MrC[pp] Plugins For Metrowerks CodeWarrior® Pro5

2/28/00

Introduction

The MPW MrC and MrCpp (hereinafter referred to as "MrC[pp]") compilers are available as plugins for the PowerPC versions of the Metrowerks CodeWarrior Pro5[1]. They are integrated with the IDE environment. Thus the standard IDE mechanics of setting up projects and preferences equally apply to the MrC[pp] plugin.

Installation

Eight files are required to install MrC[pp]; all of them go into the "CodeWarrior Plugins" folder which is located within the "Metrowerks CodeWarrior" folder. You have the choice of placing the entire "MrC[pp] Plugins" folder into the "CodeWarrior Plugins" folder or distributing the contents of "MrC[pp] Plugins" into the corresponding "Compilers" and "Preference Panels" folders within the "CodeWarrior Plugins" folder. The IDE (recursively) looks at all folders within the "CodeWarrior Plugins" folder so it doesn't matter how you organize these files.

The eight files distributed in the "MrC[pp] Plugins" folder are:

MrC[pp] Plugins	
Compilers	This folder contains the plugins and compilers
MrC	Actual MrC plugin
MrCpp	Actual MrCpp plugin
MrC.C.PPC.shlb	Called by MrC[pp] for C compilations
MrC.CPP.PPC.shlb	Called by MrC[pp] for C++ compilations
Preference Panels	This folder contains the 4 preference panels
1 Language Settings	
2 Code Generation	
3 Warnings	
4 Additional Options	

MrC will dynamically load MrC.C.PPC.shlb for C compilations while MrCpp will load MrC.CPP.PPC.shlb for C++ compilations. The four preference panels are described in the next section. The panels will appear in the IDE's Target Setting Panels list under their own unique category called "MrC[pp] Preferences".

You should install these files *before* you execute the Metrowerks IDE.

A folder of stationary called "MrC[pp]" is also supplied in the "(MrC[pp] Project Stationery)" folder. The "MrC[pp]" folder contains three standard example setups; two for MrC in the "Standard Console" folder and one for MrCpp in the "MPW Tool" folder. Drop the entire "MrC[pp]" folder into the "(Project Stationery)" folder inside the Metrowerks CodeWarrior folder (do not drop the folder containing "MrC[pp]", i.e., do not drop "(MrC[pp] Project Stationery)". The stationary is organized as follows:

[1] These plugins are designed for CWPro4 and 5 only. They will not work in anything earlier nor are they guaranteed to work in any future CodeWarrior releases. Note however that AltiVec™ cannot be used in CWPro4 with the MrC[pp] plugins due to library incompatibilities.

MrC[pp]	Drop this folder into “(Project Stationery)”
MPW Tool	Example using the MPW libraries
MrCpp MPW Tool	Hello World as an MPW tool
Standard Console	Two C examples that use SIOUX
MrC Std C Console PPC	Standard Hello World
MrC Std C Console Vector	Example using AltiVec™ (see below)

“MrC Std C Console PPC” is the equivalent to Metrowerks’ “Std C Console PPC” but using MrC instead of Metrowerk’s C/C++ compiler.

The “MrCpp MPW Tool” stationary illustrates how to build a C++ program with MrCpp. As discussed later, you *must* use the MPW libraries and headers when using MrCpp (C++). This stationary illustrates this and builds an MPW tool. You will have to change the Preferences Access Paths to the MPW “Interfaces&Libraries” folder to wherever you have them on your system.

“MrC Std C Console Vector” is similar to “MrC Std C Console PPC” but instead of a “hello world” example it illustrates the use of a few AltiVec vector operations and printf statements. CWPro 5 supports AltiVec and it’s library support is compatible with MrC[pp]’s.

Setting the IDE Preferences to Use MrC[pp]

In order to use MrC[pp] for your project source files you must define the File Mapping preferences (unless you use the provided stationery). The IDE Compiler popup menu will now have additional choices allowing you to select MrC or MrCpp for each desired source extension.

The Target Panel Settings will show a “MrC[pp] Preferences” category with four preference panels listed under it. Each of these is discussed below.

- Language Settings - All the C/C++ language settings and file information

Map newlines to CR	Controls interpretation of `'\n'` and `'\r'` depending on which libraries are being used, i.e., MPW’s or Metrowerks’. For MPW tools, this box must be checked.
Enums Always Int	Use minimum-sized or int-sized enums.
Use Unsigned Chars	Signed or unsigned chars.
Allow alloca() Intrinsic	Recognize alloca() as a built-in function.
Enable Exceptions	Exception processing. In C, only the generated tables are produced to allow pass through from a C++ caller.
Support long long	Recognize ‘long long’ as a predefined data type.

™ AltiVec is a registered trademark of Motorola, Inc.

AltiVec is supported in this release in accordance with the Motorola AltiVec Programming Model document. MrC[pp]’s implementation of the Model is covered in a separate document ("AltiVec Support in MrC[pp]").

ANSI	Controls levels of ANSI conformance, from approximately K&R style up to strict ANSI conformance.
Enable bool support	The ‘bool’ data type is supported (C++).
Direct to SOM	Direct to SOM is supported (C++).
ANSI for-stmt scoping	A variable declared in a for-statement as a iterator has its scope limited to that for statement (C++).
Enable RTTI	Enable RTTI support (normally enabled, C++).
Template Access	Controls scoping of instantiated templates (C++).
Relaxed Type Checking	Controls whether strict type checking is performed (C).
Require Funct ion Prototypes	Prototypes are required for all non-static functions (C).
Prefix File	Specify a header file or precompiled header file to be included with every compilation.
Save intermediate file	When reporting bugs it is sometimes necessary to supply the compiler’s intermediate file. Turing this option on will let that happen. Choosing the ASCII form will cause the intermediate file to be generated into the same directory as the source file. For file “foo.c” the intermediate file will be “foo.c.n”. When the checkbox is off a binary form of intermediate file is generated (which could also be saved) into the system “Temporary Items” folder.
Include headers only once	This is the default. Turing this option off allows the header to be read more than once.

• Code Generation - All options related to code generation

Generate Code	When this is *un*checked no code will be generated and all other Code Generation options are disabled. This can be used for syntax checking only but normally you should just leave it checked.
Target Processor	Use code scheduling for the specified processor. Note that when AltiVec is specified a variant of the 603 is always used regardless of the Target Processor setting.
Struct Alignment	Specify the default alignment for struct data.
Inlining Level	Sets complexity limit on functions to be inlined.
Optimization level	Specify level or form of code optimization. With optimization for size you get some additional capability to enable warnings for variables that are used before being set.

	You also get them for the speed optimizations but additionally you can limit certain kinds of speed optimizations.
All Strings Unique	Force all string constants to be unique.
Don’t se FMADD and FMSUB	Generation of floating point “maf” (multiple-add-fused) instructions. Default is to generate these instructions.
Pool Strings By Function	String constants are associated with their functions instead of the compilation unit.
Strict IEEE FP Adherence	Control strict adherence to IEEE floating point semantics.
64-bit Long Doubles	Size of long doubles is either 128 bits (default) or 64 bits. The default is 128-bit long doubles.
ASLM Compatibility	Support of RTTI and exceptions required a change in v-table format. This option exists for ASLM compatibility to cause generation of the “old” v-tables.
AltiVec Vector Support	Specify whether AltiVec vector instructions are to be generated.
Type Ptr Alias Analysis ANSI Ptr Alias Analysis Address Ptr Alias Analysis	These control various pointer alias analysis optimizations. Note that only certain combinations of these are permitted as a function of the specified optimization level.
Symbol Generation	Generate symbol information in the object file. You may control the generation of certain categories of information as specified by the checkboxes. Note that checking Symbol Generation turns off all code optimization.
Generate Traceback Tables	Allow the generation of traceback tables. These tables are also recognized by MacsBug. Optionally you can select that only functions specified by the traceback, export, or outofline pragmas be candidates for traceback table generation.

• Warnings & Errors - Controls warning to be reported

MrC[pp] has many warnings. So this preference panel is organized to specify which warnings you *don’t* want as opposed to those you do. Because there are so many possible warnings, the Warnings & Errors panel is actually made up of two panels with a button to allow you to switch between the two. There is a panel of general C and C++ warnings and a panel of warnings that are specific to C++. Aside from the warnings checkboxes, all other preferences in these panels are the same.

Select Warnings To Disable	Normally this is checked to allow you specify specific warnings that are *not* to be reported. Unchecking this box is the same as checking all the boxes, i.e., it disables all the warnings.
Treat All Warnings as Errors	Warnings are considered as errors.

Show All Errors	Continue reporting errors for a compilation unit past the normal limit (4).
Warning checkboxes	Check which warnings you want to suppress from being reported. Note, the warning numbers are displayed in place of the messages if the option key is pressed and as long as it stays pressed.

- Additional Options - Miscellaneous options

  This panel allows you to specify additional options not provided for in the other preference panels (for example the -xi option). This will also be used for new compiler options that don't as yet have preference specifications in the other panels. Options supplied here can be for either or both MrC and MrCpp. Only the options appropriate to the compiler being used are processed. These options always override those specified in the other panels.

  The options have the exact same format as those specified on an MPW Shell command line, i.e., -option, where option is a valid command line option. This may be followed by additional option parameters if applicable. Some useful options that could be specified here are:

-p	Display progress info to a "stderr". A window is displayed at the end of compilation called "stderr" showing the pathnames of all the included files. This might be useful if you want to make sure include files are coming from where you thing they should.
-help	Display all command line options. The "stderr" window shows all the available command line options. Many of these should not be specified because they either do not apply to the plugins (e.g., -load, -dump) or have options specified in the other panels.

General Restrictions and Considerations

The following are some general restrictions and/or considerations for using MrC[pp] plugins.

1. While the MPW versions of MrC[pp] are shipped as "fat" compilers, only the PowerPC version of the plugin is supported.

2. There are incompatibilities with the object models supported by Apple and Metrowerks (e.g., v-table formats, calling static constructors and destructors). You must **not** use the Metrowerks headers and libraries when using the MrCpp (C++) plugin. Instead you must use the MPW libraries and headers. When you do this with a C++ compilation, you also need to change the name of the Main entry point in the "PPC Linker" preference panel to `__cplusstart` instead of `__start`.

3. Exceptions are *not* supported in the MrCpp plugin. This is due to the fact that the Metrowerks linker does not support the exception tables produced by MrCpp.

3. All compilers have their own "idiosyncrasies" and no two are quite the same! Unless you

follow strict portability conventions you will find this out. Such things as pragmas, intrinsics, predefined macros, errors and warnings are different. Also, not all of Metrowerks' `__option` keywords are recognized by MrC[pp]. If you attempt to use MrC[pp] with the Metrowerks headers, or you are trying to port a file that was previously compiled with the Metrowerks compiler, you should define the macros those headers and sources expect and/or you may have to modify some of the Metrowerks headers. The most important macro is probably `__MWERKS__`. You'll have to define a prefix file containing its definition. If you use any of the other Metrowerks-specific predefineds, you'll have to define those as well. Metrowerks intrinsics which have no MrC[pp] counterpart have no work around. Note, the standard macro indicating a MrC[pp] compilation is `__MRC__`.

4. If you link with the MPW libraries, be sure to check the "Map newlines to CR" Language Settings checkbox. The MPW libraries assume `'\n'` is 0x0D.

5. The MrC[pp] compilers generate XCOFF object files. These are stored in the project until link time. If the project is sufficiently large, i.e., many files, you may see the code and data sizes change during the link. This is due to the fact that the linker is converting the stored XCOFF into IDE objects as part of the link process.

6. As just mentioned, each MrC[pp] XCOFF object file stored in the project is converted to the IDE-required format as part of the linking process. When the Code Generation "Symbol Generation" checkbox is checked, MrC[pp] will generate its debugging information in the form required by XCOFF. Neither XCOFF nor the conversion process necessarily support all the possible debugging functions supported for the Metrowerks compilers. Thus not all the functionality supported by the Metrowerks debugger is possible.

7. Precompiled headers generated by MrC[pp] are not compatible with those generated by Metrowerks (thus you cannot use the standard Metrowerks MacHeaders.h). Further, MrC C precompiled headers are not compatible with MrCpp C++ precompiled headers.

8. The MrC[pp] compilers use temporary (MultiFinder) memory. Specifically, they allocate the largest unused block (approximately the value displayed in the "About This Macintosh") minus a third of that (with a more reasonable limit if VM is being used). It also requires a minimum of 2 megabytes. If the temporary memory allocation fails, it allocates from the Metrowerks application heap (again requiring a minimum of 2 megabytes). In general, you should **not** allocate a large partition to the IDE since that will reduce the available temporary memory.

9. Because there are two plugins, MrC and MrCpp, you can explicitly set the File Mappings preferences as to which dialect (C or C++) to use for each file name extension. This differs from Metrowerks where the extension implies the dialect and the "C/C++ Language" preference "Activate C++ Compiler" checkbox acts as an override to force C++.

10. This version of the plugin will not work in any other release other than CWPro4 and Pro5.

Version-Specific Release Notes

The following apply to changes and/or additions to only the plugins. Changes to the MPW versions also apply here. See the MPW MrC[pp] release notes for further details.

4.1.0a8 (7/27/99)

1. The MrC[pp] plugins now support the keyword `__builtin_align` defined in Metrowerks' cstdlib. So the #define mentioned for 4.1.0a7 below is now unnecessary.

2. All other "problems" described for a7 with Pro5 still exist since they are up to Metrowerks to fix!

4.1.0a7 (7/8/99)

1. Changed "Code Generation" plugins preference checkbox that controls generation of FMADD & FMSUB instructions. The default for generating these instructions is "on" but the checkbox showed it as "off". The wording on the checkbox has been appropriately changed.

   To be sure of the setting in a project's preferences, users should make sure of the setting in the MrC[pp] Code Generation preference panel by checking it appropriately and doing a save.

2. Fixed `__option(fp_contract)`, `__option(maf)` to return the true setting of the of `-fp_cntract` and `-maf` command line options. They were returning the inverse of the setting.

Note that Codewarrior Pro5 was released shortly after the final build of the a7 compilers. Thus there is no guarantee the plugins will work properly in Pro5 even though the panels and compilers *appear* to "work" in some preliminary testing in Pro5. What is known from this preliminary testing is as follows:

- Pro5 supports AltiVec. Therefore the AltiVec.MSL4.Override.lib and substitute headers (csetjmp, cstdarg, and cstdlib) are not needed to use AltiVec in Pro5. However, Metrowerks decided to change the way cstdarg was originally defined for AltiVec and therefore will not compile using the MrC[pp] plugins. In order to use the Metrowerks cstdarg you need to add the following #define to their cstdarg.

  ```
  #define __builtin_align __va_align__
  ```

- Metrowerks appearently has decided to define their own C++ standards and has some non-standard contructs in some of their headers. Basically the construct,

  ```
  extern "C" {
    T foo[];
  }
  ```

  is illegal in C++ and MrCpp will report it as such. This occurs in the headers cctype, cfloat, cmath, cstdio, cwctype, and cmath.macos.h. Note that this incompatibility shouldn't be a problem since you need to use the MPW headers for MrCpp anyhow.

- cstdio has an unterminated comment in the line where stdio_tofiles is defined. MrC[pp] will report this even though the statement is being skipped.

4.1.0a1 (11/16/98)

No plugin-specific changes. See “MrC 4.1.0a1 Release Notes” for compiler changes.

4.1.0d4 (10/5/98)

1. Plugins updated for CWPro3 and 4. Preference panels completely redone.
2. New `__option` keywords added to minimize some of the Metrowerks header incompatibilities.

mpwc_newline	Same as “mapcr” (1 when “Map newlines to CR” is *un*checked).
preprocess	1 if syntax checking only (set when Generate Code is unchecked).
traceback	Set when the Generate Traceback Tables is checked.
wchar_type	0 for C, 1 for C++
little_endian	Always 0 (this keyword is *only* recognized in the plugins)

3.0.0d2 (10/17/96)

1. The MrC[pp] panel now has an additional “CommandLine” text entry option. This is an “escape” mechanism that can be used to supply additional MPW command line options to the compilers that are not currently supplied as standard panel items.
2. Because of the additional “CommandLine” panel item, the preference data for a MrC[pp] plugin project has changed. It is larger and items have moved. This means that if you take a previously existing MrC[pp] project from CodeWarrior 9 and have CodeWarrior 10 update it, the preference items will be in the wrong place! The most obvious indication of this is that a prefix file will appear in the “CommandLine” text entry.

   Since the preference layout has changed in both content and size, the only way to establish the new preferences in a converted project is to reset the preferences to “Factory Settings”. This means, of course, you will have to reset your original preference items manually.
3. The “MrC[pp] CodeGen” panel Optimizations have been changed. You now have more control over the degree of compiler optimization than in previous versions. Here’s a summary of the optimization levels you get in terms of the equivalent MPW command line options:

```
“Optimizations” not checked         ---> -opt off
“Optimizations” checked only        ---> -opt local
“Optimizations” checked and...
   “Optimize for size”              ---> -opt size
   “Optimize for speed”             ---> -opt speed
      and “Loop unrolling”          ---> -opt speed,unroll
      and/or “No copy propagation”  ---> -opt speed,norep
      and/or “No interproc opt.”    ---> -opt speed,nointer
      and/or “Warn uninit”          ---> -opt speed,warn_uninit
      and/or “Warn maybe uninit”    ---> -opt speed,warn_maybe_uninit
```

Note, this is only temporary. We plan to redo all the MrC[pp] preference panels when time permits. Also, as in the case of the new “CommandLine” panel item in the MrC[pp] panel, these changes to the MrC[pp] CodeGen panel will cause incompatibilities with converted projects. You should revert to the factory settings then reset you MrC[pp] CodeGen items.

4. The prefix file in MrC[pp] now has the identical behavior as in the Metrowerks compilers. Thus a prefix file may either be a text file *or* a precompiled header file. There will be no more prompting for the precompiled header name if the file extension is flagged to automatically precompile in the target preferences (for example, as illustrated by the .pch and .pch++ target extensions). As in the Metrowerks compilers you will still be prompted if the target extension is not flagged to be automatically precompiled.

   Note, when a file is precompiled, either automatically or explicitly from the menu, the prefix file is *not* used.

5. Two new pragmas are supported to supply the name of a precompiled header:

   ```
   #pragma dump "filename"
   #pragma precompile_target "filename"
   ```

   The “dump” pragma is identical to the “precompile_target” pragma. The former is the style used by the “old” MPW C compiler while the latter is for Metrowerks compatibility.

   The filename supplied by these pragmas will be the file that will receive the precompiled header. If the filename is supplied as a null string, or the pragma is not present in the file, the name of the source file is used with the extension removed. If more than one pragma appears in the source, the one closest to the end of the file is used. If the filename is a partial path name, the directory used is the same as the source file. A full pathname of course specifically indicates where the precompiled header is to “go”.

6. #include has been extended to accept a precompiled header. However, this #include must be the first statement in the source file (or prefix file if it is being used to do the #include). In other words no definitions, declarations, or #define may precede this #include. This also implies there can only one such #include of a precompiled header in the source.

7. A `__option(precompile)` has been defined to indicate that a precompiled header is being built.

   Note, the plugin release is, at the time this is being written, a more current version than what is going out on ETO 22. There are some additional `__option` keywords not yet released for ETO 22 but are available in the plugins. These are summarized here but this list will be moved to the standard MrC[pp] release notes beyond ETO 22.

`__option(RTTI)`	`-rtti on`
`__option(fp_contract)`	`-fp_contract on`
`__option(maf)`	Same as `__option(fp_contract)`
`__option(direct_to_SOM)`	`-som`
`__option(SOMCallOptimization)`	`#pragma SOMCallOptimization on` in effect.
`__option(SOMCheckEnvironment)`	`#pragma SOMCheckEnvironment on` in effect.
`__option(ldsize128)`	`-ldsize 128`

`__option(ANSI_strict)`   `-ansi strict` in effect.

8. Another feature of the plugins, because they are later than the ETO 22 release, is a change to the storage classes of instantiated templates. Like the new `__option` keywords, this summary will become part of the post-ETO 22 release notes for the MrC[pp] compilers.

   Up until now, the default storage class for instantiated templates has been ‘static’. For example,

   ```
   template <class T> T F(T x) {return x;}

   main()
   {
      F(1);
   }
   ```

   The compiler would instantiate an F(int) as if it were declared ‘static’,

   ```
   static int F(int x) {return x;}
   ```

   If you wanted to make this instantiation public (i.e., not ‘static’), then you needed to supply a `#pragma template_access public` (or `-xa public` if you are using an MPW command line). But if you did that, the compiler would not instantiate the function unless you explicitly told it to using a `#pragma template` (or -xi from the command line), i.e.,

   ```
   #pragma template_access public
   #pragma template F(int)
   ```

   This has all been changed! Now the default access is public and the instantiations *will* occur. Previous users of the MrC[pp] compilers who still need their instantiations private will have to supply a `#pragma template_access static`. This is a major change to previous behavior. But it makes MrC[pp] more consistent with other compilers.

   Note, a unique feature of the MrC[pp] compilers (which is not new) is to specify the `template_access` as `extern`. In this mode (and now in this mode only) the instantiations will not occur and appropriate ‘extern’ declarations will be generated in their place. This allows you to only have a single copy of the insantiations with all others referring to them via extern declarations.