IAR Embedded Workbench®

Migration Guide

for Renesas
V850 Microcontroller Family

MV850-2

EDITION NOTICE

Second edition: October 2010

Part number: MV850-2

This guide applies to version 3.x of IAR Embedded Workbench® for V850.

Internal reference: R11, Too6.1, IJOA.

Contents

Migrating from version 2.x to version 3.x

This guide gives hints for porting your application code and projects to the new version 3.x.

C or C++ source code that was originally written for the IAR C/EC++ Compiler for V850 version 2.x can be used also with the new IAR C/C++ Compiler for V850 version 3.x. However, some small modifications may be required.

This guide presents the major differences between IAR Embedded Workbench for V850 version 2.x and version 3.x, and describes the migration considerations. Note that there might also be additional migration information in the \doc directory on the installation media.

Key advantages

This section lists the major advantages in the IAR Embedded Workbench for V850 version 3.x as compared to version 2.x. Hereafter, the two versions are referred to as *version 3.x* and *version 2.x*, respectively.

- Efficient window management through dockable windows optionally organized in tab groups
- Source browser with a catalog of functions, classes, and variables, for a quick navigation to symbol definitions
- Template projects to get a project that links and runs *out of the box* for a smooth development start-up
- Batch build with ordered lists of configurations to build
- Improved context-sensitive help for C/C++ library functions
- Generic flash downloader framework
- Easy configuration of the C/C++ libraries
- Smart display of STL containers at debugging
- Auto-display debugger watch window
- Project workspace environment

- Implementation of MISRA C rules checking.

Migration considerations

To migrate your old project consider the following:

- IAR Embedded Workbench IDE
- Project options
- Runtime library and object files considerations.

Note: It is important to be aware of the fact that object code produced with version 2.x cannot be linked with code produced with version 3.x.

IAR Embedded Workbench IDE

Upgrading to the new version of the IAR Embedded Workbench IDE should be a smooth process as the improvements do not have any major influence on the compatibility between the versions.

Version 3.x provides new improved project management with support for complex project setup. The former *project window*—which could manage one separate project—has been exchanged with a *workspace window*. This workspace window can manage several projects and multiple build configurations for each project. For more information about project management in version 3.x, see the *IDE Project Management and Building Guide*.

Upgrading to the new version of the IDE should still be a smooth process, but you should consider the following:

- Project file and project setup
- Project options
- C-SPY layout files.

PROJECT FILE AND PROJECT SETUP

If you are using the IDE, follow these steps to verify that your project file has been properly converted.

1 Start your new version of IAR Embedded Workbench for V850 and create a new workspace by choosing **File>New** and then **Workspace**.

2 Choose **Project>Add Existing Project** to insert your old project into the workspace. This step will create two new project files with the same name as the old file, but with the extensions `ewp` and `ewd`. The `ewp` file contains all settings required to build the application, while the `ewd` file contains all settings related to the debugger. The old project file will remain untouched.

3 It is recommended that you verify that your options have been setup correctly.

To generate a text file with the command line equivalents of the project options in your old project, see *Migrating project options*.

Also, set any new options.

MIGRATING PROJECT OPTIONS

Since the available compiler options differ between the versions, you should verify your option settings after you have converted an old project.

If you are using the command line interface, you can simply compare your makefile with the option tables in this section, and modify the makefile accordingly.

If you are using the IDE, all option settings are automatically converted during the project conversion.

However, it is still recommended to verify the options manually. Follow these steps:

1 Open the old project in the old IAR Embedded Workbench version.

2 In the project window, select the project level to get information about options on all levels in your project.

3 To save the project settings to a file, right-click in the project window. On the context menu that appears, choose **Save As Text**, and save the settings to an appropriate destination.

4 Use this file and the option tables in this section to verify whether the options you used in your old project are still available or needed. Also check whether you need to use any of the new options.

For information about where to set the equivalent options in on the command line, see the *IAR C/C++ Compiler Reference Guide for V850*.

C-SPY LAYOUT FILES

Due to a new improved window management system, the C-SPY layout files support in 2.x has been removed. Any custom made `lew` files can safely be removed from your projects.

Project options

A new compiler optimization, **Type-based alias analysis**, is enabled by default. This optimization can be disabled with the option `--no_tbaa` or by deselecting the corresponding option in the IDE.

Runtime library and object files considerations

IAR Embedded Workbench for V850 version 3.x is delivered with one runtime library, DLIB.

COMPILING AND LINKING WITH THE DLIB RUNTIME LIBRARY

In earlier versions, the choice of runtime library did not have any impact on the compilation. In IAR Embedded Workbench for V850 version 3.x, this has changed. Now you can configure the runtime library to contain the features that are needed by your application.

One example is input and output. An application may use the `fprintf` function for terminal I/O (`stdout`), but the application does not use file I/O functionality on file descriptors associated with the files. In this case the library can be configured so that code related to file I/O is removed but still provides terminal I/O functionality.

This configuration involves the library header files, for example `stdio.h`. This means that when you build your application, the same header file setup must be used as when the library was built. The library setup is specified in a *library configuration file*, which is a header file that defines the library functionality.

When building an application using the IDE, there are three library configuration alternatives to choose between: **Normal**, **Full**, and **Custom**. **Normal** and **Full** are prebuilt library configurations delivered with the product, where **Normal** should be used in the above example with file I/O. **Custom** is used for custom built libraries. Note that the choice of the library configuration file is handled automatically.

When building an application from the command line, you must use the same library configuration file as when the library was built. For the prebuilt libraries (`r85`) there is a corresponding library configuration file (`h`), which has the same name as the library. The files are located in the `v850\lib` directory. The command lines for specifying the library configuration file and library object file could look like this:

```
iccv850 -D_DLIB_CONFIG_FILE=...\v850\lib dl85-tnn.h
xlink dl85-tnn.r85
```

In case you intend to build your own library version, use the default library configuration file `dlv850Custom.h`.

To take advantage of the features it is recommended that you read about the runtime environment in the *IAR C/C++ Compiler Reference Guide for V850*.

PROGRAM ENTRY

By default, the linker includes all `root` declared segment parts in program modules when building an application. However, there is a new mechanism that affects the load procedure.

There is a new linker option **Entry label** (`-s`) to specify a *start label*. By specifying the start label, the linker will look in all modules for a matching start label, and start loading from that point. Like before, any program modules containing a root segment part will also be loaded.

In version 3.x, the default program entry label in `cstartup.s85` is `__program_start`, which means the linker will start loading from there. The advantage of this new behavior is that it is much easier to override `cstartup.s85`.

If you build your application in the IAR Embedded Workbench, just add your customized `cstartup` file to your project. It will then be used instead of the `cstartup` module in the library. It is also possible to switch startup files just by overriding the name of the program entry point.

If you build your application from the command line, the `-s` option must be explicitly specified when linking a C/C++ application. If you link without the option, the resulting output executable will be empty because no modules will be referred to.