

For example, to suppress the warnings *Xx*`117` and *Xx*`177`, type:

```
Xx117,Xx177
```

Treat these as remarks

Classifies diagnostic messages as remarks. A remark is the least severe type of diagnostic message. It indicates a source code construct that might cause strange behavior in the generated code.

For example, to classify the warning `Xx177` as a remark, type:

```
Xx177
```

Treat these as warnings

Classifies diagnostic messages as warnings. A warning indicates an error or omission that is of concern, but which will not cause the assembler to stop before assembly is completed.

For example, to classify the remark `As098` as a warning, type:

```
As098
```

Treat these as errors

Classifies diagnostic messages as errors. An error indicates a violation of the language rules, of such severity that object code will not be generated, and the exit code will be non-zero.

For example, to classify the warning `Xx117` as an error, type:

```
Xx117
```

Treat all warnings as errors

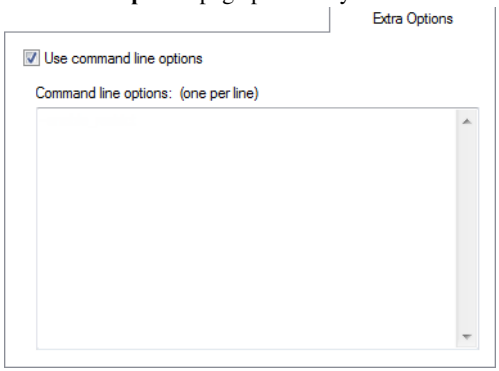
Classifies all warnings as errors. If the assembler encounters an error, object code is not generated.

Max number of errors

Specify the maximum number of errors. This means that you can increase or decrease the number of reported errors, for example, to see more errors in a single assembly. By default, the maximum number of errors reported by the assembler is 100.

Extra Options

The **Extra Options** page provides you with a command line interface to the tool.



Use command line options

Specify additional command line arguments to be passed to the tool (not supported by the GUI).

Output converter options

- Description of output converter options

Description of output converter options

Reference information about:

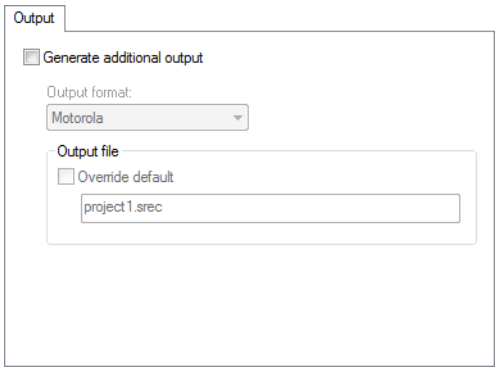
- Output

To set output converter options in the IDE:

1. Choose **Project>Options** to display the **Options** dialog box.
2. Select **Output Converter** in the **Category** list.

Output

The **Output Converter** options determine details about the promable output format.



Generate additional output

The ILINK linker generates ELF as output, optionally including DWARF for debug information. **Generate additional output** makes the converter `ielftool` convert the ELF output to the format you specify, for example Motorola or Intel-extended. For more information about the converter, see the *IAR C/C++ Development Guide for RH850*.

Note: If you change the filename extension for linker output and want to use the output converter `ielftool` to convert the output, make sure `ielftool` will recognize the new filename extension. To achieve this, choose **Tools>Filename Extension**, select your toolchain, and click **Edit**. In the **Filename Extension Overrides** dialog box, select

Output Converter and click **Edit**. In the **Edit Filename Extensions** dialog box, select **Override** and type the new filename extension and click **OK**. `ielftool` will now recognize the new filename extension.

Output format

Selects the format for the output from `ielftool`. Choose between: **Motorola S-records**, **Intel Extended hex**, **Texas Instruments TI-TXT**, **Raw binary**, and **Simple-code**. For more information about the converter, see the *IAR C/C++ Development Guide for RH850*.

Output file

Specifies the name of the `ielftool` converted output file. By default, the linker will use the project name with a filename extension. The filename extension depends on which output format you choose, for example, either `srec` or `hex`. To override the default name, select **Override default** and specify the alternative filename or filename extension.

Custom build options

- Description of custom build options

Description of custom build options

Reference information about:

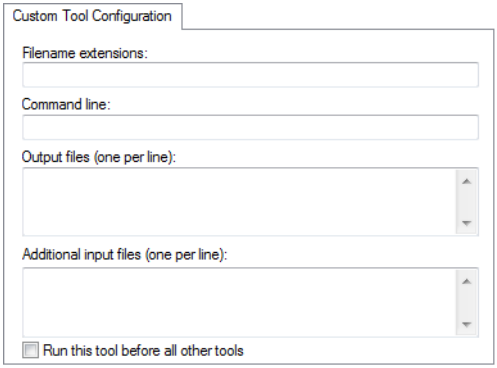
- Custom Tool Configuration

To set custom build options in the IDE:

1. Choose **Project>Options** to display the **Options** dialog box.
2. Select **Custom Build** in the **Category** list.

Custom Tool Configuration

The **Custom Tool Configuration** options control the invocation of the tools you want to add to the tool chain.



For an example, see *Extending the toolchain*, page 109.

Filename extensions

Specify the filename extensions for the types of files that are to be processed by the custom tool. You can type several filename extensions. Use commas, semicolons, or blank spaces as separators. For example:

```
.htm; .html
```

Command line

Specify the command line for executing the external tool.

Output file

Specify the name for the output files from the external tool.

Additional input files

Specify any additional files to be used by the external tool during the build process. If these additional input files, *dependency* files, are modified, the need for a rebuild is detected.

Run this tool before all other tools

Forces the specified custom build tool to be run before all other tools. This can be useful for some tools after a clean command has been executed or when running the tool for the first time, typically to solve errors caused by unknown build dependencies. For example, if the tool produces a header file (`h`), and this option is not used, the source file cannot include the header file before it has been generated.

Build actions options

- Description of build actions options

Description of build actions options

Reference information about:

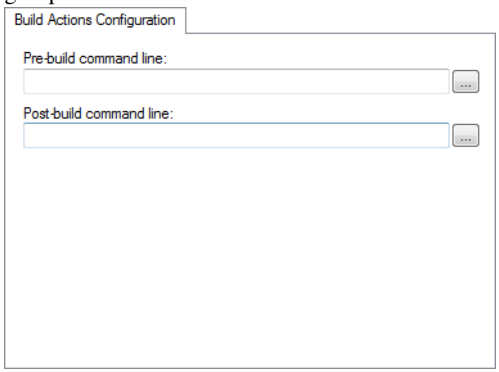
- Build Actions Configuration

To set build action options in the IDE:

1. Choose **Project>Options** to display the **Options** dialog box.
2. Select **Build Actions** in the **Category** list.

Build Actions Configuration

The **Build Actions Configuration** options specify pre-build and post-build actions in the IDE. These options apply to the whole build configuration, and cannot be set on groups or files.



If a pre- or post-build action returns a non-zero error code, the entire **Build** or **Make** command is aborted.

Pre-build command line

Specify the command line to be executed directly before a build. Use the browse button to locate the tool you want to be executed. The commands will not be executed if the configuration is already up-to-date.

Post-build command line

Specify the command line to be executed directly after each successful build. Use the browse button to locate the tool you want to be executed. The commands will not be executed if the configuration was up-to-date. This is useful for copying or post-processing the output file.

Linker options

- Description of linker options

Description of linker options

Reference information about:

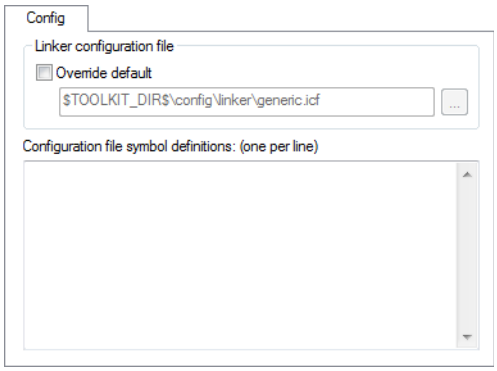
- Config
- Library
- Input
- Optimizations
- Advanced
- Output
- List
- #define
- Diagnostics
- Checksum
- Encodings
- Extra Options
- Edit Additional Libraries dialog box

To set linker options in the IDE:

1. Choose **Project>Options** to display the **Options** dialog box.
2. Select **Linker** in the **Category** list.
3. To restore all settings to the default factory settings, click the **Factory Settings** button.

Config

The **Config** options specify the path and name of the linker configuration file and define symbols for the configuration file.



Linker configuration file

A default linker configuration file is selected automatically based on your project settings. To override the default file, select **Override default** and specify an alternative file.

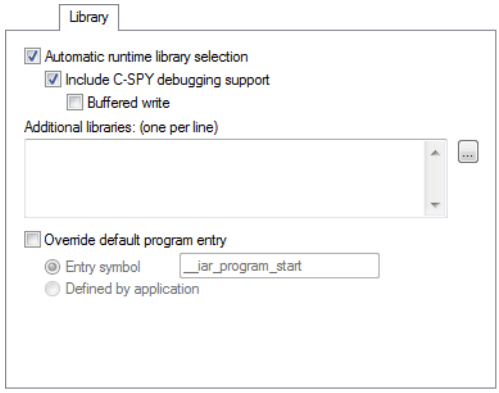
The argument variables `$TOOLKIT_DIR$` or `$PROJ_DIR$` can be used for specifying a project-specific or predefined configuration file.

Configuration file symbol definitions

Define constant configuration symbols to be used in the configuration file. Such a symbol has the same effect as a symbol defined using the `define symbol` directive in the linker configuration file.

Library

The **Library** options select the set of used libraries.



For more information about available libraries, see the *IAR C/C++ Development Guide for RH850*.

Automatic runtime library selection

Makes the linker automatically choose the appropriate library based on your project settings.

Include C-SPY debugging support

Includes a debug library for communication between the application you debug and the debugger itself.

Note: If your code contains calls to I/O functions, performance analysis will not work correctly if this option is selected. See the *C-SPY® Debugging Guide for RH850* for more information.

Buffered write

Buffers terminal output during program execution, instead of instantly printing each new character to the C-SPY **Terminal I/O** window. This option is useful when you use debugger systems that have slow communication.

Additional libraries

Specify additional libraries that you want the linker to include during the link process. You can only specify one library per line and you must specify the full path to the library.

Use the browse button to display the **Edit Additional Libraries** dialog box, where you can specify libraries using a file browser. For more information, see *Edit Additional Libraries dialog box*, page 254.

The argument variables `$PROJ_DIR$` and `$TOOLKIT_DIR$` can be used, see *Argument variables*, page 81.

Alternatively, you can add an additional library directly to your project in the **Workspace** window. You can find an example of this in the tutorial for creating and using libraries.

Override default program entry

By default, the program entry is the label `__iar_program_start`*n* (where *n* is the number of cores that your microcontroller has). The linker makes sure that a module containing the program entry label is included, and that the section containing that label is not discarded.

Override default program entry overrides the default entry label; choose between:

Entry symbol

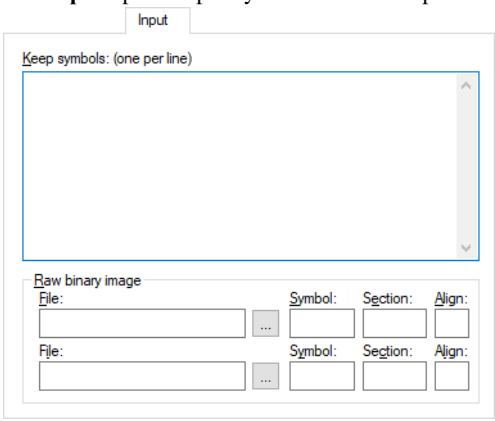
Specify an entry symbol other than default.

Defined by application

Uses an entry symbol defined in the linked object code. The linker will, as always, include all program modules, and enough library modules to satisfy all symbol references, keeping all sections that are marked with the `root` attribute or that are referenced, directly or indirectly, from such a section.

Input

The **Input** options specify how to handle input to the linker.



Keep symbols

Define the symbol, or several symbols one per line, that shall always be included in the final application.

By default, the linker keeps a symbol only if your application needs it.

Raw binary image

Links pure binary files in addition to the ordinary input files. Specify these parameters:

File

The pure binary file you want to link.

Symbol

The symbol defined by the section where the binary data is placed.

Section

The section where the binary data is placed.

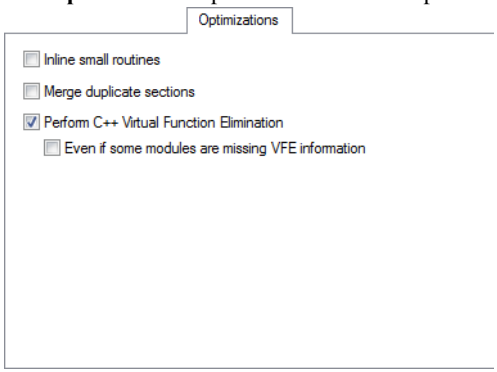
Align

The alignment of the section where the binary data is placed.

The entire contents of the files are placed in the section you specify, which means they can only contain pure binary data, for example, the raw binary output format. The section where the contents of a specified file are placed, is only included if the specified symbol is required by your application. Use **Keep symbols** if you want to force a reference to the symbol. Read more about single output files and the `--keep` option in the *IAR C/C++ Development Guide for RH850*.

Optimizations

The **Optimizations** options control linker optimizations.



For more information about these options, see the *IAR C/C++ Development Guide for RH850.*

Inline small routines

Makes the linker replace the call of a routine with the body of the routine, where applicable.

Merge duplicate sections

Makes the linker keep only one copy of equivalent read-only sections.

Note that this can cause different functions or constants to have the same address, so an application that depends on the addresses being different will not work correctly with this option selected.

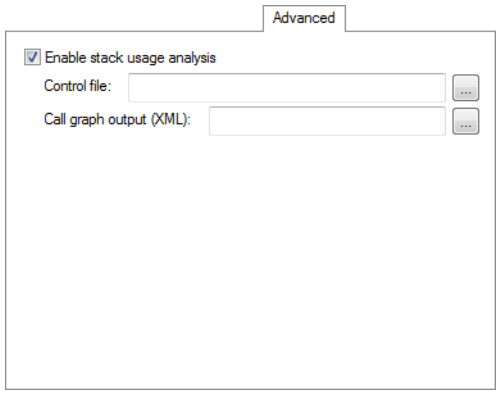
Perform C++ Virtual Function Elimination

Enables the Virtual Function Elimination optimization.

To force the use of Virtual Function Elimination, enable the **Even if some modules are missing VFE information** option. This might be unsafe if some of the modules that lack the needed information perform virtual function calls or use dynamic Runtime Type Information.

Advanced

The **Advanced** options control some miscellaneous linker features.



For more information about these options, see the *IAR C/C++ Development Guide for RH850*.

Enable stack usage analysis

Enables stack usage analysis. If you choose to produce a linker map file, a stack usage chapter is included in the map file. Additionally, you can:

Control file

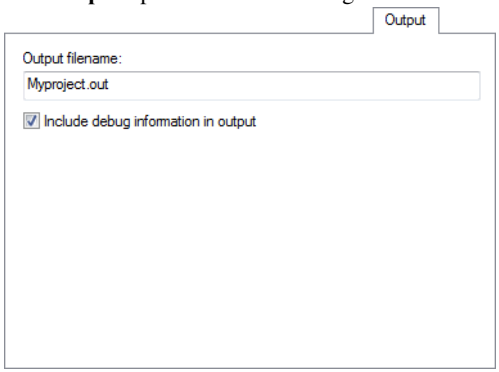
Specify a stack usage control file to use to control stack usage analysis or provide more stack usage information for modules or functions. If no filename extension is specified, the extension `suc` is used.

Call graph output

Specify the name of a call graph file to be generated by the linker. If no filename extension is specified, the extension `cgx` is used.

Output

The **Output** options determine the generated linker output.



Output filename

Sets the name of the ILINK output file. By default, the linker will use the project name with the filename extension `out`. To override the default name, specify an alternative name of the output file.

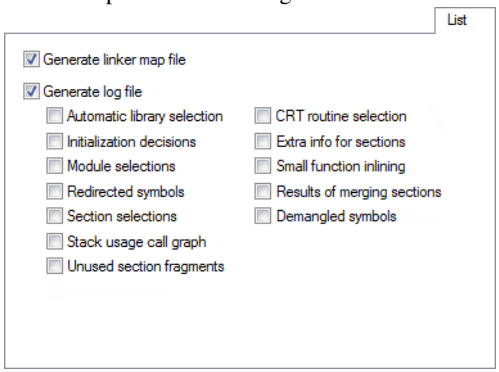
Note: If you change the filename extension for linker output and want to use the output converter `ielftool` to convert the output, make sure `ielftool` will recognize the new filename extension. To achieve this, choose **Tools>Filename Extension**, select your toolchain, and click **Edit**. In the **Filename Extension Overrides** dialog box, select **Output Converter** and click **Edit**. In the **Edit Filename Extensions** dialog box, select **Override** and type the new filename extension and click **OK**. `ielftool` will now recognize the new filename extension.

Include debug information in output

Makes the linker generate an ELF output file including DWARF for debug information.

List

The **List** options control the generation of linker listings.



Generate linker map file

Makes the linker generate a linker memory map file and send it to the *projectname*`.map` file located in the `list` directory. For detailed information about the map file and its contents, see the *IAR C/C++ Development Guide for RH850*.

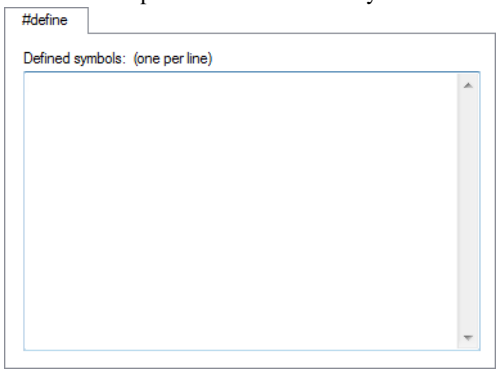
Generate log file

Makes the linker save log information to the *projectname*`.log` file located in the `list` directory. The log information can be useful for understanding why an executable image became the way it is. You can log:

- Automatic library selection
- Initialization decisions
- Module selections
- Redirected symbols
- Section selections
- Stack usage call graph
- Unused section fragments
- CRT routine selection
- Extra info for sections
- Small function inlining
- Results of merging sections
- Demangled symbols

#define

The **#define** options define absolute symbols at link time.



Defined symbols

Define absolute symbols to be used at link time. This is especially useful for configuration purposes. Type the symbols that you want to define for the project, one per line, and specify their value. For example:

```
TESTVER=1
```

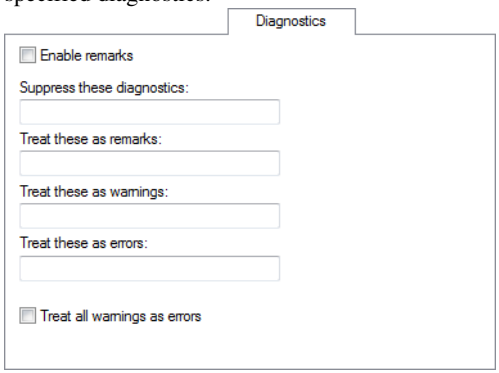
Note that there should be no space around the equals (=) sign.

Any number of symbols can be defined in a linker configuration file. The symbol(s) defined in this manner will be located in a special module called `?ABS_ENTRY_MOD`, which is generated by the linker.

The linker will display an error message if you attempt to redefine an existing symbol.

Diagnostics

The **Diagnostics** options determine how diagnostic messages are classified and displayed. Use the diagnostics options to override the default classification of the specified diagnostics.



Note: The diagnostic messages cannot be suppressed for fatal errors, and fatal errors cannot be reclassified.

Enable remarks

Enables the generation of remarks. By default, remarks are not issued.

The least severe diagnostic messages are called remarks. A remark indicates a source code construct that might cause strange behavior in the generated code.

Suppress these diagnostics

Suppresses the output of diagnostic messages for the tags that you specify.

For example, to suppress the warnings *Xx*`117` and *Xx*`177`, type:

```
Xx117,Xx177
```

Treat these as remarks

Classifies diagnostic messages as remarks. A remark is the least severe type of diagnostic message. It indicates a source code construct that might cause strange behavior in the generated code.

For example, to classify the warning `Xx177` as a remark, type:

```
Xx177
```

Treat these as warnings

Classifies diagnostic messages as warnings. A warning indicates an error or omission that is of concern, but which will not cause the linker to stop before linking is completed.

For example, to classify the remark `Xx826` as a warning, type:

```
Xx826
```

Treat these as errors

Classifies diagnostic messages as errors. An error indicates a violation of the linking rules, of such severity that an executable image will not be generated, and the exit code will be non-zero.

For example, to classify the warning `Xx117` as an error, type:

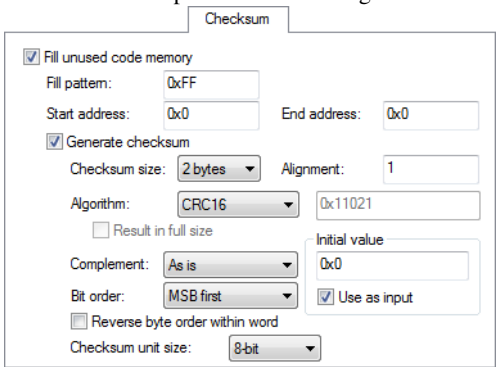
```
Xx117
```

Treat all warnings as errors

Classifies all warnings as errors. If the linker encounters an error, an executable image is not generated.

Checksum

The **Checksum** options control filling and checksumming.



For more information about checksum calculation, see the *IAR C/C++ Development Guide for RH850*.

Fill unused code memory

Fills unused memory in the range you specify:

Fill pattern

Specify a size, in hexadecimal notation, of the filler to be used in gaps between segment parts.

Start address

Specify the start address for the range to be filled.

End address

Specify the end address for the range to be filled.

Generate checksum

Generates a checksum for the specified range.

Choose between:

Checksum size

Selects the size of the checksum, which can be **1**, **2**, or **4** bytes.

Alignment

Specify an optional alignment for the checksum. Typically, this is useful when the processor cannot access unaligned data. If you do not specify an alignment explicitly, an alignment of **1** is used.

Algorithm

Selects the algorithm to be used when calculating the checksum. Choose between:

Arithmetic sum, the simple arithmetic sum algorithm. The result is truncated to one byte.

CRC16 (default), the CRC16 algorithm (generating polynomial 0x1021).

CRC32, the CRC32 algorithm (generating polynomial 0x4C11DB7).

CRC polynomial, the CRC polynomial algorithm, a generating polynomial of the value you specify.

CRC64ISO, the CRC64ISO algorithm (generating polynomial 0x1B).

CRC64ECMA, the CRC64ECMA algorithm (generating polynomial 0x42F0E1EBA9EA3693).

Result in full size

Generates the result of the arithmetic sum algorithm in the size you specify instead of truncating it to one byte.

Complement

Selects the complement variant, either the one's complement or two's complement.

Bit order

Selects the order in which the bits in each byte will be processed. Choose between:

MSB first, which outputs the most significant bit first for each byte.

LSB first, which reverses the bit order for each byte and outputs the least significant bit first.

Reverse byte order within word

Reverses the byte order of the input data within each word of the size specified in **Size**.

Initial value

Specify an initial value for the checksum. This is useful if the microcontroller you are using has its own checksum calculation and you want that calculation to correspond to the calculation performed by the linker.

Use as input

Prefixes the input data with a word of size **Size** that contains the value specified in **Initial value**.

Checksum unit size

Selects the size of the unit for which a checksum should be calculated. Typically, this is useful to make the linker produce the same checksum as some hardware CRC implementations that calculate a checksum for more than 8 bits per iteration. Choose between:

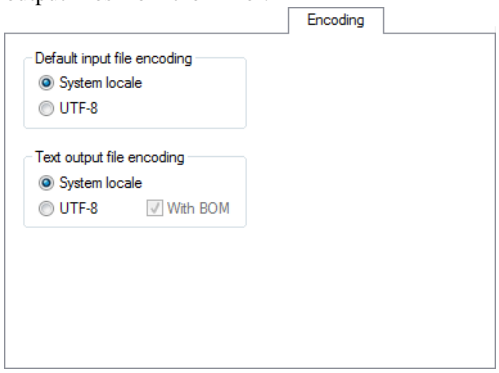
8-bit, calculates a checksum for 8 bits in every iteration.

16-bit, calculates a checksum for 16 bits in every iteration.

32-bit, calculates a checksum for 32 bits in every iteration.

Encodings

The **Encodings** options control the character encodings of the input files to and the output files from the linker.



Default input file encoding

Specifies the default encoding that the linker shall use when reading a text input file with no Byte Order Mark (BOM).

System locale

Sets the system locale as the default encoding.

UTF-8

Sets the UTF-8 encoding as the default.

Text output file encoding

Specifies the encoding that the linker shall use when generating a text output file.

System locale

Uses the system locale encoding.

UTF-8

Uses the UTF-8 encoding.

UTF-16 little-endian

Uses the UTF-16 little-endian encoding.

UTF-16 big-endian

Uses the UTF-16 big-endian encoding.

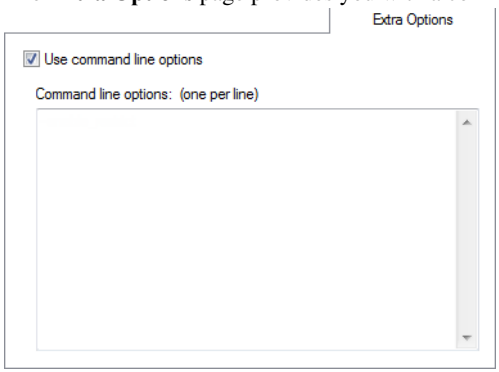
With BOM

Adds a Byte Order Mark to the output file.

This option is only available when you have selected one of the UTF encodings for your output file.

Extra Options

The **Extra Options** page provides you with a command line interface to the tool.

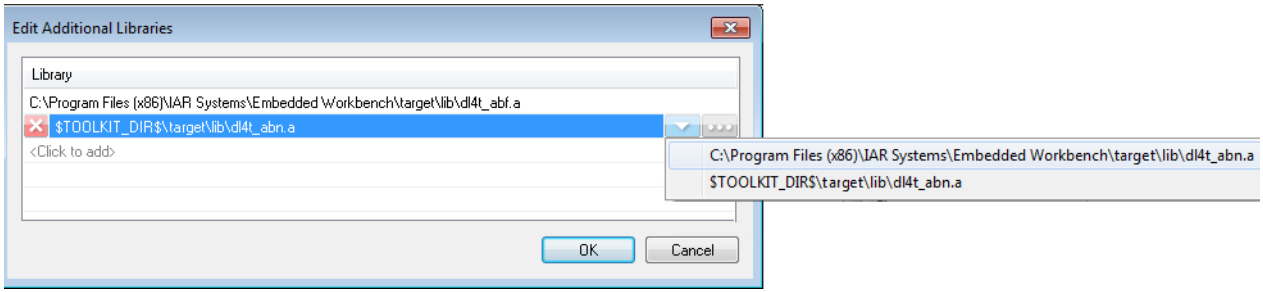


Use command line options

Specify additional command line arguments to be passed to the tool (not supported by the GUI).

Edit Additional Libraries dialog box

The **Edit Additional Libraries** dialog box is available from the **Library** page in the **Options** dialog box.



Use this dialog box to specify additional libraries, or to make a path to a library relative or absolute.

To specify an additional library:

1. Click the text **<Click to add>**. A browse dialog box is displayed.
2. Browse to the appropriate include directory and click **Open**. The library is listed.

   To add yet another one, click **<Click to add>**.

To make the path relative or absolute:

1. Click the drop-down arrow. A context menu is displayed, which shows the absolute path and paths relative to the argument variables `$PROJ_DIR$` and `$TOOLKIT_DIR$`, when possible.
2. Choose one of the alternatives.

To change the order of the libraries:

1. Use the shortcut key combinations Ctrl+Up/Down.
2. Notice that the list will be sorted accordingly.

To delete a library from the list:

1. Select the library and click the red cross at the beginning of the line, alternatively press the **Delete** key.
2. Notice that the selected library will disappear.

Library builder options

- Description of library builder options

Description of library builder options

Reference information about:

- Output

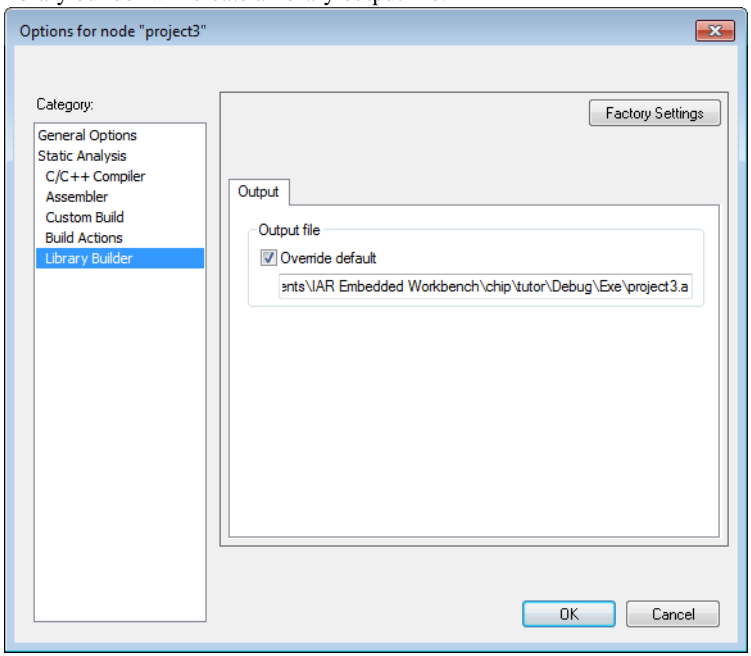
Options for the library builder are not available by default. Before you can set these options in the IDE, you must add the library builder tool to the list of categories.

To set Library Builder options in the IDE:

1. Choose **Project>Options>General Options>Output**.
2. Select the **Library** option, which means that **Library Builder** appears as a category in the **Options** dialog box.
3. Select **Library Builder** in the **Category** list.

Output

The **Output** options control the library builder and as a result of the build process, the library builder will create a library output file.



Output file

Specifies the name of the output file from the library builder. By default, the linker will use the project name with a filename extension. To override the default name, select **Override default** and specify an alternative name of the output file.

Glossary

This is a general glossary for terms relevant to embedded systems programming. Some of the terms do not apply to the IAR Embedded Workbench® version that you are using.

A

Absolute location.
A specific memory address for an object specified in the source code, as opposed to the object being assigned a location by the linker

Address expression
An expression which has an address as its value.

Application
The program developed by the user of the IAR Systems toolkit and which will be run as an embedded application on a target processor.

Ar
The GNU binary utility for creating, modifying, and extracting from archives, that is, libraries. See also *Iarchive*.

Architecture
A term used by computer designers to designate the structure of complex information-processing systems. It includes the kinds of instructions and data used, the memory organization and addressing, and the methods by which the system is implemented. The two main architecture types used in processor design are *Harvard architecture* and *von Neumann architecture*.

Archive
See *Library*.

Assembler directives
The set of commands that control how the assembler operates.

Assembler language
A machine-specific set of mnemonics used to specify operations to the target processor and input or output registers or data areas. Assembler language might sometimes be preferred over C/C++ to save memory or to enhance the execution speed of the application.

Assembler options
Parameters you can specify to change the default behavior of the assembler.

Attributes
See *Section attributes*.

Auto variables
The term refers to the fact that each time the function in which the variable is declared is called, a new instance of the variable is created automatically. This can be compared with the behavior of local variables in systems using static overlay, where a local variable only exists in one instance, even if the function is called recursively. Also called local variables. Compare *Register variables*.

B

Backtrace
Information for keeping call frame information up to date so that the IAR C-SPY® Debugger can return from a function correctly. See also *Call frame information*.

Bank
See *Memory bank*.

Bank switching
Switching between different sets of memory banks. This software technique increases a computer's usable memory by allowing different pieces of memory to occupy the same address space.

Banked code
Code that is distributed over several banks of memory. Each function must reside in only one bank.

Banked data
Data that is distributed over several banks of memory. Each data object must fit inside one memory bank.

Banked memory
Has multiple storage locations for the same address. See also *Memory bank*.

Bank-switching routines
Code that selects a memory bank.

Batch files
A text file containing operating system commands which are executed by the command line interpreter. In Unix, this is called a "shell script" because it is the Unix shell which includes the command line interpreter. Batch files can be used as a simple way to combine existing commands into new commands.

Bitfield
A group of bits considered as a unit.

Block, in linker configuration file
A continuous piece of code or data. It is either built up of blocks, overlays, and sections or it is empty. A block has a name, and the start and end address of the block can be referred to from the application. It can have attributes such as a maximum size, a specific size, or a minimum alignment. The contents can have a specific order or not.

Breakpoint

1 Code breakpoint. A point in a program that, when reached, triggers some special behavior useful to the process of debugging. Generally, breakpoints are used for stopping program execution or dumping the values of some or all of the program variables. Breakpoints can be part of the program itself, or they can be set by the programmer as part of an interactive session with a debugging tool for scrutinizing the program's execution.

2 Data breakpoint. A point in memory that, when accessed, triggers some special behavior useful to the process of debugging. Generally, data breakpoints are used to stop program execution when an address location is accessed either by a read operation or a write operation.

3 Immediate breakpoint. A point in memory that, when accessed, trigger some special behavior useful in the process of debugging. Immediate breakpoints are generally used for halting the program execution in the middle of a memory access instruction (before or after the actual memory access depending on the access type) while performing some user-specified action. The execution is then resumed. This feature is only available in the simulator version of C-SPY.

C

Call frame information
Information that allows the IAR C-SPY® Debugger to show, without any runtime penalty, the complete stack of function calls—*call stack*—wherever the program counter is, provided that the code comes from compiled C functions. See also *Backtrace*.

Calling convention
A calling convention describes the way one function in a program calls another function. This includes how register parameters are handled, how the return value is returned, and which registers that will be preserved by the called function. The compiler handles this automatically for all C and C++ functions. All code written in assembler language must conform to the rules in the calling convention to be callable from C or C++, or to be able to call C and C++ functions. The C calling convention and the C++ calling conventions are not necessarily the same.

Cheap
As in *cheap memory access*. A cheap memory access either requires few cycles to perform, or few bytes of code to implement. A cheap memory access is said to have a low cost. See *Memory access cost*.

Checksum
A small piece of data calculated from a larger block of data for the purpose of detecting errors that might have been introduced during its transmission or storage. Compare *CRC (cyclic redundancy check)*.

Code banking
See *Banked code*.

Code model
The code model controls how code is generated for an application. Typically, the code model controls behavior such as how functions are called and in which code segment functions will be located. All object files of an application must be compiled using the same code model.

Code pointers
A code pointer is a function pointer. As many microcontrollers allow several different methods of calling a function, compilers for embedded systems usually provide the users with the ability to use all these methods.

Do not confuse code pointers with data pointers.

Code segments
Read-only segments that contain code. See also *Section*.

Compilation unit
See *Translation unit*.

Compiler options
Parameters you can specify to change the default behavior of the compiler.

Context menu
A context menu appears when you right-click in the user interface, and provides context-specific menu commands.

Cost
See *Memory access cost*.

CRC (cyclic redundancy check)
A checksum algorithm based on binary polynomials and an initial value. A CRC algorithm is more complex than a simple arithmetic checksum algorithm and has a greater error detecting capability. Most checksum calculation algorithms currently in wide used are based on CRC. Compare *Checksum*.

C-SPY options
Parameters you can specify to change the default behavior of the IAR C-SPY Debugger.

Cstartup
Code that sets up the system before the application starts executing.

C-style preprocessor
A preprocessor is either a stand-alone application or an integrated part of a compiler, that performs preprocessing of the input stream before the actual compilation occurs. A C-style preprocessor follows the rules set up in Standard C and implements commands like `#define`, `#if`, and `#include`, which are used to handle textual macro substitution, conditional compilation, and inclusion of other files.

D

Data banking
See *Banked data*.

Data model
The data model specifies the default memory type. This means that the data model typically controls one or more of the following: The method used and the code generated to access static and global variables, dynamically allocated data, and the runtime stack. It also controls the default pointer type and in which data sections static and global variables will be located. A project can only use one data model at a time, and the same model must be used by all user modules and all library modules in the project.

Data pointers
Many microcontrollers have different addressing modes to access different memory types or address spaces. Compilers for embedded systems usually have a set of different data pointer types so they can access the available memory efficiently.

Data representation
How different data types are laid out in memory and what value ranges they represent.

Declaration
A specification to the compiler that an object, a variable or function, exists. The object itself must be defined in exactly one translation unit (source file). An object must either be

declared or defined before it is used. Normally an object that is used in many files is defined in one source file. A declaration is normally placed in a header file that is included by the files that use the object.

For example:

```
/* Variable "a" exists somewhere. Function
 "b" takes two int parameters and returns an
 int. */

extern int a;
int b(int, int);
```

Definition

The variable or function itself. Only one definition can exist for each variable or function in an application. See also *Tentative definition*.

For example:

```
int a;
int b(int x, int y)
{
   return x + y;
}
```

Demangling

To restore a mangled name to the more common C/C++ name. See also *Mangling*.

Device description file

A file used by C-SPY that contains various device-specific information such as I/O register (SFR) definitions, interrupt vectors, and control register definitions.

Device driver

Software that provides a high-level programming interface to a particular peripheral device.

Digital signal processor (DSP)

A device that is similar to a microprocessor, except that the internal CPU is optimized for use in applications involving discrete-time signal processing. In addition to standard microprocessor instructions, digital signal processors usually support a set of complex instructions to perform common signal-processing computations quickly.

Disassembly window

A C-SPY window that shows the memory contents disassembled as machine instructions, interspersed with the corresponding C source code (if available).

DWARF

An industry-standard debugging format which supports source level debugging. This is the format used by the IAR ILINK Linker for representing debug information in an object.

Dynamic initialization

Variables in a program written in C are initialized during the initial phase of execution, before the main function is called. These variables are always initialized with a static value, which is determined either at compile time or at link time. This is called static initialization. In C++, variables might require initialization to be performed by executing code, for example, running the constructor of global objects, or performing dynamic memory allocation.

Dynamic memory allocation

There are two main strategies for storing variables: statically at link time, or dynamically at runtime. Dynamic memory allocation is often performed from the heap and it is the size of the heap that determines how much memory that can be used for dynamic objects and variables. The advantage of dynamic memory allocation is that several variables or objects that are not active at the same time can be stored in the same memory, thus reducing the memory requirements of an application. See also *Heap memory*.

Dynamic object

An object that is allocated, created, destroyed, and released at runtime. Dynamic objects are almost always stored in memory that is dynamically allocated. Compare *Static object*.

E

EEPROM
Electrically Erasable, Programmable Read-Only Memory. A type of ROM that can be erased electronically, and then be re-programmed.

ELF
Executable and Linking Format, an industry-standard object file format. This is the format used by the IAR ILINK Linker. The debug information is formatted using DWARF.

Embedded C++
A subset of the C++ programming language, which is intended for embedded systems programming. The fact that performance and portability are particularly important in embedded systems development was considered when defining the language.

Embedded system
A combination of hardware and software, designed for a specific purpose. Embedded systems are often part of a larger system or product.

Emulator
An emulator is a hardware device that performs emulation of one or more derivatives of a processor family. An emulator can often be used instead of the actual microcontroller and connects directly to the printed circuit board—where the microcontroller would have been connected—via a connecting device. An emulator always behaves exactly as the processor it emulates, and is used when debugging requires all systems actuators, or when debugging device drivers.

Enea OSE Load module format
A specific ELF format that is loadable by the OSE operating system. See also *ELF*.

Enumeration
A type which includes in its definition an exhaustive list of possible values for variables of that type. Common examples include Boolean, which takes values from the list [true, false], and day-of-week which takes values [Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday].

Enumerated types are a feature of typed languages, including C and Ada.

Characters, (fixed-size) integers, and even floating-point types might be (but are not usually) considered to be (large) enumerated types.

EPROM
Erasable, Programmable Read-Only Memory. A type of ROM that can be erased by exposing it to ultraviolet light, and then be re-programmed.

Executable image
Contains the executable image; the result of linking several relocatable object files and libraries. The file format used for an object file is ELF with embedded DWARF for debug information.

Exceptions
An exception is an interrupt initiated by the processor hardware, or hardware that is tightly coupled with the processor, for instance, a memory management unit (MMU). The exception signals a violation of the rules of the architecture (access to protected memory), or an extreme error condition (division by zero).

Do not confuse this use of the word exception with the term *exception* used in the C++ language (but not in Embedded C++).

Expensive
As in *expensive memory access*. An expensive memory access either requires many cycles to perform, or many bytes of code to implement. An expensive memory access is said to have a high cost. See *Memory access cost.*

Extended keywords
Non-standard keywords in C and C++. These usually control the definition and declaration of objects (that is, data and functions). See also *Keywords*.

F

Filling
How to fill up bytes—with a specific fill pattern—that exists between the sections in an executable image. These bytes exist because of the alignment demands on the sections.

Format specifiers
Used to specify the format of strings sent by library functions such as `printf`. In the following example, the function call contains one format string with one format specifier, `%c`, that prints the value of `a` as a single ASCII character:

```
printf("a = %c", a);
```

G

General options
Parameters you can specify to change the default behavior of all tools that are included in the IDE.

Generic pointers
Pointers that have the ability to point to all different memory types in, for example, a microcontroller based on the Harvard architecture.

H

Harvard architecture
A microcontroller based on the Harvard architecture has separate data and instruction buses. This allows execution to occur in parallel. As an instruction is being fetched, the current instruction is executing on the data bus. Once the current instruction is complete, the next instruction is ready to go. This theoretically allows for much faster execution than a von Neumann architecture, but adds some silicon complexity. Compare *von Neumann architecture*.

Heap memory
The heap is a pool of memory in a system that is reserved for dynamic memory allocation. An application can request parts of the heap for its own use; once memory is allocated from the heap it remains valid until it is explicitly released back to the heap by the application. This type of memory is useful when the number of objects is not known until the application executes. Note that this type of memory is risky to use in systems with a limited amount of memory or systems that are expected to run for a very long time.

Heap size
Total size of memory that can be dynamically allocated.

Host
The computer that communicates with the target processor. The term is used to distinguish the computer on which the debugger is running from the microcontroller the embedded application you develop runs on.

I

Iarchive
The IAR Systems utility for creating archives, that is, libraries. Iarchive is delivered with IAR Embedded Workbench.

IDE (integrated development environment)
A programming environment with all necessary tools integrated into one single application.

IelfdumpRH850
The IAR Systems utility for creating a text representation of the contents of ELF relocatable or executable image.

Ielftool
The IAR Systems utility for performing various transformations on an ELF executable image, such as fill, checksum, and format conversion.

ILINK
The IAR ILINK Linker which produces absolute output in the ELF/DWARF format.

ILINK configuration
The definition of available physical memories and the placement of sections—pieces of code and data—into those memories. ILINK requires a configuration to build an executable image.

Image
See *Executable image*.

Include file
A text file which is included into a source file. This is often done by the preprocessor.

Initialization setup in linker configuration file
Defines how to initialize RAM sections with their initializers. Normally, only non-constant non-noinit variables are initialized but, for example, pieces of code can be initialized as well.

Initialized sections
Read-write sections that should be initialized with specific values at startup. See also *Section*.

Inline assembler
Assembler language code that is inserted directly between C statements.

Inlining
An optimization that replaces function calls with the body of the called function. This optimization increases the execution speed and can even reduce the size of the generated code.

Instruction mnemonics
A word or acronym used in assembler language to represent a machine instruction. Different processors have different instruction sets and therefore use a different set of mnemonics to represent them, such as, `ADD`, `BR` (branch), `BLT` (branch if less than), `MOVE`, `LDR` (load register).

Interrupt vector
A small piece of code that will be executed, or a pointer that points to code that will be executed when an interrupt occurs.

Interrupt vector table
A table containing interrupt vectors, indexed by interrupt type. This table contains the processor's mapping between interrupts and interrupt service routines and must be initialized by the programmer.

Interrupts
In embedded systems, the use of interrupts is a method of detecting external events immediately, for example a timer overflow or the pressing of a button.

Interrupts are asynchronous events that suspend normal processing and temporarily divert the flow of control through an "interrupt handler" routine. Interrupts can be caused by both hardware (I/O, timer, machine check) and software (supervisor, system call or trap instruction). Compare *Trap*.

Intrinsic
An adjective describing native compiler objects, properties, events, and methods.

Intrinsic functions
1. Function calls that are directly expanded into specific sequences of machine code. 2. Functions called by the compiler for internal purposes (that is, floating-point arithmetic etc.).

Iobjmanip
The IAR Systems utility for performing low-level manipulation of ELF object files.

K

Key bindings
Key shortcuts for menu commands used in the IDE.

Keywords
A fixed set of symbols built into the syntax of a programming language. All keywords used in a language are reserved—they cannot be used as identifiers (in other words, user-defined objects such as variables or procedures). See also *Extended keywords*.

L

L-value
A value that can be found on the left side of an assignment and thus be changed. This includes plain variables and

de-referenced pointers. Expressions like `(x + 10)` cannot be assigned a new value and are therefore not L-values.

Language extensions
Target-specific extensions to the C language.

Library
See *Runtime library*.

Library configuration file
A file that contains a configuration of the runtime library. The file contains information about what functionality is part of the runtime environment. The file is used for tailoring a build of a runtime library. See also *Runtime library*.

Linker configuration file
A file that contains a configuration used by the IAR ILINK Linker when building an executable image. See also *ILINK configuration*.

Local variable
See *Auto variables*.

Location counter
See *Program location counter (PLC).*

Logical address
See *Virtual address (logical address).*

M

MAC (Multiply and accumulate)
A special instruction, or on-chip device, that performs a multiplication together with an addition. This is very useful when performing signal processing where many filters and transforms have the form:

$$y_j = \sum_{i=0}^{N} c_i \cdot x_{i+j}$$

The accumulator of the MAC usually has a higher precision (more bits) than normal registers. See also *Digital signal processor (DSP)*.

Macro
1. Assembler macros are user-defined sets of assembler lines that can be expanded later in the source file by referring to the given macro name. Parameters will be substituted if referred to.

2. C macro. A text substitution mechanism used during preprocessing of source files. Macros are defined using the `#define` preprocessing directive. The replacement text of each macro is then substituted for any occurrences of the macro name in the rest of the translation unit.

3. C-SPY macros are programs that you can write to enhance the functionality of C-SPY. A typical application of C-SPY macros is to associate them with breakpoints; when such a breakpoint is hit, the macro is run and can for example be used to simulate peripheral devices, to evaluate complex conditions, or to output a trace.

The C-SPY macro language is like a simple dialect of C, but is less strict with types.

Mailbox
A mailbox in an RTOS is a point of communication between two or more tasks. One task can send messages to another task by placing the message in the mailbox of the other task. Mailboxes are also known as message queues or message ports.

Mangling
Mangling is a technique used for mapping a complex C/C++ name into a simple name. Both mangled and unmangled names can be produced for C/C++ symbols in ILINK messages.

Memory, in linker configuration file
A physical memory. The number of units it contains and how many bits a unit consists of, are defined in the linker configuration file. The memory is always addressable from `0x0` to size -1.

Memory access cost
The cost of a memory access can be in clock cycles, or in the number of bytes of code needed to perform the access. A memory which requires large instructions or many instructions

is said to have a higher access cost than a memory which can be accessed with few, or small instructions.

Memory area
A region of the memory.

Memory bank
The smallest unit of continuous memory in banked memory. One memory bank at a time is visible in a microcontroller's physical address space.

Memory map
A map of the different memory areas available to the microcontroller.

Memory model
Specifies the memory hierarchy and how much memory the system can handle. Your application must use only one memory model at a time, and the same model must be used by all user modules and all library modules.

Microcontroller
A microprocessor on a single integrated circuit intended to operate as an embedded system. In addition to a CPU, a microcontroller typically includes small amounts of RAM, PROM, timers, and I/O ports.

Microprocessor
A CPU contained on one (or a few) integrated circuits. A single-chip microprocessor can include other components such as memory, memory management, caches, floating-point unit, I/O ports and timers. Such devices are also known as microcontrollers.

Module
An object. An object file contains a module and library contains one or more objects. The basic unit of linking. A module contains definitions for symbols (exports) and references to external symbols (imports). When you compile C/C++, each translation unit produces one module.

Multi-file compilation
A technique which means that the compiler compiles several source files as one compilation unit, which enables for interprocedural optimizations such as inlining, cross call, and cross jump on multiple source files in a compilation unit.

N

Nested interrupts
A system where an interrupt can be interrupted by another interrupt is said to have nested interrupts.

Non-banked memory
Has a single storage location for each memory address in a microcontroller's physical address space.

Non-initialized memory
Memory that can contain any value at reset, or in the case of a soft reset, can remember the value it had before the reset.

No-init sections
Read-write sections that should not be initialized at startup. See also *Section*.

Non-volatile storage
Memory devices such as battery-backed RAM, ROM, magnetic tape and magnetic disks that can retain data when electric power is shut off. Compare *Volatile storage*.

NOP
No operation. This is an instruction that does not do anything, but is used to create a delay. In pipelined architectures, the `NOP` instruction can be used for synchronizing the pipeline. See also *Pipeline*.

O

Objcopy
A GNU binary utility for converting an absolute object file in ELF format into an absolute object file, for example the format Motorola-std or Intel-std. See also *Ielftool*.

Object
An object file or a library member.

Object file, absolute
See *Executable image*.

Object file, relocatable
The result of compiling or assembling a source file. The file format used for an object file is ELF with embedded DWARF for debug information.

Operator
A symbol used as a function, with infix syntax if it has two arguments (`+`, for example) or prefix syntax if it has only one (for instance, bitwise negation, `~`). Many languages use operators for built-in functions such as arithmetic and logic.

Operator precedence
Each operator has a precedence number assigned to it that determines the order in which the operator and its operands are evaluated. The highest precedence operators are evaluated first. Use parentheses to group operators and operands to control the order in which the expressions are evaluated.

Options
A set of commands that control the behavior of a tool, for example the compiler or linker. The options can be specified on the command line or via the IDE.

Output image
See *Executable image*.

Overlay, in linker configuration file
Like a block, but it contains several overlaid entities, each built up of blocks, overlays, and sections. The size of an overlay is determined by its largest constituent.

P

Parameter passing
See *Calling convention*.

Peripheral unit
A hardware component other than the processor, for example memory or an I/O device.

Pipeline
A structure that consists of a sequence of stages through which a computation flows. New operations can be initiated at the start of the pipeline even though other operations are already in progress through the pipeline.

Placement, in linker configuration file
How to place blocks, overlays, and sections into a region. It determines how pieces of code and data are actually placed in the available physical memory.

Pointer
An object that contains an address to another object of a specified type.

#pragma
During compilation of a C/C++ program, the #pragma preprocessing directive causes the compiler to behave in an implementation-defined manner. This can include, for example, producing output on the console, changing the declaration of a subsequent object, changing the optimization level, or enabling/disabling language extensions.

Pre-emptive multitasking
An RTOS task is allowed to run until a higher priority process is activated. The higher priority task might become active as the result of an interrupt. The term preemptive indicates that although a task is allotted to run a given length of time (a timeslice), it might lose the processor at any time. Each time an interrupt occurs, the task scheduler looks for the highest priority task that is active and switches to that task. If the located task is different from the task that was executing before the interrupt, the previous task is suspended at the point of interruption.

Compare *Round Robin.*

Preprocessing directives
A set of directives that are executed before the parsing of the actual code is started.

Preprocessor
See *C-style preprocessor*.

Processor variant
The different chip setups that the compiler supports.

Program counter (PC)
A special processor register that is used to address instructions. Compare *Program location counter (PLC)*.

Program location counter (PLC)
Used in the IAR Assembler to denote the code address of the current instruction. The `PLC` is represented by a special symbol (typically `$`) that can be used in arithmetic expressions. Also called simply location counter (`LC`).

Project
The user application development project.

Project options
General options that apply to an entire project, for example the target processor that the application will run on.

PROM
Programmable Read-Only Memory. A type of ROM that can be programmed only once.

Q

Qualifiers
See *Type qualifiers*.

R

Range, in linker configuration file
A range of consecutive addresses in a memory. A region is built up of ranges.

Read-only sections
Sections that contain code or constants. See also *Section*.

Real-time operating system (RTOS)
An operating system which guarantees the latency between an interrupt being triggered and the interrupt handler starting, and how tasks are scheduled. An RTOS is typically much smaller than a normal desktop operating system. Compare *Real-time system*.

Real-time system
A computer system whose processes are time-sensitive. Compare *Real-time operating system (RTOS)*.

Region, in linker configuration file
A set of non-overlapping ranges. The ranges can lie in one or more memories. For ILINK, blocks, overlays, and sections are placed into regions in the linker configuration file..

Region expression, in linker configuration file
A region built up from region literals, regions, and the common set operations possible in the linker configuration file.

Region literal, in linker configuration file
A literal that defines a set of one or more non-overlapping ranges in a memory.

Register
A small on-chip memory unit, usually just one or a few bytes in size, which is particularly efficient to access and therefore often reserved as a temporary storage area during program execution.

Register constant
A register constant is a value that is loaded into a dedicated processor register when the system is initialized. The compiler can then generate code that assumes that the constants are present in the dedicated registers.

Register locking
Register locking means that the compiler can be instructed that some processor registers shall not be used during normal code generation. This is useful in many situations. For example, some parts of a system might be written in assembler language to gain speed. These parts might be given dedicated processor registers. Or the register might be used by an operating system, or by other third-party software.

Register variables
Typically, register variables are local variables that are placed in registers instead of on the (stack) frame of the function. Register variables are much more efficient than other variables because they do not require memory accesses, so the compiler can use shorter/faster instructions when working with them. See also *Auto variables*.

Relay
A synonym to veneer, see *Veneer*.

Relocatable sections
Sections that have no fixed location in memory before linking.

Reset
A reset is a restart from the initial state of a system. A reset can originate from hardware (hard reset), or from software (soft reset). A hard reset can usually not be distinguished from the power-on condition, which a soft reset can be.

ROM-monitor
A piece of embedded software designed specifically for use as a debugging tool. It resides in the ROM of the evaluation board chip and communicates with a debugger via a serial port or network connection. The ROM-monitor provides a set of primitive commands to view and modify memory locations and registers, create and remove breakpoints, and execute your application. The debugger combines these primitives to fulfill higher-level requests like program download and single-step.

Round Robin
Task scheduling in an operating system, where all tasks have the same priority level and are executed in turn, one after the other. Compare *Pre-emptive multitasking*.

RTOS
See *Real-time operating system (RTOS)*.

Runtime library
A collection of relocatable object files that will be included in the executable image only if referred to from an object file, in other words conditionally linked.

Runtime model attributes
A mechanism that is designed to prevent modules that are not compatible to be linked into an application. A runtime attribute is a pair constituted of a named key and its corresponding value.

ILINK uses the runtime model attributes when automatically choosing a library, to verify that the correct one is used.

R-value
A value that can be found on the right side of an assignment. This is just a plain value. See also *L-value*.

S

Saturation arithmetics
Most, if not all, C and C++ implementations use mod–2^N 2-complement-based arithmetics where an overflow wraps the value in the value domain, that is, (127 + 1) = -128. Saturation arithmetics, on the other hand, does *not* allow wrapping in the value domain, for instance, (127 + 1) = 127, if 127 is the upper limit. Saturation arithmetics is often used in signal processing, where an overflow condition would have been fatal if value wrapping had been allowed.

Scheduler
The part of an RTOS that performs task-switching. It is also responsible for selecting which task that should be allowed to run. Many scheduling algorithms exist, but most of them are either based on static scheduling (performed at compile-time), or on dynamic scheduling (where the actual choice of which task to run next is taken at runtime, depending on the state of the system at the time of the task-switch). Most real-time systems use static scheduling, because it makes it possible to prove that the system will not violate the real-time requirements.

Scope
The section of an application where a function or a variable can be referenced by name. The scope of an item can be limited to file, function, or block.

Section
An entity that either contains data or text. Typically, one or more variables, or functions. A section is the smallest linkable unit.

Section attributes
Each section has a name and an attribute. The attribute defines what a section contains, that is, if the section content is read-only, read/write, code, data, etc.

Section fragment
A part of a section, typically a variable or a function.

Section selection
In the linker configuration file, defining a set of sections by using section selectors. A section belongs to the most restrictive section selector if it can be part of more than one selection. Three different selectors can be used individually or in conjunction to select the set of sections: *section attribute* (selecting by the section content), *section name* (selecting by the section name), and *object name* (selecting from a specific object).

Semaphore
A semaphore is a type of flag that is used for guaranteeing exclusive access to resources. The resource can be a hardware port, a configuration memory, or a set of variables. If several tasks must access the same resource, the parts of the code (the critical sections) that access the resource must be made exclusive for every task. This is done by obtaining the semaphore that protects that resource, thus blocking all other tasks from it. If another task wishes to use the resource, it also must obtain the semaphore. If the semaphore is already in use, the second task must wait until the semaphore is released. After the semaphore is released, the second task is allowed to execute and can obtain the semaphore for its own exclusive access.

Severity level
The level of seriousness of the diagnostic response from the assembler, compiler, or debugger, when it notices that something is wrong. Typical severity levels are remarks, warnings, errors, and fatal errors. A remark just points to a possible problem, while a fatal error means that the programming tool exits without finishing.

Sharing
A physical memory that can be addressed in several ways. For ILINK, defined in the linker configuration file.

Short addressing
Many microcontrollers have special addressing modes for efficient access to internal RAM and memory mapped I/O. Short addressing is therefore provided as an extended feature by many compilers for embedded systems. See also *Data pointers*.

Side effect
An expression in C or C++ is said to have a side-effect if it changes the state of the system. Examples are assignments to a variable, or using a variable with the post-increment operator. The C and C++ standards state that a variable that is subject to a side-effect should not be used more that once in an expression. As an example, this statement violates that rule:

```
*d++ = *d;
```

Signal
Signals provide event-based communication between tasks. A task can wait for one or more signals from other tasks. Once a task receives a signal it waits for, execution continues. A task in an RTOS that waits for a signal does not use any processing time, which allows other tasks to execute.

Simulator
A debugging tool that runs on the host and behaves as similar to the target processor as possible. A simulator is used for debugging the application when the hardware is unavailable, or not needed for proper debugging. A simulator is usually not connected to any physical peripheral devices. A simulated processor is often slower, or even much slower, than the real hardware.

Single stepping
Executing one instruction or one C statement at a time in the debugger.

Skeleton code
An incomplete code framework that allows the user to specialize the code.

Special function register (SFR)
A register that is used to read and write to the hardware components of the microcontroller.

Stack frames
Data structures containing data objects like preserved registers, local variables, and other data objects that must be stored temporary for a particular scope (usually a function).

Earlier compilers usually had a fixed size and layout on a stack frame throughout a complete function, while modern compilers might have a very dynamic layout and size that can change anywhere and anytime in a function.

Stack sections
The section or sections that reserve space for the stack(s). Most processors use the same stack for calls and parameters, but some have separate stacks.

Standard libraries
The C and C++ library functions as specified by the C and C++ standard, and support routines for the compiler, like floating-point routines.

Static object
An object whose memory is allocated at link-time and is created during system startup (or at first use). Compare *Dynamic object*.

Static overlay
Instead of using a dynamic allocation scheme for parameters and auto variables, the linker allocates space for parameters and auto variables at link time. This generates a worst-case scenario of stack usage, but might be preferable for small chips with expensive stack access or no stack access at all.

Statically allocated memory
This kind of memory is allocated once and for all at link-time, and remains valid all through the execution of the application. Variables that are either global or declared `static` are allocated this way.

Structure value
A collecting names for structs and unions. A struct is a collection of data object placed sequentially in memory (possibly with pad bytes between them). A union is a collection of data sharing the same memory location.

Symbolic location
A location that uses a symbolic name because the exact address is unknown.

T

Target
1. An architecture. 2. A piece of hardware. The particular embedded system you are developing the application for. The term is usually used to distinguish the system from the host system.

Task (thread)
A task is an execution thread in a system. Systems that contain many tasks that execute in parallel are called multitasking systems. Because a processor only executes one instruction stream at the time, most systems implement some sort of task-switch mechanism (often called context switch) so that all tasks get their share of processing time. The process of determining which task that should be allowed to run next is called scheduling. Two common scheduling methods are *Pre-emptive multitasking* and *Round Robin*.

Tentative definition
A variable that can be defined in multiple files, provided that the definition is identical and that it is an absolute variable.

Terminal I/O
A simulated terminal window in C-SPY.

Timer
A peripheral that counts independent of the program execution.

Timeslice
The (longest) time an RTOS allows a task to run without running the task-scheduling algorithm. A task might be allowed to execute during several consecutive timeslices before being switched out. A task might also not be allowed to use its entire time slice, for example if, in a preemptive system, a higher priority task is activated by an interrupt.

Translation unit
A source file together with all the header files and source files included via the preprocessor directive `#include`, except for the lines skipped by conditional preprocessor directives such as `#if` and `#ifdef`.

Trap
A trap is an interrupt initiated by inserting a special instruction into the instruction stream. Many systems use traps to call operating system functions. Another name for trap is software interrupt.

Type qualifiers
In Standard C/C++, `const` or `volatile`. IAR Systems compilers usually add target-specific type qualifiers for memory and other type attributes.

U

UBROF (Universal Binary Relocatable Object Format)
File format produced by some of the IAR Systems programming tools, if your product package includes the XLINK linker.

V

Value expressions, in linker configuration file
A constant number that can be built up out of expressions that has a syntax similar to C expressions.

Veneer
A small piece of code that is inserted as a springboard between caller and callee when the call instruction does not reach its destination.

Virtual address (logical address)
An address that must be translated by the compiler, linker or the runtime system into a physical memory address before it is used. The virtual address is the address seen by the application, which can be different from the address seen by other parts of the system.

Virtual space
An IAR Embedded Workbench Editor feature which allows you to place the insertion point outside of the area where there are actual characters.

Volatile storage
Data stored in a volatile storage device is not retained when the power to the device is turned off. To preserve data during a power-down cycle, you should store it in non-volatile storage. This should not be confused with the C keyword `volatile`. Compare *Non-volatile storage*.

von Neumann architecture
A computer architecture where both instructions and data are transferred over a common data channel. Compare *Harvard architecture*.

W

Watchpoints
Watchpoints keep track of the values of C variables or expressions in the C-SPY **Watch** window as the application is being executed.

X

XAR
An IAR tool that creates archives (libraries) in the UBROF format. XAR is delivered with IAR Embedded Workbench.

XLIB
An IAR tool that creates archives (libraries) in the UBROF format, listing object code, converting and absolute object file into an absolute object file in another format. XLIB is delivered with IAR Embedded Workbench.

XLINK
The IAR XLINK Linker which uses the UBROF output format.

Z

Zero-initialized sections
Sections that should be initialized to zero at startup. See also *Section*.

Zero-overhead loop
A loop in which the loop condition, including branching back to the beginning of the loop, does not take any time at all. This is usually implemented as a special hardware feature of the processor and is not available in all architectures.

Zone
Different processors have widely differing memory architectures. *Zone* is the term C-SPY uses for a named memory area. For example, on processors with separately addressable code and data memory there would be at least two zones. A processor with an intricate banked memory scheme might have several zones.

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Symbols

Numerics