Freescale Semiconductor
Application Note

Document Number: AN4104

Converting ColdFire Projects to CodeWarrior Development Studio for Microcontrollers V10.0

Contents

1 Introduction

This application note explains how to convert a ColdFire project created in CodeWarrior Development Studio for Microcontrollers V6.2 or CodeWarrior Development Studio for ColdFire Architectures V7.1 to CodeWarrior Development Studio for Microcontrollers V10.0.

2 Terms and Abbreviations

This application note uses following terms and abbreviations:

- MSL — Main Standard Libraries
- EWL — Embedded Warrior Libraries
- MCU 6.2 — Refers to CodeWarrior Development Studio for Microcontrollers, Version 6.2.
- CW 7.1 — Refers to CodeWarrior Development Studio for ColdFire Architectures, Version 7.1.
- MCU 10.0 — Refers to CodeWarrior Development Studio for Microcontrollers, Version 10.0

3 Embedded Warrior Libraries (EWL)

Embedded Warrior Libraries (EWL) model for the libraries reduces the memory footprint taken by the IO operations and simplifies the memory allocation. The IO operations are divided in three categories:

- printing,
- scanning, and
- file operations.

The printing and scanning formatters for EWL are grouped in an effort to provide only the support required for the application:

- `int` — integer and string processing
- `int_FP` — integer, string, and floating point
- `int_LL` — integer (including long long) and string
- `int_FP_LL` — all but wide chars

You can replace buffered IO with raw IO to bypass the buffering and write directly to the device. However, this works only when `printf` and `scanf` are used to perform IO.

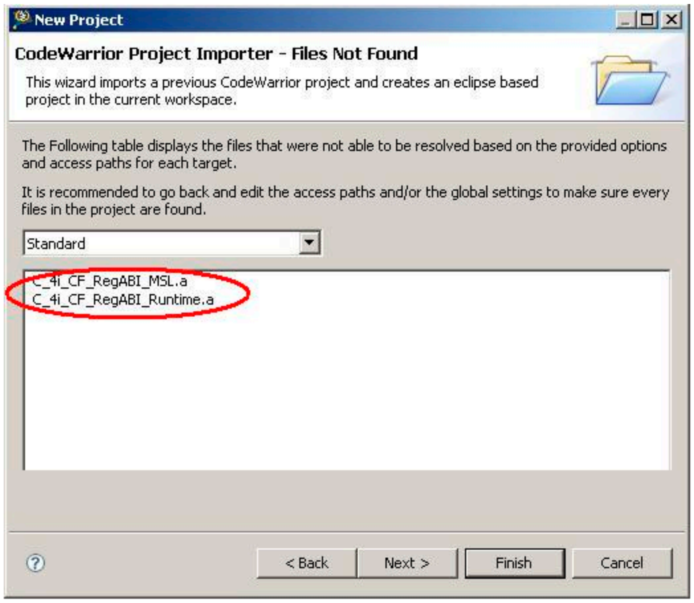
EWL libraries contain prebuilt versions for all formatters and IO modes. Selecting a model combination enables correct compiling and linking. The EWL layout for ColdFire is built per core architecture. The EWL layout for the ColdFire projects consists of:

- `libm.a` — math support (c9x or not)
- `libc.a` — non c9x std C libs
- `libc99.a` — c9x libs
- `librt.a` — runtime libraries
- `libc++.a` — non-c9x matching c++ libraries
- `libstdc++.a` — c9x/c++ compliant libs
- `fp_coldfire.a` — FPU emulation libraries

If you have selected an EWL model for the libraries, you do not need to add libraries to the project. The linker determines the correct library set from the settings, such as processor, pid/pic, hard/soft FPU. Although the library names are known to the toolset their location is not. For information about how to select a library model, refer Section 5, “Librarian.

NOTE The importer generates an error because the old libraries could not be found, as shown in .You should ignore the error as these libraries are replaced by the EWL scheme.

Figure 1. CodeWarrior Project Importer Error — MSL Library Files could not be Found



New Project

CodeWarrior Project Importer - Files Not Found

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

The Following table displays the files that were not able to be resolved based on the provided options and access paths for each target.

It is recommended to go back and edit the access paths and/or the global settings to make sure every files in the project are found.

Standard

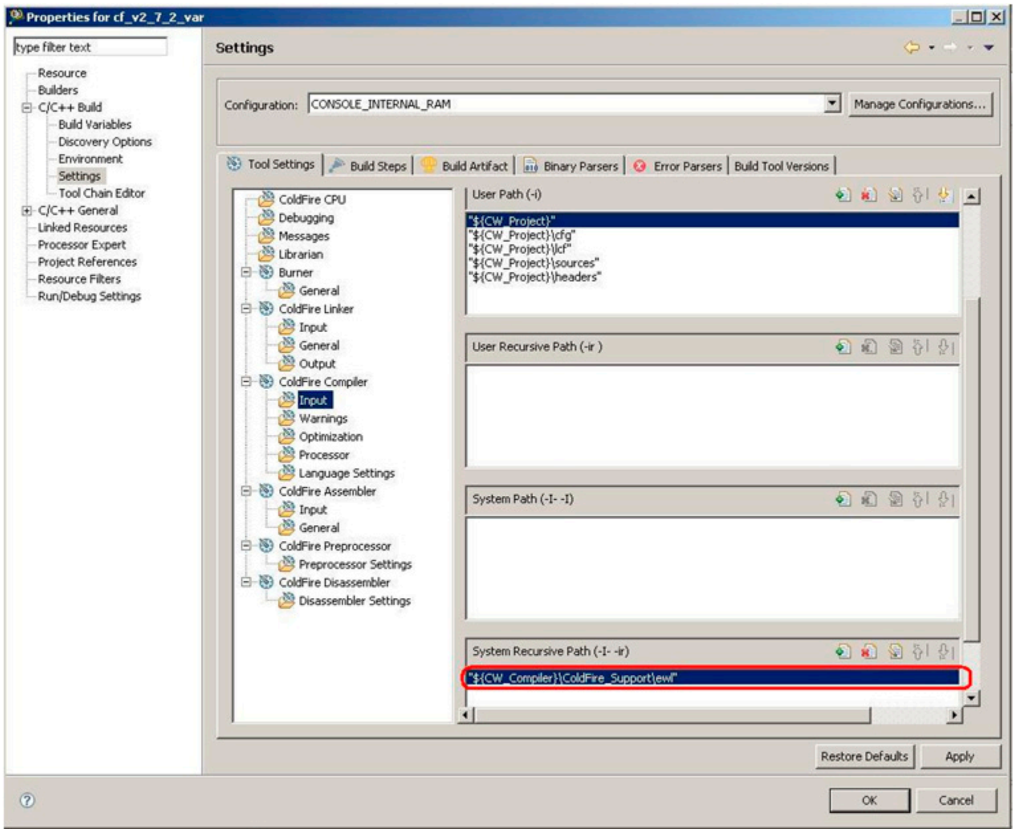
C_4i_CF_RegABI_MSL.a
C_4i_CF_RegABI_Runtime.a

4 Access Include Paths

In the old projects "System Access Paths" contained entries to MSL code. The importer translates them to point to the new libraries. In MCU V10.0, the "System Recursive Path" has an entry which points to the root EWL folder:

```
"${CW_Compiler}/ColdFire_Support/ewl"
```

Figure 2. System Recursive Path



Configuration: CONSOLE_INTERNAL_RAM

User Path (-i)

```
"${CW_Project}"
"${CW_Project}\cfg"
"${CW_Project}\lcf"
"${CW_Project}\sources"
"${CW_Project}\headers"
```

User Recursive Path (-ir)

System Path (-I- -I)

System Recursive Path (-I- -ir)

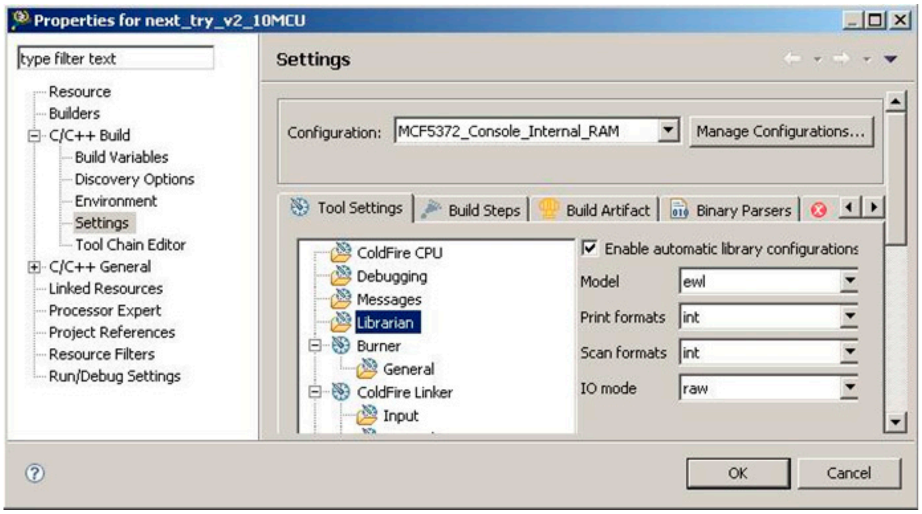
```
"${CW_Compiler}\ColdFire_Support\ewl"
```

5 Librarian

In the **Librarian** panel, you can select a library model from a pre-defined list of models:

- ewl
- ewl_c++
- c9x
- c9x_c++

Figure 3. Librarian Panel



Configuration: MCF5372_Console_Internal_RAM

☑ Enable automatic library configurations

Model: ewl

Print formats: int

Scan formats: int

IO mode: raw

- **ewl** and **ewl_c++** models have a smaller memory footprint. Also, the **ewl** and **ewl_c++** models have relevant sub-models that allow the user to select the desired print and scan formatters and the IO scheme.
- **c9x** and **c9x_c++** models are fully C99 compliant. The **c9x** and **c9x_c++** models do not have sub-models.

For the print and scan sub-models the available choices and the functionality that they cover are listed in the "Libraries" section above.

The IO mode can be either raw or buffered. The raw IO mode implies that no buffer is used for the IO operations. For example, if the code is only doing `fread()` and/or `fwrite()` on a file in binary mode, then raw model should be used, otherwise, using raw IO mode will result in poor performance. In case of the buffered IO mode, all IO passes through a buffer.

If you select the **c9x** or **c9x_c++** model, the sub-model drop-down lists are disabled, as they do not apply to these models.

The **Enable automatic library configuration** checkbox in the **Librarian** panel, determines whether or not the EWL mechanism of library selection will be used by the build tools (compiler and linker). When this checkbox is not checked, you need to manually add the required library files to the project.

You can select the relevant library files by choosing the correct architecture (CF v1, v2-4) and specifying whether or not FPU and PIC/PID is used in the **Project -> Properties -> C/C++ Build -> Settings -> ColdFire Compiler -> Processor** panel.

NOTE If you import a CFv1 MCU 6.2 project or a CW 7.1 project to MCU 10.0, the importer selects **ewl** model for the project, and sets print and scan formatters to **int** and IO mode to **raw**.

If the IO mode is set to **raw**, compilation of some projects may generate the following conversion error:
`../MCU/ColdFire_Tools/Command_Line_Tools/mwldmcf:`

```
Undefined : "__files".
```

You can resolve the error by changing the IO mode to **buffered.**

6 Parameter Calling Convention

The parameter passing affects space and time performance. The best performance for both occurs when selecting the register passing ABI.

The default calling convention for ColdFire compiler is register ABI. Compact ABI and Standard ABI are available through one of the following methods:

1. Use `declspec` for function prototypes, also described in the "Declaration Specifications" section of the ColdFire Architectures Build Tools Reference Manual:

   ```
   asm void __declspec(compact_abi) check_CC(unsigned long)
   {
   ....
   }
   ```

2. Use `pragma` to specify the calling convention for function defined from:

   ```
   #pragma compact_abi
   asm void check_CC(unsigned long)
   {
   ....
   }
   ```

NOTE A function prototype specifies the function's name, arity, argument types, and return type. If a function does not have a prototype, a default behavior is selected. But, if the default behavior does not match with the function behavior it is possible to have unexpected results. To avoid this problem you need to use the require functions prototypes option in **Project > Properties > C/C++ Build > Settings > Language Settings**.

7 Assembly Function Declarations and Definitions

For all pure assembly functions in the application, the function definition and declaration(s) should contain a `__declspec` qualifier that defines the parameter passing convention. For example:

```
asm void __declspec(register_abi) TrapHandler_printf(void)
```

The compiler generate the following warning message for all the pure assembly function definition and declaration that does not contain the `__declspec` qualifier:

```
WARNING! "Possible abi conflict, use __declspec(register_abi):"
```

Therefore, while converting a MCU 6.2 CF v1 project or a CW 7.1 CF v2-4 project to a MCU 10.0 project, you need to modify the code of the project such that the assembly functions contain the __declspec qualifier.

Also, if the function contains code that uses a calling convention other than Register, and is called from a C function, the code should be modified to use the Register parameter passing convention (if the calling convention is not explicitly mentioned using __declspec qualifier).

For example, the following function assumes that the parameter is on stack, at offset 14, not in register:

```
asm void mcf5xxx_wr_vbr(unsigned long) { /* Set VBR */
move.14(SP),D0
movec d0,VBR
nop
rts
}
```

The code should be modified to use the Register parameter passing convention. In this example, the following code line should be removed because a C function which calls this assembly function put the parameter in the D0 register.:

```
move.14(SP),D0
```

Another solution is to add the __declspec qualifier with the proper calling convention (as intended in the original project). Please note that the performance may be affected if the calling convention is other than Register.

8 EWL Memory Allocation Scheme

EWL supports an improved memory allocation scheme. The memory allocation scheme in EWL requires the following symbols to be defined in the LCF file:

- ___mem_limit
- ___stack_safety: Specifies the size of the cushion between the stack and the heap.

Listing 1 shows an example.

Listing 1. Example: EWL Memory Allocation Scheme

```
___mem_limit = ___HEAP_END;
___stack_safety = 16;
```

In Listing 1, ___stack_safety is set to 16 bytes. You can add these symbols to the LCF file right after the definition of ___HEAP_END.

9 Additional Information

For more information about MCU 10.0:

- Refer Release Notes in the <*CodeWarriorInstallDir*>`\MCU\Release_Notes` folder.
- Refer Microcontrollers V10.0 documentation in the <*CodeWarriorInstallDir*>`\MCU\Help\PDF` folder

NOTE *CodeWarriorInstallDir* is the folder where MCU 10.0 is installed.

- Visit http://www.freescale.com/support for additional assistance.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals”, must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.