Converting Projects to CodeWarrior™ ColdFire® V7.0

by: Marcel Achim, Oscar Gueta and Alfredo Soto

Project Conversion

Converting a project created in CodeWarrior™ Development Studio for ColdFire® Architectures V6.4 to V7.0 is a straightforward procedure. Two major changes in V7.0 may impact projects when moving from V6.4 to V7.0:

1. Default alignment changed to ColdFire alignment: To increase runtime performance, all default alignments on V7.0 have been changed to modulo-4 (ColdFire alignment). All libraries are now built with these settings.
2. Simplified library configurations: Removing the TRK libraries reduced the number of libraries by half. When debugging virtual console-based targets, include the source code module.

This document provides a general overview of the changes in V7.0 and a quick guide to successfully migrate projects from V6.4 to V7.0. The following items are covered in this document:

Terms and Abbreviations

The following terms and abbreviations are used in this document:

- MSL — Main Standard Libraries
- SZ_ — Designation for special, small library configurations appropriate for limited-memory ColdFire devices such as the MCF52235.
- TRK — Target Resident Kernel
- V6.4 — Refers to CodeWarrior Development Studio for ColdFire Architectures, Version 6.4.
- V7.0 — Refers to CodeWarrior Development Studio for ColdFire Architectures, Version 7.0.

Libraries

CodeWarrior for ColdFire V7.0 delivers simplified library configurations providing the best performance with the highest code density possible. Removing the TRK libraries (used for virtual console targets) decreases the number of MSLs by half.

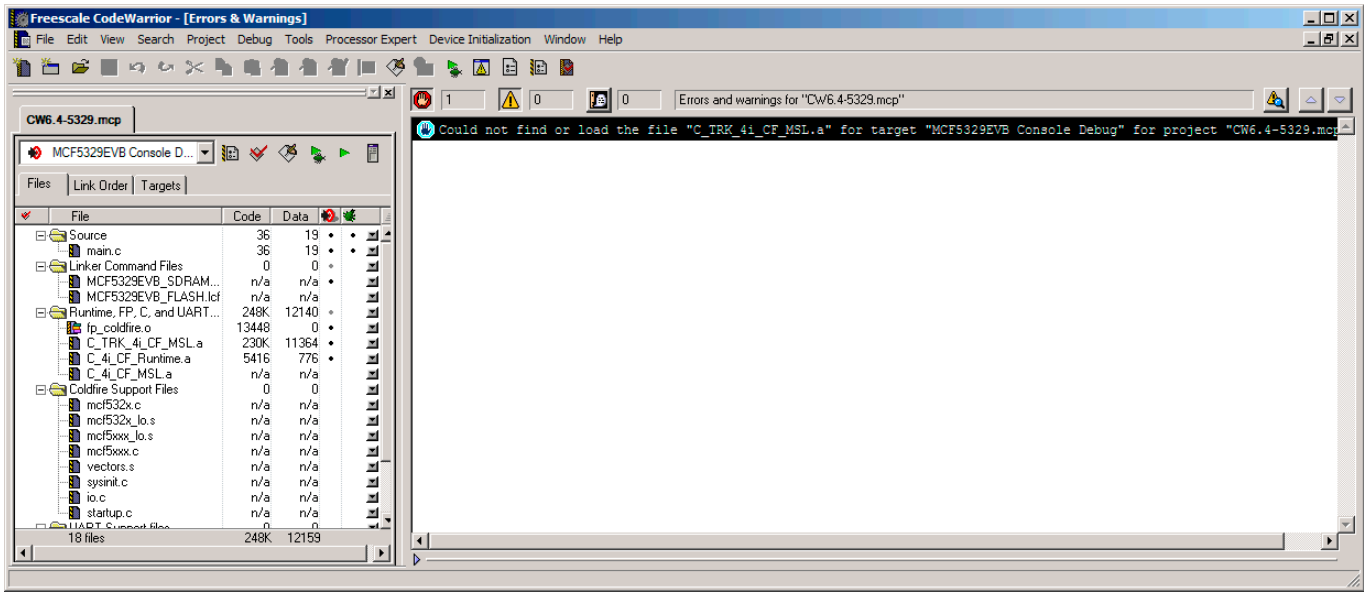
Consider these facts about the V7.0 libraries:

- V7.0 adds a new set of libraries specifically for ColdFire V4. These libraries include the V4 designation in their name (e.g., `C_V4_4i_CF_MSL.a`).
- V7.0 no longer builds 2i libraries by default. (For details, see Avoiding 2-byte Integers below.)
- V7.0 builds all libraries for data smaller than 8 bytes with A5-relative addressing. This limits the amount of accessible global data to 64K for the data affected by this setting. Select this setting in the linker panel to use A5 relative addressing for non-MSL data as well.
- All libraries now compile with ColdFire alignment as the default. If this is not what is needed, rebuild the libraries. (For more details, see Alignment below.)
- Full C99 support is included only on full libraries (libraries without the SZ designation in their name).
- In addition to the pre-built binaries, the CodeWarrior development tools include the source code and project files for MSL so that you can customize the libraries.

NOTE The MCF52235 and related processors have smaller memories than many other members of the ColdFire family. The V7.0 libraries include special, small library configurations appropriate for such limited-memory devices. The names of these small library files include the designation SZ_.

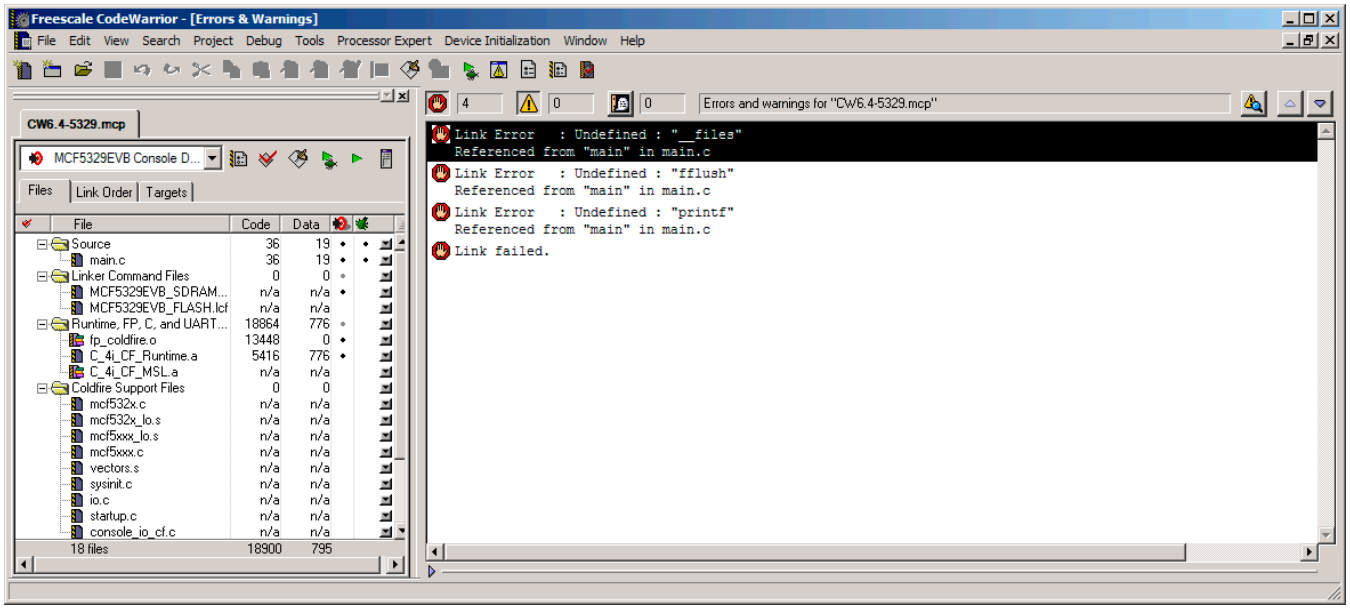
Migrating Virtual Console Targets from V6.4 to V7.0

- As mentioned before, the TRK libraries no longer exist in V7.0, so opening a V6.4 project in V7.0 displays the error message shown in this figure:



The solution is simple: just remove that library from the project.

- Trying to link the virtual console target causes several undefined symbols to appear, as shown below:



Those symbols are implemented in `C_4i_CF_MSL.a`. You must add that library to the project for all targets. Of course, if those symbols are not needed, the library is not needed either.

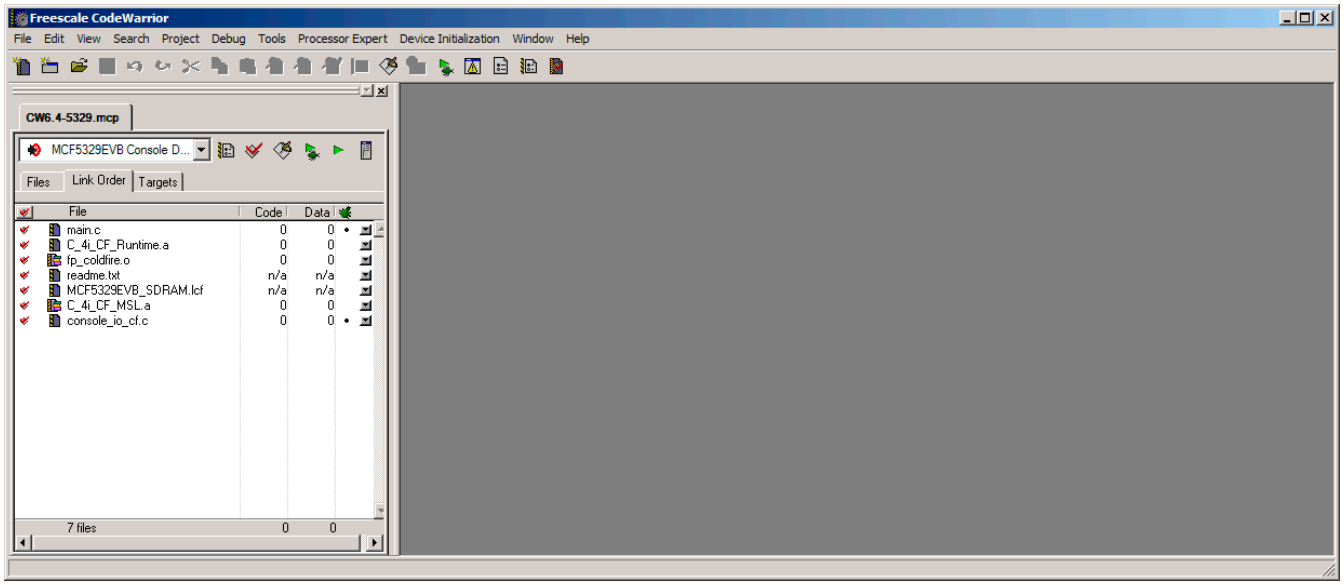
- Output of `printf()` for this library goes to the serial port. You need another file to send the output to the virtual console:

  `msl/MSL_C/MSL_ColdFire/srcs/console_io_cf.c.`

  Copy the file to your project directory and add the file to your console target only.

NOTE Add this file to the console target only, otherwise all targets will send their output to the UART.

- Make sure to add both files to the **Link Order** tab:



- Build the project and start debugging normally.

Other Linking Issues

If you run into unreferenced symbols, `C_4i_CF_MSL.a` contains most of the symbols needed. Include this library in your project while removing any other MSL library (i.e. SZ libraries).

Other non-TRK targets are not affected by the migration. However, there may be situations where segment adjustments may be needed in the Linker Configuration File (LCF) in order to complete the linking.

Avoiding 2-Byte Integers

The ColdFire cores provide only 32-bit arithmetic support. Avoid using integers as 2-byte quantities whenever possible, as this introduces data size conversions that impact code size.

Performance Improvements

V7.0 incorporates the following performance improvements.

Alignment

The Code Generation settings panel's alignment option affects both structure internal alignment (offsets within structures) and global data alignment (memory address). The default value for this setting in V6.4 was word (16 bit) (modifiable via pragma statements), reminiscent of the early 68000 16-bit bus used in Macintosh computers. ColdFire is built on a 32-bit (long) architecture and using the 68000 alignment imposes a runtime penalty for misaligned memory accesses, since the processor performs multiple memory fetches to recompose misaligned data.

A major change introduced in V7.0 addresses this issue and improve performance is changing the native alignment from 68000 to ColdFire. Thus `#pragma align=native` is now automatically equivalent to `#pragma align=coldfire`.

In scenarios where native alignment is set to ColdFire and there is a need for compatibility with previous releases, you must explicitly reset the native alignment in your code using a new pragma:

```
#pragma native_coldfire_alignment on|off|reset
```

If binary compatibility with the legacy 68000 alignment is absolutely required, the MSL C and C++ libraries must be recompiled manually with `#pragma native_coldfire_alignment off`.

A set of backwards-compatible projects and makefiles preset to these settings can be found in:

```
msl/MSL_C/MSL_ColdFire/Projects
msl/MSL_C++/MSL_ColdFire/Projects
```

These are identified with a `.word` suffix in their name.

> **NOTE** Building the backwards-compatible projects overwrites the default pre-built libraries.

Passing Parameters ABI

The Register and Standard ABI align pass arguments on 4-byte boundary offsets. The Compact ABI forces on a 2-byte alignment all parameters except for those passed as part of a variable arguments list; these are aligned on 2-byte boundaries for MC68000 alignment and on 4-byte boundaries otherwise. Under the Compact ABI, passing mixed-size arguments using the MC68000 alignment can potentially degrade performance.

You are encouraged to use the Standard or Register ABI. The Register ABI passes the first few arguments in registers when possible, reducing code size and increasing performance. This is important as most of the V6.4 projects are compiled against compact libraries and the default setting on each project is compact as well.

V7.0 includes compact libraries and the Compact Parameters Passing setting for backwards compatibility.

When switching to a non-compact ABI, be sure to include the correct libraries (RegABI or StdABI) for your project and select the correct Parameter Passing in your project settings.

Backwards Compatibility

The simplest backwards-compatible solution that helps achieve better runtime performance without changing internal data type alignment involves the linker. To support unreferenced objects stripping, the compiler allocates each variable in its own section in the ELF object, and each section possesses a header specifying its alignment (`sh_addralign`) set from the processor settings panel. The LCF ALIGNALL directive overrides this value.

In a simple scenario where only variable addresses are aligned, the LCF adds `ALIGNALL(4)` in the `data` and `bss` sections. The `ALIGN` directive does not force individual section alignment; it only aligns the current PC on the specified boundary. A V6.4 project using the `ALIGNALL` directive forces the linker to allocate all data sections on a `mod-4` boundary, but does not affect the internal alignment of `structs`.

See the Alignment section above for M68000 binary alignment compatibility.

Memory Map

In general, the linker memory map for stationery and examples is the same in V7.0 as in V6.4.

A typical memory map for a ColdFire device has a vector segment, and a combined `.text` and `.rodata` segment. It has global data split between absolute and small `.data` and `.bss` segments, heap and stack, and memory-mapped devices. Memory map segment organization varies as a function of kind (Flash, SRAM, SDRAM) and size.

Addressing and Sections

The current toolset supports the following addressing modes: near, smart (code), far, small (data) and position-independent (code and data). This is identical to those supported in V6.4.

ELF sections in V7.0 have the same default addressing as in V6.4. Default values can be changed globally, either by creating new ones using `#pragma section`, or by redefining the properties of existing sections. You can also change values locally using declaration qualifiers. The following table shows the types of ELF sections and form of addressing associated with each object class.

Table 1 Object Class Associations

Object class	ELF sections		Addressing
Code	`.text`		Standard
Data	`.data`	`.bss`	Standard
Sdata	`.sdata`	`.sbss`	near_data

Table 1 Object Class Associations (continued)

Object class	ELF sections		Addressing
Const	`.rodata`		Standard
String	`.rodata`		Standard
Absolute	`.abs`		far_absolute
Exceptions	`.exception`		far_absolute
Exceptlist	`.exceptlist`		far_absolute
Picdynrel	`.picdynrel`		far_absolute
Piddynrel	`.piddynrel`		far_absolute

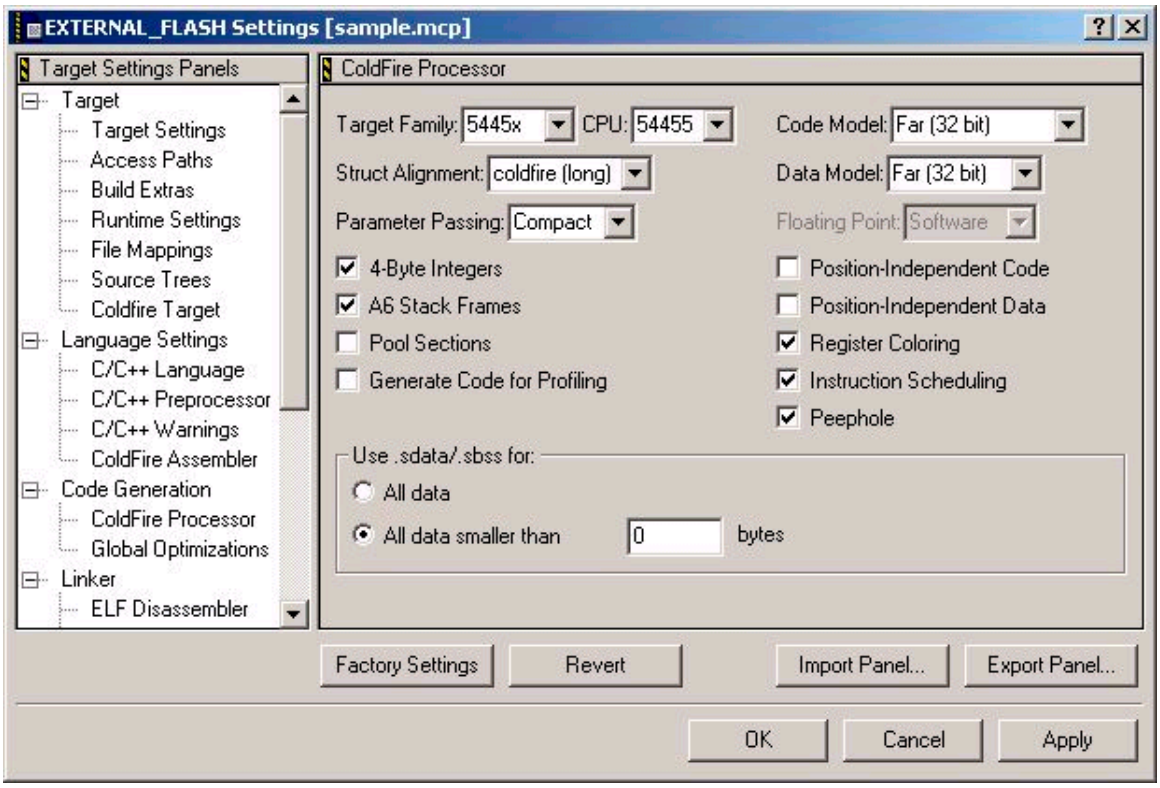
Code Generation

CodeWarrior Development Studio for ColdFire Architectures V7.0 supports the following new EVBs and processors:

- M54455EVB
- M5373EVB

New Code Generation Options

New code generation options are available in the V7.0 Code Generation panel of your project settings. They are: Register Coloring, Peephole, Instruction Scheduling, and Pool Sections.



Register Coloring

Enable Register Coloring to make the compiler allocate local variables to registers whenever possible. The final effect of this option is smaller code size even with debug code. To achieve this, the compiler performs variable lifetime analysis, causing some variables used as temporaries to disappear. Variable lifetimes are recorded in the debugging information to provide accurate runtime values, provided the variable still exists. Disable Register Coloring to force all local variables to be stack-based, except for compiler-generated temporaries.

This setting corresponds to `#pragma no_register_coloring`.

Peephole

When on (default setting) the compiler compiles long instruction sequences into shorter ones to reduce code size and potentially increase performance. It does not affect debugging unless the resulting instruction is a memory-to-memory operation which might make a variable used as temporary disappear.

Scheduling

Enable Scheduling to select instruction scheduling for the ColdFire V4 whenever the optimization level is two or higher. It schedules instructions to minimize instruction latency, enhancing performance.

Pool Sections

The Pool Sections setting instructs the compiler to pool definitions into a single section rather than generating a section per definition. This applies for `text`, `data` and `bss`. Be aware that it prevents deadstripping from happening because the linker cannot break contiguous sections.

Predefined symbols and processor families

The compiler defines a few symbols to support the ColdFire processors and families. The symbol `__COLDFIRE__` is always set to represent the currently selected processor. All processors and families are symbolically represented with `__MCF`*`zzzz`*`__`, where *`zzzz`* represents either a family or a processor part number.

ColdFire processors are combined into functional families. Selecting a family brings the parts belonging to that set into scope. This selection occurs in the code generation settings panel. Selecting a family and processor defines a series of built-in macros, making it possible for users to specialize their code by family and processors within a family. The following table shows the families and the processors associated with each family.

Table 2 ColdFire Processor Families and Associated Parts

Family	Parts
521x	5211, 5212, 5213, 5214, 5215
5221x	52210, 52211, 52212, 52213
521x0	52100, 52110
5222x	52221, 52223
5223x	52230, 52231, 52232, 52233, 52234, 52235, 52236
528x	5280, 5281, 5282
5206e	5206e
5207_8	5207, 5208
523x	5235
524x	5249
525x	5251, 5253
5270_1	5270, 5271
5272	5272
5274_5	5274, 5275
5307	5307
532x	5327, 5328, 5329

Table 2 ColdFire Processor Families and Associated Parts

Family	Parts
537x	5371, 5372, 5373
5407	5407
5445x	54450, 54451, 54452, 54453, 54454, 54455
547x	5470, 5471, 5472, 5473, 5474, 5475
548x	5480, 5481, 5482, 5483, 5484, 5485

For example, selecting 54455 brings in scope all other processors belonging to that family, as well as the family. The resulting set of built-in symbols is { __MCF5445x__, __MCF54450__, __MCF54451__, __MCF54452__, __MCF54453__, __MCF54454__, __MCF54455__ }. The processor values are all distinct and the family value is simply defined. The following code illustrates this example.

```
 #ifdef __MCF5445x__
/** this is true for **/

#if __COLDFIRE__ == __MCF54455__
/** this is true **/
#elif __COLDFIRE__ == __MCF54454__
/** this id false since the 54455 was selected **/
#endif

#endif

#ifdef __MCF547x__
/** this is false, we didn't select that family **/
#endif

#if __COLDFIRE__ == __MCF5445x__
/** this is false since __COLDFIRE__ represents a part and not a family **/
#endif
```

New and Obsolete Options

The following options are new or are obsolete in this release.

New Options

V7.0 has the following new option:

- New Declaration Specifier: `__declspec(bare)`

  This specifier prevents the compiler from generating '_' prefixed link time names for C objects (it does not prevent C++ name mangling). It is useful for interfacing with assembly language.

Obsolete Pragmas

The following obsolete pragmas have been deprecated in V7.0.

Table 3 Obsolete Pragmas

Pragma	Purpose
code68020	Enabled extended MC680x0 addressing modes
code68881	Enabled FPU code generation
Fp_pilot_traps	Enabled PalmPilot FP emulation library calls
IEEEdoubles	Select between 64, 80 or 96-bit format for doubles
interrupt_fast	Documentation-only entry
Macsbug	Write routine names in binaries following code
Mpwc	Use MPW C calling conventions
oldstyle_symbols	Write routine names in binaries following code
Parameter	Breaks ABI, register spec to be used instead
SDS_debug_support	DWARF 1 support
Segment	MacOS segment loader support
stack_cleanup	Support for Pascal calling conventions on MacOS
Toc_data	CFM 68K support

Obsolete and New Alignments

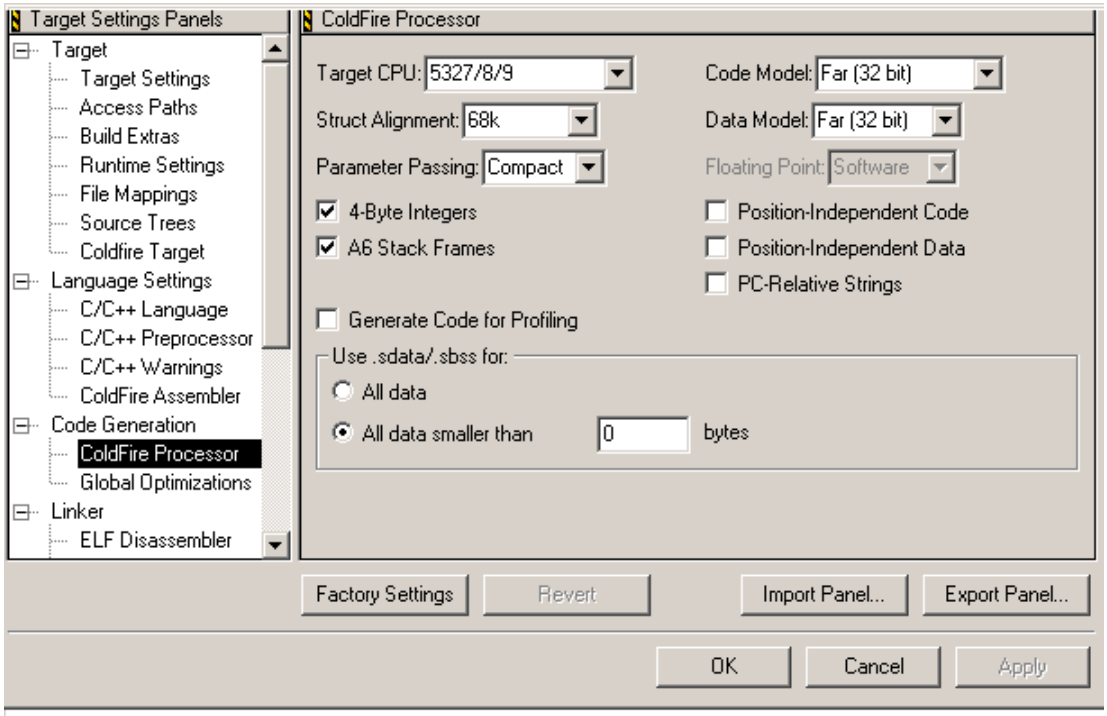
Macintosh-legacy area alignment values are deprecated and replaced with more meaningful names. Old names are still accepted but display compile-time warnings.

Table 4 Alignment value changes

Deprecated names	Mac68k, mac68k4bytes, power
New accepted values	byte, 68k or word, coldfire or long

PC-Relative Strings removed from Code Generation panel

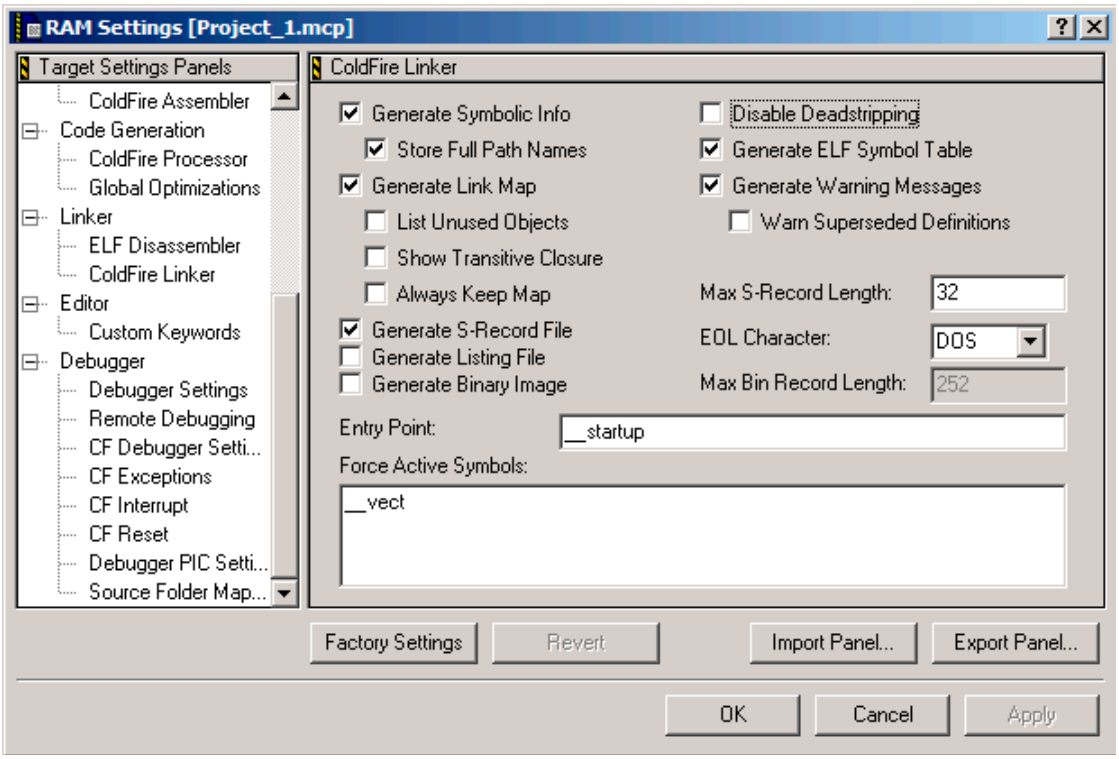
The PC-Relative Strings setting was removed from the Code Generation panel to provide consistent behavior with all PC-relative pragmas: `pcrelstrings`, `pcreldata` and `pcrelconstdata`. You can use the `__declspec(pcrel)` declaration specifier to force individual declarations to use PC-relative addressing.


Target Settings Panels
Target
Target Settings
Access Paths
Build Extras
Runtime Settings
File Mappings
Source Trees
Coldfire Target
Language Settings
C/C++ Language
C/C++ Preprocessor
C/C++ Warnings
ColdFire Assembler
Code Generation
ColdFire Processor
Global Optimizations
Linker
ELF Disassembler
ColdFire Processor
Target CPU: 5327/8/9
Code Model: Far (32 bit)
Struct Alignment: 68k
Data Model: Far (32 bit)
Parameter Passing: Compact
Floating Point: Software
4-Byte Integers
A6 Stack Frames
Position-Independent Code
Position-Independent Data
PC-Relative Strings
Generate Code for Profiling
Use .sdata/.sbss for:
All data
All data smaller than
0
bytes
Factory Settings
Revert
Import Panel...
Export Panel...
OK
Cancel
Apply

New Linker Options

The following new linker options are available in the V7.0 ColdFire Linker panel of your project settings:

- Always Keep Map
- Generate Listing File



Always Keep Map

This feature retains the link map (`.xMAP`) so that even if the link fails, you can still use it to investigate placement-related link failures.

Generate Listing File

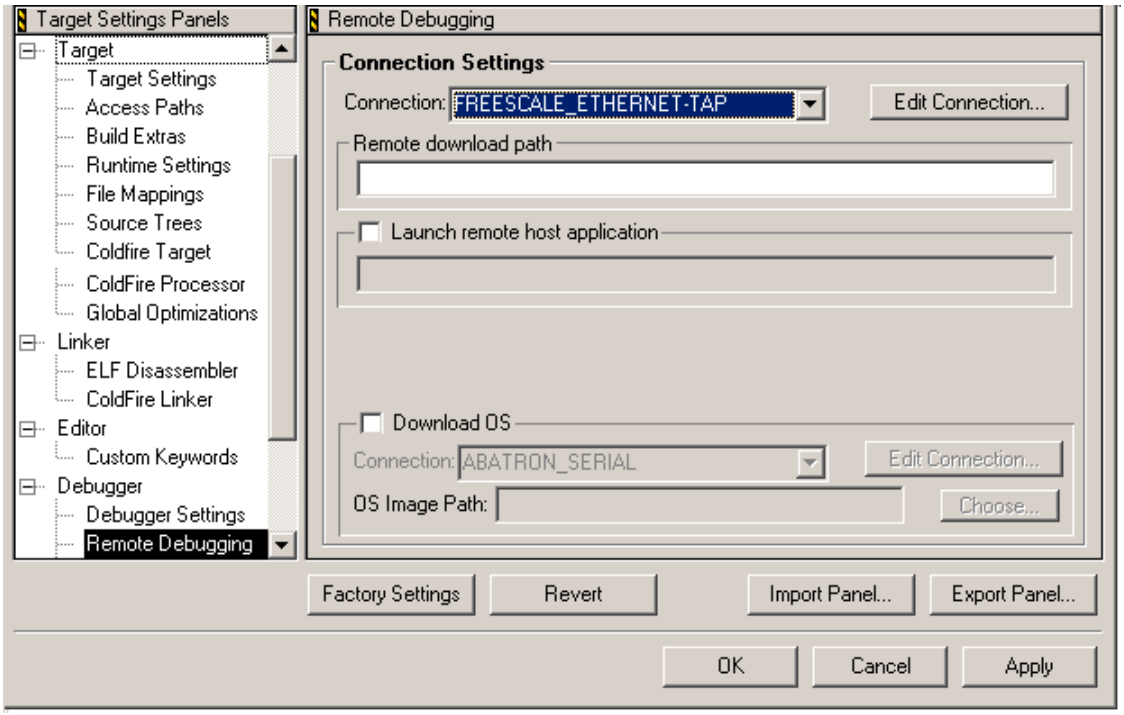
This feature generates a listing of the output binary, formatting the listing according to the disassembler settings.

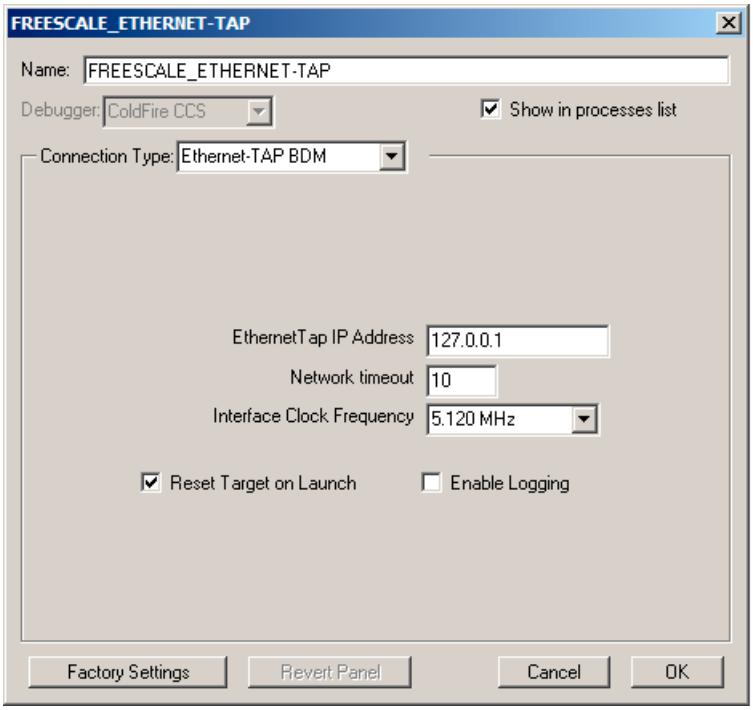
New Debugging Interfaces

CodeWarrior™ Development Studio for ColdFire® Architectures V7.0 adds support for the following debugging interfaces:

- CodeWarrior EthernetTAP remote connection support
- P&E Cyclone MAX

The corresponding settings are available in the Connection Settings panel of your project settings.





DWARF2 support

CodeWarrior™ Development Studio for ColdFire® Architectures V7.0 supports DWARF2 debug information, which makes it compatible with most third-party debuggers in the market. This is a change from V6.4, which supported DWARF1.

> **NOTE** While it is possible to link objects built with DWARF1 debug information in V7.0, debug information will not be available for those objects.

Additional information

See the CodeWarrior™ Development Studio for ColdFire® Architectures V7.0 Release Notes and documentation for more information.

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

***For Literature Requests Only*:**
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-521-6274 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: TN256

6 November 2007

freescale semiconductor