CodeWarrior™ Development Tools

Porting Guide

Revised: 4/5/02

How to Contact Metrowerks:

Corporate Headquarters	Metrowerks Corporation 9801 Metric Blvd. Austin, TX 78758 U.S.A.
World Wide Web	`http://www.metrowerks.com`
Ordering & Technical Support	Voice: (800) 377-5416 Fax: (512) 997-4901

Table of Contents

1

Welcome

Welcome to the *CodeWarrior Porting Guide*. This guide offers hints and tips on moving your programming project from other software development environments to CodeWarrior. It also points you to other CodeWarrior documentation for more in-depth topics.

Use this guide if you are new CodeWarrior or you are converting a programming project from another development system to CodeWarrior.

This guide refers to development tools that might not be available with the CodeWarrior package you have. Consult the *QuickStart* guide that came with your CodeWarrior package for information on what is in your CodeWarrior package.

This chapter has these sections:

- Read the Release Notes
- What is New in This Release
- What is in This Guide
- Where to Go From Here

Read the Release Notes

Before using CodeWarrior, read the information in the `Release Notes` folder, which is on the CodeWarrior CD-ROM and installed in the CodeWarrior folder on your computer's hard drive.

The release notes contain important information about new features, bug fixes, and any late-breaking changes.

What is New in This Release

This guide has been significantly rewritten. It now has new and updated information on moving a programming project to CodeWarrior.

Also, future revisions of this guide will add information about moving to CodeWarrior from more software development environments.

What is in This Guide

The chapters in this guide are listed in Table 1.1.

Table 1.1 Chapters in this guide

Read this chapter…	to learn about
“Welcome”	using this guide
“Common Porting Issues”	solutions to common problems you might encounter when porting your project to CodeWarrior
“Microsoft® Visual Studio® Porting Issues”	converting a programming project from Miscrosoft Visual Studio to CodeWarrior

Where to Go From Here

This guide only discusses unique issues and problems that you might encounter when converting your programming project to CodeWarrior.

If you are new to CodeWarrior, you will find these manuals and references the most useful:

- *CodeWarrior IDE User Guide*—how to use the CodeWarrior IDE to edit, search, navigate, compile, link, debug, and manage programming projects
- *C Compilers Reference*—discusses the CodeWarrior implementations of the C, C++, Embedded C++, and Objective C programming languages
- *MSL C++ Reference*—describes the Metrowerks Standard Library for C
- *MSL C Reference*—describes the Metrowerks Standard Library for C++
- *Targeting* manuals—describe how to use CodeWarrior to develop software for a specific processor or operating system

  For example, to learn how to use CodeWarrior to create software for the Apple Macintosh, read *Targeting Mac OS*.

2

Common Porting Issues

This chapter covers general problems and differences among other software development tools and CodeWarrior. The topics in this chapter describe the CodeWarrior features that have subtle and obvious differences from most other development environments.

This chapter has these sections:

- C and C++ Issues
- Mac OS Issues

C and C++ Issues

This section covers topics on differences between the CodeWarrior C and C++ compilers and other compilers:

- About Metrowerks Standard Libraries
- ARM and Other C++ Implementations
- CFront and Metrowerks C++
- UNIX and POSIX Libraries
- The sizeof() operator and int

About Metrowerks Standard Libraries

The Metrowerks Standard Libraries (MSL) for C and C++ comply with the libraries described by the ANSI/ISO C and C++ Standards.

MSL C and MSL C++ also have functions that are commonly available but are not part of their respective standards. For example, MSL C has functions that are commonly available in UNIX.

ARM and Other C++ Implementations

CodeWarrior C++ conforms to the ANSI/ISO C++ standard, although it has options to compile source code that conforms to the C++ specification in the *Annotated C++ Reference Manual* (ARM).

For information on compiling non-ANSI C++ source code, refer to the *C, C++, And Assembly Language Reference.*

CFront and Metrowerks C++

MSL C++ conforms to the ANSI/ISO C++ Standard for the C++ streams libraries. There are a few variations from the standard as accepted and published in such books as Stroustrup's *The C++ Programming Language, 2nd edition* (Addison-Wesly, 1991) and Stanley B. Lippman's *C++ Primer, 2nd edition* (Addison-Wesley, 1991) commonly referred to as the C++ Programming Language or CFront C++.

For information on compiling non-ANSI C++ source code, refer to the *C Compilers Reference.*

#include header naming conventions

ANSI C++ no longer require a file name extension for the include files. However, the file extension naming conventions are provided for by ANSI C++, and included in Metrowerks.

CFront C++

```
#include <iostream.h>
```

ANSI C++

```
#include <iostream>
```

File open modes

Listing 2.1 shows the valid combinations of `ios::openmode` for opening a file as defined in the ANSI C++ library—these are defined in terms of the equivalent `modestr` used in `fopen(s, modestr)` (see para 7.9.5.3 of ANSI Standard for C).

Listing 2.1 File open modes

```
ios::in   modestr = "r"
ios::out | ios::trunc  modestr = "w"
ios::out | ios::app  modestr = "a"
ios::in  | ios::bin  modestr = "rb"
ios::out | ios::trunc | ios::bin modestr = "wb"
ios::out | ios::app   | ios::bin modestr = "ab"
ios::in  | ios::out     modestr = "r+"
ios::in  | ios::out   | ios::trunc modestr = "w+"
ios::in  | ios::out   | ios::app modestr = "a+"
ios::in  | ios::out   | ios::bin modestr = "r+b"
ios::in  | ios::out   | ios::trunc | ios::bin modestr = "w+b"
ios::in  | ios::out   | ios::app   | ios::bin modestr = "a+b"
```

All other combinations are invalid and no file is opened and no error message is produced.

File open and close testing

Use the `is_open()` function to test for an open file.

In CFront C++:

```
ofstream to("testFile");
  if (!to ) /* ... */
```

In ANSI C++:

```
ofstream to("testFile");
  if (to.is_open() == 0) /* ... */
```

iostream `fail()` vs. `eof()`

Historically (before the advent of the ANSI/ISO C++ specification) the `eofbit` was set haphazardly. ANSI C++ libraries do not set the `eofbit`, therefore the previous practice of using the function `eof()`, which gave a non-zero result when the end of file was reached, is no longer useful. Instead you should test the value of the `fail()` function, which will pick up both `eof` and other kinds of failure, as shown in Listing 2.2.

Listing 2.2 fail() vs. eof()

```
#include <iostream>
#include <fstream>
```

```
void main()
{
  // fill file with 10 x's
         ofstream out("testfile");
         for(int i = 0; i < 10; i++ ) out.put( 'X');
         out.close();

        char c = 0;
        ifstream input("testfile");

    //  while ( !input.fail() )   this leaves EOF character
    //  while( input.peek() != EOF )this works but less safe
  while ( !input.fail() && input.peek() != EOF )
        {
           c = input.get() ;
           cout << "char: " << c << endl;
       }
     input.close();
}
```

UNIX and POSIX Libraries

There are header files that provide some of the functions in the POSIX standard and many functions found in UNIX libraries that are not specified in the ANSI C or ANSI C++ standards.

Refer to `fnctl.h`, `stat.h`, `unistd.h`, `unix.h`, `utime.h`, and `utsname.h` in the *C Library Reference* for more information.

The `sizeof()` operator and `int`

When programming in C, do not assume that the value of a `sizeof()` operator is assignment compatible with the `int` or `long` data types.

According to the ANSI C standard `sizeof()` returns a value of type `size_t`, defined in `stddef.h`.

The `size_t` and `int` data types are often the same, but might differ depending on the platform or processor. Refer to Listing 2.3 for an example of how a program executes incorrectly when it assumes that `size_t` and `int` are the same size.

Listing 2.3 Making assumptions about `sizeof()` and `int`

```
#include <stdio.h>
#include <stddef.h>

typedef struct {
  charbigArray[100000];
} MyStruct;

void main(void)
{
  int j;
  size_t t;


  j = sizeof(MyStruct); /* This doesn't work */
  t = sizeof(MyStruct); /* This works */

  printf("bad size of MyStruct = %ld\n", j);
  printf("good size of  MyStruct = %ld\n", t);
}

/* Output, running on PPC:

bad size of MyStruct = -2036334591
good size of  MyStruct = 100000

*/
```

There are a few variations from the MSL implementation of C++ and *previous* versions of ANSI/ISO C++. MSL C++ conforms as closely as possible to the ANSI/ISO C++ standard.

- The `fstream` class is now included for mixed input and output.
- The file reading facilities `tellg()`, `tellp()`, `seekg()`, and `seekp()` are now included in the ANSI C++ standard.
- The STL algorithms are now part of the proposed standard and conflict with older versions of the STL Libraries.

Mac OS Issues

The topics in this section deal with common problems you might have when porting your programming project to Apple's Mac OS. Of course, this section cannot cover every aspect of porting a project to a new operating system. Instead, it gives you tips and references to other documentation to help you.

The topics in this section are:

Where to Find More Information About Mac OS

You'll find comprehensive documentation, sample source code, and other resources for Mac OS development at Apple's web site for software developers, `http://www.apple.com/developer/`.

To learn how to use CodeWarrior to develop software for Mac OS, see *Targeting Mac OS*.

Console I/O for Mac OS

For programs that use simple text input/output without calling on any Mac OS-specific features, CodeWarrior provides SIOUX. SIOUX is a Mac OS software package that automatically opens a text window, accepts characters from the keyboard, and handles menus. SIOUX does all this transparently; you, the programmer, do not have to explicitly invoke SIOUX.

Simply by calling a standard C or C++ function that reads from or writes to the standard input, output, or error files will invoke SIOUX automatically.

Refer to the *MSL C Reference* for information on customizing SIOUX.

Command-Line Arguments for Mac OS

The C/C++ `ccommand()` function provides a dialog box that allows the user to enter text as if it were at the command line. Refer to `ccommand()` in the *MSL C Reference* for more information.

File Redirection for Mac OS

For UNIX-style file redirection, use the C/C++ `ccommand()` function. Refer to ccommand in the *MSL C Reference* for more information.

Including Files In C/C++ on Mac OS

When using CodeWarrior on Mac OS, CodeWarrior C/C++ handles subdirectory names in `#include` directives differently than UNIX compilers. In particular, Mac OS uses the colon, ":", as a directory separator character, not the slash, "/".

For example, when using CodeWarrior on a Mac OS computer, issuing this directive

```
#include "special/datatypes.h"
```

will actually include a file named "`special/datatypes.h`," not the file "`datatypes.h`" in the "special" directory.

Specify Subdirectories for #include Directives

The recommended way to specify a directory for an `#include` directive is to add the directory to the list of access paths in the project's **Access Paths** project settings panel and remove references to subdirectories in all `#include` directives. Refer to the *CodeWarrior IDE Guide* for more information on access path preferences.

Another way to specify a subdirectory for an `#include` directive is to convert the pathname to a Mac OS pathname. For example

```
#include "special/datatypes.h"
```

becomes

```
#include ":special:datatypes.h"
```

3

Microsoft® Visual Studio® Porting Issues

This chapter covers common problems and issues you will encounter when porting a programming project from Microsoft Visual Studio's C/C++ compilers to CodeWarrior. Specifically, this chapter covers differences and potential problems (and solutions) you will encounter when porting a project originally developed for the Win32/x86 platform.

This chapter has these sections:

- C/C++ Compiler Differences
- C/C++ Library Differences

C/C++ Compiler Differences

Notable differences between CodeWarrior C and C++ compilers and the Microsoft C and C++ compilers are listed here:

- Conforming to the ANSI/ISO C and C++ Standards
- Relaxed Pointer Type Rules
- RTTI
- Exception Handling
- Name Mangling
- IEEE Floating Point Standards

Conforming to the ANSI/ISO C and C++ Standards

When the **ANSI Strict** option in the **C/C++ Language** settings panel is enabled, all non-standard language extensions are disabled.

Unless you are writing software that must be portable to any ANSI C/C++ platform, you probably should leave ANSI Strict option turned off.

Refer to the *C Compilers Reference* for complete information on the **ANSI Strict** option.

These language extensions include:

- array sizes of 0 and empty arrays as trailing structure members
- multiple identical `typedef`s
- using a `void` return type for `main()`
- unsigned enumeration types
- bitfields that are not `int`-sized
- anonymous unions
- computed goto labels
- GNU-style temporary initialization casts
- `asm()`-form inline assembly
- unnamed arguments
- pointer to integer conversions
- C++-style single-line comments in C source code (“`//`”)
- `long long` constants with “`i64`” suffix
- double constants with a “`d`” suffix
- binary constants with a “`0b`” prefix
- # on line by itself
- ignored tokens after `#else` and `#endif`
- supporting `#warning` and `#ident` preprocessor directives

Relaxed Pointer Type Rules

When you turn on the **Relaxed Pointer Type Rules** option in the **C/C++ Language** settings panel, CodeWarrior C overlooks the normal type checking it does when assigning from one pointer to another. The compiler does not warn or give an error message in cases where you mix pointers to different types. For example, when this option is on, the compiler will allow code like this:

```
struct foo *mary;
char *bob = mary;
```

This option should be avoided when possible.

This option has no effect on C++. When compiling C++ source code, the compiler differentiates pointer types even if this option is on.

This option exists to make it easier to port old pre-ANSI C code that assumed that any pointer had the same representation as any other pointer. C source code for Microsoft Windows that uses generic handles (`HANDLE`) instead of specific types of handles (for example, `HWND`) might not compile with this option turned off.

RTTI

CodeWarrior currently does not support Microsoft-compatible C++ runtime type information (RTTI). CodeWarrior does support ANSI/ISO C++ RTTI, but you cannot use `typeof()` or `dynamic_cast<>` on objects compiled by Microsoft C++.

Exception Handling

CodeWarrior provides full support for Microsoft-compatible C++ exception handling.

Name Mangling

CodeWarrior C++'s name mangling on templates is incompatible with Microsoft C++'s mangling when **ARM Conformance** is turned off in the **C/C++ Language** settings panel. With **ARM Conformance** off, CodeWarrior can support the differentiation between template and non-template functions that do not include the template type in their parameter lists.

For example:

```
template <class T> void foo (int a);
void foo (int a);
```

Using the Microsoft name mangling scheme, these two functions are mangled identically, although, in your code, you could differentiate them by:

```
foo<int> (42);
foo (42);
```

IEEE Floating Point Standards

CodeWarrior C/C++ follows the IEEE standards regarding the outcome of comparison operators and `NaN` (not a number). These are all considered unordered, so any comparison involving `NaN` will be false, except for the not equals comparison (!=) which returns true.

Microsoft C++ version 5 and Microsoft C++ version 6 do not generate instructions to check the unordered flag on comparisons, so the result of the comparison may vary depending on the exact encoding of the `NaN`s involved in the operation.

CodeWarrior C/C++ compilers for x86 and Microsoft C++ correctly handle propagation of `NaN`s through arithmetic operations, as that is handled by the floating point processor. For any operation, if one or both of the operands is a `NaN`, the result will also be a `NaN`.

If you have problems with this discrepancy in your code, you may want to enable the processor exception on `NaN` generation, but most code should never have to deal with these values.

To do so, you can add this code fragment to your project, with a call to `enable_nan_exception()` sometime before you expect the `NaN` to be generated. You should then enable the **Float Invalid Op** exception in the **x86 Exception** debugging settings panel.

```
#include <fenv.h>
void enable_nan_exception(void)
{
   fenv_t fe;
   // this should turn on exceptions
   // on NaN generation
   fegetenv(&fe);
   fe = fe & ~FE_INVALID;
   fesetenv(&fe);
}
```

Inline Assembler

Microsoft uses a syntax for the inline assembler in their C/C++ compiler similar to, but not exactly like MASM. Since this is barely documented, the CodeWarrior assembler attempts to duplicate the observed behavior of the Microsoft assembler. Note that there may be some cases where it will not detect an error.

Some of the inline assembler support includes:

- Both CodeWarrior C++ and Microsoft C++ use Intel syntax for their inline assembler.
- The Microsoft assembler treats labels as case insensitive, while CodeWarrior requires an exact match on case.
- Directives not supported in the x86 assembler: `EVEN`.
- CodeWarrior supports the `SIZE` keyword in assembly expressions for getting the size (in bytes) of an object. CodeWarrior does not support `LENGTH` or `TYPE`.
- CodeWarrior supports the `ALIGN`, `DB`, `DW`, and `DD` directives. CodeWarrior supports `EMIT` rather than `_emit` for directly inserting bytes into the assembly code.
- CodeWarrior ignores the `SHORT` modifier on jump instructions because it generates short jumps, if possible, by default. Microsoft uses `SHORT` as a flag to generate a short jump if possible.
- The assembler for x86 does not always correctly determine the size of file-scope static objects, especially when they are declared as arrays. If you are referring to them from assembly, you should explicitly name a size, for example:

  ```
  mov dword ptr [foo], eax
  ```

  is preferred over:

  ```
  mov [foo], eax
  ```

  as the second form may generate the wrong instruction. This has been fixed in later versions of the assembler.
- The CodeWarrior assembler does not accept suffix notation hexadecimal numbers. `0x1AE3` is allowed, but `1AE3h` is not.

C/C++ Library Differences

The `extras.c` file (part of MSL for Intel platforms) defines the following functions which are equivalent to functions by the same name in Microsoft's library. These functions include:

`_chdrive()`	`_heapmin()`	`_stricmp()`
`_strnicmp()`	`_strrev()`	`_wstrrev()`
`_strdup()`	`_strupr()`	`_strdate()`
`_itoa()`	`_itow()`	`_ultoa()`
`_fullpath()`	`_alloca()`	`_makepath()`
`_searchenv()`	`_getdiskfree()`	`_getdcwd()`
`_getdrive()`	`_getdrives()`	`_strlwr()`
`_splitpath()`	`_wtoi()`	`_wcslwr()`
`_wcsupr()`	`_wcsdup()`	`_wcsicmp()`
`_wcsnicmp()`	`_wcsrev()`	`_wcsset()`
`_wcsnset()`	`_gcvt()`	

CodeWarrior also supports many of the C9X standard's new floating point functions. In some cases, there are C9X equivalents for Microsoft library functions, although their interfaces may not be the same. The MS FPU control functions `_clear87()`, `_clearfp()`, `_control87()`, `_controlfp()`, `_status87()`, `_fpreset()`, and `_fpieee_ flt()` are supported by functions declared in the C9X `fenv.h` header file. CodeWarrior does not implement equivalents of `_chgsign()` and the Bessel functions.

Some of the functions supported are listed in Table 3.1.

Table 3.1 Microsoft functions and their ANSI/ISO equivalents

This Microsoft function…	is equivalent to this ANSI/ISO C9X function
`_finite()`	`isfinite()`
`_hypot()`	`hypot()`

This Microsoft function…	is equivalent to this ANSI/ISO C9X function
`_isnan()`	`isnan()`
`_copysign()`	`copysign()`
`_nextafter()`	`nextafter()`
`_scalb()`	`scalb()`
`_fpclass()`	`fpclassify()`

Index