CodeWarrior®
Error Reference

metrowerks

Because of last-minute changes to CodeWarrior, some parts of in this manual may be out of date. Please read all the Release Notes files that come with CodeWarrior to get important last minute information.

Copyright

How to Contact Metrowerks

U. S. A. and international:	Metrowerks Corporation 2201 Donley Drive, suite 310, Austin, TX 78758 U. S. A.
Canada:	Metrowerks Inc. 1500 du College, suite 300, Ville St-Laurent, QC Canada H4L 5G6
Metrowerks Mail Order:	voice: 800 377-5416 fax: 512 873-4901
World Wide Web:	http://www.metrowerks.com
Registration information:	register@metrowerks.com
Technical support:	support@metrowerks.com
Sales, marketing, & licensing:	sales@metrowerks.com
AppleLink:	METROWERKS
America OnLine:	keyword: Metrowerks
Compuserve:	goto Metrowerks

Table of Contents

1

Introduction

This manual lists the errors you may encounter from the CodeWarrior compilers and linkers.

Overview of the Error Reference

When you compile and link code, CodeWarrior may discover problems. If there is a problem, the compiler or linker posts an error in the Message window. This manual discusses each error, what it means, and in many cases provides you with suggestions for correcting the error.

The chapters in this manual describe the errors you may encounter from each compiler and linker. They are:

- "C/C++ Compiler Errors"
- "Linker Errors"
- "Magic Cap Linker Errors"
- "Magic Cap Precompiler Errors"
- "Pascal Compiler Errors"

Conventions Used in This Manual

The following list describes the font conventions and structure used in this reference manual.

<Error Message>

This section following the error message explains the nature of the error and its possible causes.

')' expected

An example of how an error or warning message would appear.

The compiler did not find a right parenthesis where it expected to find one.

Listing 1.1 sample source code

```
In some cases, sample source code is provided
that triggers the error message.
```

Fix: Some error messages include a suggestion about how the error could be fixed.

See Also Some error messages include a reference where more information can be found. This section often points out a Metrowerks CodeWarrior feature that can be used to detect or stop the error from being generated.

Preferences affect errors

For more information on the panels which affect error recognition, consult the CodeWarrior Users Guide

Language preferences not only affect the way the compiler translates the source code in your program files, these preferences also affect which source code is flagged as errors and which source code compiles. In many cases, the appearance of errors depend heavily on the selected Language preferences. Where applicable, the description of an error will include the name of the checkbox that, when selected, screens for a particular error.

To change the language options used by the compiler, select Preferences from the Edit menu and click the Language icon in the Preferences dialog box.

2

C/C++ Compiler Error Messages

This chapter gives an alphabetical list of the compiler errors which may be encountered while using Metrowerks CodeWarrior compilers for the PowerPC-based, 68K-based Mac OS, Win32/x86 and Windows NT application code generators.

C/C++ Compiler Errors

In this list, errors with variable initial text (such as a class or function name) come first. Errors beginning with a non-alphabetic symbol character come next. After that, errors are listed alphabetically.

NOTE: *The description of some C++ errors contains a reference to Margaret A Ellis and Bjarne Stroustrup's The Annotated C++ Reference Manual, Addison-Wesley, Reading, MA, 1990. These page references are indicated by the abbreviation ARM, followed by the page number and section. For example: ARM p.202, 10.1.1 Ambiguities.*

Symbol Names

These are C/C++ compiler error messages that begin with a symbol name, the name of a variable or function.

class is not a SOM class

You are using one of the SOM pragmas within the declaration of a class that isn't a SOM class. The pragmas `SOMReleaseOrder`, `SOMClassVersion`, `SOMMetaClass`, and `SOMCallStyle` may appear only within the declaration of the SOM class they apply to.

Fix Be sure the pragma appears within the declaration of the SOM class it applies to. Don't use these pragmas with classes that are not descended from SOMObject.

'_variable' is not a struct/union/class member

An item referenced as a member/method of a struct, union, or class is not defined as being a member/method. For example, in Listing 2.1 below, `theColors.color` references a member, `var`, that does not belong to the `struct ColorValues`.

Listing 2.1 *Not a struct/union/class member*

```
typedef struct
{
  short  seq;          // var is not defined in
  short  group;        // this struct
} ColorValues;

ColorValues    theColors;
...
...
  theColors.color     = 1;    // error: see above
```

Fix Check the structure, union, or class in question. This error may be caused by a simple spelling mistake. In C++ this error often happens when a class member function or constructor's arguments do not match the prototype. If this is not the case, either change the item referenced or modify the structure, union, or class.

Punctuation

These are C/C++ compiler error messages that begin with punctuation marks.

'func1' hides inherited virtual function 'func2'

You declared a non-virtual member function that hides a virtual function in a superclass. One function hides another if it has the same name but a different argument types. For example:

```
class A {
  public:
    virtual void f(int);
    virtual void g(int);
};

class B: public A {
  public:
    void f(char);        // WARNING:
                         //  Hides A::f(int)
    virtual void g(int); // OK:
                         //  Overrides A::g(int)
};
```

This warning appears only if you turned on the **Hidden virtual functions** option in the C/C++ Warnings settings panel

Fix Turn off the **Hidden virtual functions** option or choose another name for one of the functions.

#if nesting overflow

This error occurs when the number of nested `#if` processor directives exceeds the maximum number allowed.

Fix To fix this error, study the logic behind your nested `#if`s. There's probably a way of dividing the large nested `#if` into a series of smaller nests.

#include nesting overflow

This error occurs when the number of nested `#includes` processor directives exceeds the maximum number allowed.

Fix To fix this error, study the logic behind your nested `#includes`. There's probably a way of dividing the large nested `#includes` into a series of smaller nests.

‘(’ expected

The compiler did not find a left parenthesis where it expected to find one.

Fix Use the **Balance** command to balance all left and right parenthesis.

‘)’ expected

The compiler did not find a right parenthesis where it expected to find one.

Fix Use the **Balance** command to balance all left and right parenthesis. This error may be caused by a syntax error in a previous statement.

To prevent this error while typing in source code, select the **Balance While Typing** checkbox in the Editor preference panel. When selected, this preference sounds an alert if an un-matching right parenthesis is typed.

See Also For more on **Balance While Typing**, consult the *CodeWarrior IDE User's Guide*.

NOTE: *This error may be caused by a syntax error or missing symbol in a previous statement.*

‘<’ expected

The compiler did not find a left angle bracket where it expected to find one.

Fix This error may be caused by a syntax error or missing symbol in a previous statement.

NOTE: *The Balance command does not check for angle brackets, ’<’ and ’>’.*

'>' expected

The compiler did not find a right angle bracket where it expected to find one. For example, Listing 2.2 gives an example of a missing right angle bracket.

Listing 2.2 '>' expected

```
template <class T> class Ca;
CA *aClass;                      //  error
```

Fix This error may be caused by a syntax error in a previous statement.

NOTE: *The Balance While Typing does not check for angle brackets, '<' and '>'.*

See Also For more on **Balance While Typing**, consult the *CodeWarrior User's Guide*.

',' expected

The compiler did not find a comma where it expected to find one.

Fix This error may be caused by a previous syntax error. For example, the compiler expects to find a comma in the call to `GetMenu()` in the Listing 2.3 below. Actually, to fix this error, a parenthesis must be added to end the function call to `GetMenu()`.

Listing 2.3 Comma expected

```
  gAppleMenu = GetMenu(APPLE_MENU_ID);
  gFileMenu  = GetMenu(FILE_MENU_ID;
      // error  ^
      //  compiler expected a comma because a right
      //  parenthesis is missing.
```

‘:’ expected

The compiler did not find a colon where it expected to find one. For example, in the `switch` statement in Listing 2.4, a colon is missing after `APPLE_MENU_ID`.

Listing 2.4 Colon expected

```
...
  switch(theMenu)
  {
    case APPLE_MENU_ID    // error:  missing ':'
      switch(theItem)
      {
        case ABOUT_ITEM :
          ...
```

Fix If the error is not apparent, an error in a previous statement may be causing the problem. Correct all previous errors first and re-compile.

‘;’ expected

The compiler did not find a semicolon where it expected to find one. For example, in Listing 2.5 below, a semicolon is missing after the function call `WindowInit()`.

Listing 2.5 Semicolon expected

```
...
  ToolBoxInit();
  WindowInit()    // ';' missing from this line
  MenuBarInit();
...
```

Fix If the error is not apparent, it is likely being caused by a previous error. Correct all previous errors first and recompile.

‘[’ expected

The compiler did not find a left bracket where it expected to find one.

Fix If the error is not apparent, it is likely being caused by a previous error. Correct all previous errors first and recompile.

To prevent this error while typing in source code, select the **Balance While Typing** checkbox in the Editor preferences panel. When selected, this preference allows you to balance brackets as you type them.

See Also For more on **Balance While Typing**, consult the *CodeWarrior IDE User’s Guide*.

‘]’ expected

The compiler did not find a right bracket where it expected to find one.

Fix If the error is not apparent, it is likely being caused by a previous error. Correct all previous errors first and recompile.

To prevent this error while typing in source code, select the **Balance While Typing** checkbox in the Editor preference panel. When selected, this preference allows you to balance brackets as you type them.

See Also For more on **Balance While Typing**, consult the *CodeWarrior IDE User’s Guide*.

‘{’ expected

The compiler did not find a left brace where it expected to find one.

Fix Use the **Balance** command to balance all left and right braces. This error may be caused by a syntax error in a previous statement.

‘}’ expected

The compiler did not find a right brace where it expected to find one.

Fix Use the **Balance** command to balance all left and right braces. This error may be caused by a syntax error in a previous statement.

To prevent this error while typing in source code, select the **Balance While Typing** checkbox in the Editor preference panel. When selected, this preference lets you balance braces as you type them.

‘&’ reference member ‘_var’ is not initialized

A reference member was not initialized. All reference types must be evaluated in the scope of the constructor. For example, Listing 2.6 below shows an un-initialized and a properly initialized reference.

Listing 2.6 Reference member not initialized

```
class caClass {
   private:
      int x;
   public
      const int &ref;
      caClass() {}  //  <-- no  initialization
 };
  // properly initialized reference
class caClass {
   private:
      int x;
   public:
      const int &ref;
      caClass():ref(x) {} //<-- now reference is
initialized
 };
```

A, B, C

These are error messages that begin with *A*, *B*, or *C*.

ambiguous access to class/struct/union member

The compiler signals this error when a reference to a class, struct, or union member is ambiguous.

You may get this message when calling a function that is defined in both a virtual base class and another base class with the same parameters. The CodeWarrior C++ compiler is more strict about this situation. You must use the fully qualified form to define which function you want to use.

See Also ARM p.202, 10.1.1 Ambiguities.

ambiguous access to overloaded function

This error is displayed when an ambiguous reference is made to an overloaded function. References to overloaded functions must be unambiguous. InListing 2.7, `funcA()` is overloaded with a default argument. When it is called in the `main()` function, the compiler does not understand which member function to use.

NOTE: *This error is also common using* `double`/`float` *or* `short`/`long` *arguments to overload a function.*

See Also ARM p.307, 13 Overloading.

Listing 2.7 Ambiguous access to overloaded function

```
class aClass
{
    int x;
  public:
      int funcA( ) { x = 2; return x; }
      int funcA(int y = 3)  { x = y; return x; }
};

main()
{
  aClass obj;

  obj.funcA();
  return 0;
}
```

assignment is not supported for SOM classes

You cannot use a class descended from SOMObject in an assignment operation, since SOM classes do not support copy constructors. For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SOMObjects Developer Toolkit* (IBM).

branch out of range

A branch destination in an assembly function is out of range, as in the branch call in Listing 2.8.

Listing 2.8 Branch out of range

```
...
bra.s 10000
...
```

call of non-function

An attempt was made to call a non-function. For example, Listing 2.9 below attempts to call the variable `i` as if it were a function.

Listing 2.9 Call of a non-function

```
main()
{

int i;
...
  i();    // error: “i” is not a function
...
}
```

Fix Check the non-function call in question. This error may be caused by a spelling mistake that attempts to call a similarly named non-function instead of the desired function.

cannot construct base class ‘_aClass’

This error appears when the base class *aClass* has no ctor initializer or default constructor.

cannot construct direct member ‘_aClass’

This compiler gives this error when the direct member *aClass* has no ctor initializer or default constructor.

cannot convert from ‘Type_A’ to ‘Type_B’

The compiler generates this error message when a type conversion was attempted without proper conversion constructors, or with incompatible types. The code in Listing 2.10 attempts to convert a type `long *` to an `int *`.

Listing 2.10 Cannot convert Type_A to Type_B

```
main()
{
 int  *ptr;
 ptr = new long;  // <-- Error  wrong type
 return 0;
}
```

cannot delete pointer to const

The compiler generates this error message when it encounters an attempt to delete a pointer to a const value. For example, in Listing 2.11 below an attempt is made to delete a pointer to a `const` type.

Listing 2.11 Attempt to delete a pointer to a const

```
main()
{
 const int y = 3;
 int const *ptr = &y;
 delete ptr; // <-- Error here
```

```
  return 0;
 }
```

cannot destroy const object

The compiler generates this error when an attempt to destroy a `const` object is encountered.

cannot instantiate '_obj'

The compiler generates this error when the template *_obj* cannot be instantiated because its definition is missing, such as the Listing 2.12 below.

Listing 2.12 Cannot instantiate

```
template <class T> class aClass;
template class aClass<int>;        //  error
```

cannot pass const/volatile data object to non-const/volatile member function

This compile error occurs when an attempt is made to pass a data object declared as a `const` to a member function that is not declared as `const`. The Listing 2.13 below generates this compiler error.

Listing 2.13 Cannot pass const data object to non-const member function

```
struct stType {
    void bar();             //  non-const member 
function
    void cbar() const;      //  const member 
function
};
...
    stType f;
    const stType cf;

    f.bar();    //  OK
    f.cbar();   //  OK
    cf.bar();   //  error
    cf.cbar();  //  OK
```

cannot throw class with ambiguous base class ‘cBase’

The class in the throw point has an ambiguous base class.

Fix Declare a virtual base class or eliminate any ambiguities to resolve this error message.

case constant defined more than once

A constant used in a switch statement is already in use. For example, in Listing 2.14 below, the constant `ABOUT_ITEM` is used more than once.

Fix Remove one of the constant labels.

Listing 2.14 Case constant defined more than once

```
...
    switch(theItem)
    {
      case ABOUT_ITEM :
        Alert(ABOUT_ALRT, NIL);
        break;
      case ABOUT_ITEM :
        Alert(ABOUT_ALRT, NIL);
        break;
...
```

‘catch’ expected const member ‘aVar’ is not initialized

This error message is generated when the compiler encounters a const member that was not initialized correctly.

Fix The const member must be initialized at the time of the object’s construction.

class has no default constructor

This error occurs when the compiler cannot construct a class because it has no default constructor, as shown in Listing 2.15.

Listing 2.15 Class has no default constructor

```
...
struct stType {
  stType(int);
};

stType f;     // error: no default constructor
...
```

class type expected

The compiler generates this error message when a class type was expected.

'const' or '&' variable needs initializer

You must initialize const or reference variables when you declare it. For example:

Listing 2.16 Const and reference variables need initializers

```
const int a = 1;  // OK
const int b;      // ERROR: const variable
                  //        needs initializer

int c;
int &rc = c;      // OK
int &rd;          // ERROR: reference variable
                  //        needs initializer
```

constness casted away

This error message is generated when an attempt to cast a `const` to a `volatile` type is encountered.

D, E, F

These are C/C++ compiler error messages that begin with *D*, *E*, or *F*.

data type is incomplete

The data type usually a class or structure is incomplete. "Incomplete class" errors usually happen when attempts are made to use classes that have been partially declared usually using "forward declarations".

See Also "illegal use of incomplete struct/union/class" on page 64

declaration syntax error

The compiler encountered a syntax error while trying to resolve a declaration.

Fix Examine the declaration in question. If the error is not apparent, it is likely being caused by a previous error. Correct all previous errors first and recompile.

declarator expected

The compiler expected to find a declaration, but found something else instead.

Fix Check the declaration in question. If the error is not apparent, it is likely being caused by a previous error. Correct all previous errors first and recompile.

default label used defined more than once

The compiler found more than one `default` label in the same `switch` statement. For example,Listing 2.17 below causes this error.

Listing 2.17 More than one `default:`

```
...
switch(...)
{
  default:;
  default:;  // only one default in switch !
}
```

Fix Remove one of the `default` labels.

derived function differs from virtual base function in return type only

The compiler generates this error when the return type of the derived function differs from the return type of a virtual base function. The Listing 2.18 provides an example.

Listing 2.18 Derived function differs from virtual base in return type only

```
class aClass { virtual int f(); }
class bar : aClass {
  void f();        //  error
}
```

division by 0

When a constant expression tries to divide by zero or use modulo zero, a division by 0 error is signaled.

end of line expected

This error can occur in many circumstances, and may be the result of another error on a previous line of code. For example, if you turn on the **ANSI Keywords Only** option in C/C++ Language settings panel, this error occurs when text follows the `#endif` directive. The ANSI standard specifies that only a comment can follow an `#endif` directive. Listing 2.19 below shows another example where more tokens are expected on a line.

Listing 2.19 More tokens expected on a line

```
#define    // error:  compiler expects more tokens
#if       //          on both lines.
...
...
```

Fix In the case of text rather than a comment following the `#endif` directive, deselect the **ANSI Keywords Only** checkbox in C/C++ Language settings panel, and recompile.

exception specification list mismatch

The exception specification lists for a function declaration and a function definition don't match, as shown in Listing 2.20.

Listing 2.20 exception specification list mismatch

```
void f() throw(int);
        // exception specification list mismatch
void f() throw(long)
{
}
```

exception handling does not work with 'direct destruction'

The compiler generates this error when you use C++'s exception handling and the **Enable C++ Exceptions** option in the C/C++ Language settings panel is off or the `direct_destruction` pragma is on.

Fix Turn on the **Enable C++ Exceptions** option in the C/C++ Language settings panel or turn off the pragma `direct_destruction`.

expression syntax error

The compiler generates this error when it encounters any kind of illegal expression syntax.

Fix If the error is not apparent, it is likely being caused by a previous error. Correct all previous errors first and recompile.

function already has a stackframe

This error occurs when the compiler finds more than one `fralloc` directive in an assembly function.

Fix Remove all but one of the `fralloc` directives from the assembly function.

function call '_func' does not match

The compiler generates this error when a call to a function does not match the expected arguments. An attempt to initialize an object without a proper matching constructor also generates this message.

NOTE: *This message is often a result of not terminating a class definition with an semicolon. If there is no semicolon, the compiler expects to find an object list. Listing 2.21 below demonstrates this problem.*

Listing 2.21 Function '_func' does not match

```
class A {
  public:
  A() {}
}  //<-- no semicolon, object list expected

main()
{
 A a;  // <-- error reported here
  return 0;
}
```

Fix Add a default constructor for your class. Also, check the previous defined class or structure, including the header file named prior to this error message, for a missing object list.

function call does not match prototype

A function call's arguments do not correspond to the function's prototype. Listing 2.22 shows that the call to `SetWaggle()` passes two arguments when the function prototype requires one.

Fix Match function call with function prototype. Select the function call and choose **Find Definition** from the **Search** menu to locate the function prototype definition.

Listing 2.22 Function call does not match prototype

```
long SetFoo(long foonum)
{
...
}
...
MyFoo = SetFoo(size, length);
  // error: two variables parameters in call to
  //        SetFoo, prototype has only one argument
```

function declaration hides inherited virtual function, class 'C: _A(x)' has more than one final overrider '_A(x)' and '_A(y)'

(Warning Message) The compiler generates this warning message when the declaration has hidden the virtualness of the function. The virtual function has been overloaded rather than overridden.This is only a warning because as in Listing 2.23 below it is legal (but dangerous).

Fix To resolve this warning ensure that all derived virtual functions have identical parameter lists as the base virtual function.

Listing 2.23 Function declaration hides inherited virtual function.

```
class X     { virtual void f(); };
class Y : X { void f(int); };
    // Y::f() hides X::f()
```

function defined ‘inline’ after being called

This error message is generated when a function is declared inline after it has already been called. In Listing 2.24 below the function is used in the class before the function definition where it is declared inline.

Fix Declare the function prototype to be inline.

Listing 2.24 *Function defined inline after being called*

```
int func( int x );

 class cA {
    int i;
    public:
      cA() { i = func( 3); }
};

 inline int func( int x ) { return x + 1; }
```

function has no initialized stackframe

This error occurs when the compiler encounters an assembly function whose stack frame as not been allocated using the `fralloc` directive.

Fix Modify your assembly function so is that it uses `fralloc`.

function has no prototype

A function is defined without a preceding prototype. This error occurs if the **Require Function Prototypes** checkbox, in the C/C++ Language settings panel, is selected.

Fix Either define a preceding prototype for the function in question, or deselect the Require function prototypes checkbox.

See Also For more on the **Require Function Prototypes** option, consult the *C, C++, and Assembly Language Reference*.

function nesting too complex

This error occurs when the compiler encounters a function that contains too many nested `{}` blocks.

Fix To fix this error, study the function in question. There's probably a way to divide the function into a series of dependent functions.

functions cannot return SOM classes

A function cannot return a class which is descended from SOMObject, since SOM classes do not support copy constructors. For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SOMObjects Developer Toolkit* (IBM).

functions cannot have SOM class arguments

A function cannot contain in its argument list any class which is descended from SOMObject, since SOM classes do not support copy constructors. For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SOMObjects Developer Toolkit* (IBM).

G, H, I

These are C/C++ compiler error messages that begin with *G*, *H*, or *I*.

global SOM class objects are not supported

If an object is a type descended from SOMObject, you cannot use it as a global variable. For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SOMObjects Developer Toolkit* (IBM).

identifier '_name' redeclared was declared as: 'a_type' now declared as: 'b_type'

The compiler issues this error when the source code attempts to redefine an identifier. For example, in Listing 2.25 below, the identifier `var` is declared as both a `long` and a `Point`.

NOTE: *This error is often the result of re-using the name of a Macintosh declared variable or function.*

Fix Change the name of one of the identifiers.

Listing 2.25 Identifier ‘var’ redeclared

```
  long    var;
  Double  temp;
  Point   var;
           // var has already been declared above

// Using a Macintosh Toolbox function name
// as a class name.

  class Random { public: void get(); };

  void Random::get() // <-- this produces an error
  { //... }
```

identifier expected

The compiler expected to find an identifier, but instead found another token. For example, in Listing 2.26 below , the compiler issues this error because `short` is placed where an identifier should be.

Fix If the error is not apparent, it is likely being caused by a missing symbol, such as a semicolon or comma, on a previous line. Correct all previous errors first and recompile.

Listing 2.26 Identifier expected

```
  long   Waggle;
  long  temp, short; // short is placed where an
                     // identifier should be
  Point  myPt;
```

illegal #pragma

The compiler found an unrecognized `#pragma` directive.

Fix A list of `#pragma`s recognized by Metrowerks C is listed in the *C, C++, and Assembly Language Reference*. Consult this list to make sure the `#pragma` you are using exists and is spelled correctly.

illegal ‘&’ reference

This error is issued when the compiler encounters an illegal & reference, as in Listing 2.27.

Listing 2.27 Illegal ‘&’ reference

```
...
int &&g;
// error: cannot dereference an integer twice
...
```

illegal constructor/destructor declaration

The compiler generates this error when an attempt to declare a constructor or destructor is done in an illegal manner. Listing 2.28 attempts to call the constructor after the object is already initialized.

Listing 2.28 Illegal constructor declaration

```
class cA {
  public:
     void cA() {}
};

cA A;

main() {
  A.cA();
  return 0;
}
```

illegal const/volatile '&' reference initialization

The compiler generates this error message when either a `const` or a `volatile` reference was improperly initialized.

Fix This is often the result of not initializing a reference during object construction.

illegal data in precompiled header

The compiler issued this error because the precompiled header contained improper data.

illegal exception specification

You used a exception specification where it isn't allowed, as shown in Listing 2.29.

Listing 2.29 *illegal exception specification*

```
typedef void (*f)() throw(int);
                    // cannot use spec in typedef
```

illegal explicit conversion from 'type_A' to 'type_B'

The compiler issued this error when an explicit conversion of one type to an improper type is encountered. Listing 2.30 attempts to convert a pointer to a class into a `long`.

Listing 2.30 *Illegal explicit conversion*

```
class D { };

 main()
 {
    long x;
    D d;
    x = (long)d;
    return 0;
 }
```

illegal ‘friend’ declaration

The compiler issues this error when it encounters an illegal declaration. For example,Listing 2.31 shows `friend` being illegally declared.

Listing 2.31 Illegal ‘friend’ declaration

```
int i;
class aClass {
  friend i;      // illegal 'friend' declaration
};
```

illegal ‘inline’ function definition

This error is given when a function that has already been referenced is defined as inline.

illegal ‘operator’ declaration

This error is displayed when the compiler finds an illegal operator declaration, as in Listing 2.32.

Listing 2.32 Illegal ‘operator’ declaration

```
...
int operator +(int,int,int);
...
```

illegal ‘class::constructor(argument)’ copy constructor

This error appears if a class constructor function is declared with an argument of the same class type, as in Listing 2.33.

Listing 2.33 Illegal class::constructor(argument) copy constructor

```
class aClass {
  aClass(aClass);    // error
  aClass(aClass&);   // OK
};
```

illegal access declaration

This error occurs when you attempt to incorrectly declare access to a base class member from a derived class. You can adjust the access to base class member by using the base class member's qualified-name, in a public or protected part of a derived class declaration. Listing 2.34 demonstrates this error.

See Also ARM p. 244, 11.3 Access Declaration.

Listing 2.34 Illegal access declaration.

```
class B {
    private:
  int a
    public:
  int b;
};

class D : private B {
    public:
  B::a;     // <-- Error illegal access declaration
  B::b;     // legal allows B::b to be used in
            //  an external function
  int x;
  void fx() { b = x; }
}Derived;

inline void fx() { Derived.b = 3; }
```

illegal access qualifier

This error is signaled when an illegal qualification is encountered. In Listing 2.35, this code is illegal because `foo` doesn't exist.

See Also ARM p. 112.

Listing 2.35 Illegal access qualifier

```
struct stType { enum efoo { nfoo } mfoo; };
...
  foo::xfoo;     // error: illegal qualified
```

illegal access to protected/private member

This error is signaled when a function attempts to access a private member. For example, in the Listing 2.36 `priv` is declared as `private`. Function `func()` attempts to access this member.

Listing 2.36 Illegal access to private member

```
class aClass { private: int priv; }
func() {
  aClass x;
  x.priv=0;
      //func() cannot access private member priv
}
```

illegal addressing mode

An assembly-language instruction attempts to use an addressing mode that is not possible with this instruction, as in Listing 2.37.

Listing 2.37 Illegal addressing mode

```
...
moveq  d0,d1
...
```

illegal argument list

This error appears when the compiler finds an illegal macro argument list, as in Listing 2.38.

Listing 2.38 Illegal argument list

```
#define macro(arg,,)
...
```

illegal array definition

The compiler gives this error when it encounters an array defined with a negative or zero subscript (also illegal array base type). Making the last member of a struct an empty array is a non-ANSI language extension that is not supported by Metrowerks CodeWarrior C/C++ as demonstrated in Listing 2.39.

Fix As a work around for the source in Listing 2.39 , the code should change to what's shown in Listing 2.40.

Listing 2.39 Illegal array definition

```
typedef struct {
  short howMany;
  Data *dataBase[];    // error: non-ANSI extension
} DataBase
```

NOTE: *Remember to change your allocation routines so that they allocate the right size for these structs. For example, in Listing 2.40 the proper allocation routine would be* `sizeof(DataBase) - (nb_elements -1) * sizeof(Data)`.

Listing 2.40 Fix for illegal array definition

```
typedef struct {
  short howMany;
  Data *dataBase[1]; // OK: now ANSI compliant
} DataBase
```

illegal assignment to constant

This error is issued by the compiler when an expression attempts to assign a value to a constant, as in Listing 2.41.

Listing 2.41 Illegal assignment to constant

```
...
const int i=5;
...
i=10;   // cannot assign to a const
```

Fix Check the assignment in question. This error may be caused by a spelling mistake that attempts to assign a value to a similarly named constant instead of the desired variable.

illegal bitfield declaration

The compiler issues this error when a bitfield of size zero is declared. This error is also issued if too many bits are requested, as in the example below.

Fix Check the bitfield declaration in question to make sure that a size of zero is not declared, and that too many bits are not requested.

Listing 2.42 Illegal bitfield declaration

```
...
long err:33;
...
```

illegal character constant

This error is signaled when an attempt is made to assign an illegal character constant. Listing 2.43 contains three examples of illegal character constants.

Fix Examine the character constant to make sure it is assigned a legal character.

Listing 2.43 Illegal character constant

```
  char FooCh;
...
...

  FooCh = '';         // each of these assignments
  FooCh = '\x';       // contain illegal character
  FooCh = 'ddjdjdj';  // constants.
```

illegal constant expression

This error is issued when the compiler encounters a constant expression that contains an illegal value or operator.

Fix Examine and correct the constant expression in question. If this error is not apparent, it is likely being caused by a previous error. Correct all previous errors and recompile.

illegal ctor initializer

This error is signaled when the compiler encounters an illegal ctor initializer. For example, the ctor initializer in Listing 2.44 is illegal because there's no `i` member of the `aClass`.

Fix If you are having problems with constructors of a sub-class, you are probably not naming the parent class explicitly, such as in Listing 2.45.

Listing 2.44 Illegal ctor initializer

```
class aClass {
  aClass():   i(12) {} // error:   illegal ctor
                       //  (no i member)
};
```

`CButton` is a subclass of `Clickable` which takes a type `DisplaySystem*` as its argument. In CodeWarrior C++ when making a constructor which passes parameters to the parent class, you need to name the parent class explicitly. You must use the following template:

```
   BLAH::BLAH(parameter1, parameter2) :
       PARENT_OF_BLAH(parameter2)
```

Listing 2.45 Illegal ctor initializer, explicit parent class

```
CButton::CButton(long resourceBase,
  DisplaySystem* displaySystem) : (displaySystem)
```

illegal data size

The compiler issues this error when a line in an assembly function contains an illegal data size, as in Listing 2.46.

Listing 2.46 Illegal data size

```
...
move.Z #0,d9
...
```

illegal default argument(s)

This error is given when the compiler finds a function which contains one or more illegal arguments, as in the function prototype of Listing 2.47.

Listing 2.47 Illegal default argument

```
...
int func(int x=1, int z);
...
```

illegal empty declaration

The compiler gives this error when a declaration is missing an identifier, as in the declaration in Listing 2.48.

Listing 2.48 Illegal empty declaration

```
...
int ;
...
```

illegal explicit template instantiation

The compiler generates this error whenever it encounters an illegal explicit template instantiation. Listing 2.49 provides an example where the function `f()` was not properly declared as a template function.

Listing 2.49 Illegal explicit template instantiation

```
//template <class T>
void f();
template void f<int>();     //  error
```

illegal function definition

This error is signaled whenever the compiler encounters an illegally defined function.

Fix If the error is not apparent, it is likely being caused by a previous error. Correct all previous errors first and recompile.

illegal function overloading

A common cause for this error is the declaration of functions with the same name and identical arguments, but different return types.

See Also ARM p.307, 13 Overloading.

illegal function return type

This error is given when the compiler finds a function that returns an array or function. A function cannot return an array or function. A function can only return a pointer to an array or function.

Fix Modify your code so that the function in question returns a pointer to the array or function.

illegal implicit const pointer conversion

(Warning Message) The compiler issues this warning when you convert a `const` pointer into a variable, as in Listing 2.50.

Listing 2.50 Illegal implicit const pointer conversion

```
void func(const char *cptr)
{
  char *ptr=cptr;
      // illegal implicit const pointer conversion
}
```

illegal implicit conversion from ‘Type_A’ to ‘Type_B’

This error is signaled when the compiler encounters an illegal implicit conversion as in Listing 2.51.

NOTE: *ANSI C++ differs from ANSI C in the treatment of* `void*`*. ANSI C allows an implicit conversion from a pointer to void to a pointer to another object type (but not to a pointer to function type--see Section 5.4); in C++ a* `void*` *cannot be assigned to an object of any type other than* `void*` *without an explicit cast. Thus, Listing 2.51 is legal ANSI C, but is not accepted in C++:*

See Also ANSI Draft Standard Section 5.4 “Pointer Conversions,” ANSI C Standard 3.2.2.3

Listing 2.51 Illegal implicit conversion

```
void f(char *cptr, void *vptr)
{
  cptr = vptr;
                 // Illegal in C++, but not C
  char *ptr=(void *)0;
                 //   Illegal in C, but not C++
  // . . .
}
```

illegal implicit enum conversion

The compiler gives this message when an illegal implicit conversion, involving an enum, is encountered. If the source code is C++, the compiler gives this message as an error. If the source code is C and the **Extended Error Checking** option in the C/C++ Warnings settings panel is on, the compiler gives this message as a warning. An example is shown below in Listing 2.52.

Listing 2.52 Illegal implicit enum conversion

```
enum ff { foo };
enum ff x = 0;
//error:illegal implicit enum conversion
```

illegal initialization

This error occurs when a variable, or other data type, is illegally initialized within a function.

illegal instruction for this processor

The compiler issues this error when an assembly-language instruction is found that does not exist for the 68000 family of microprocessors.

illegal jump past initializer

This error is signaled when a transfer is made into a block that bypasses initializers. In certain cases, it is illegal to jump past explicit or implicit initializers. Generally this error occurs whenever there is a section of code that can be jumped past in the same scope. Typically in `switch` or `goto` statements as in Listing 2.53.

NOTE: *It's possible for the definition of* `p` *to be skipped over within the scope it's in (the switch statement). The fix to this is to either define* `p` *outside of the switch, or make a new scope. If you* *have any `goto` statements in your function, you'll get this error for any variables that are defined after a `goto`. The solution is the same: either define all variables before the `goto`, or introduce a new scope.*

See Also ARM p. 87, 6.4.2 The Switch Statement and p.91, 6.7 Declaration Statement.

Listing 2.53 Illegal jump past initializer

```
switch (i) {
  int v1 = 2;  // error
   case 1:
     short v2 = 3;
   case 2:
      if( v2 == 7 ) {}  // error
 }
```

illegal operand

This error is signaled when an operator is applied to a non-compatible operand.

Fix Try type-casting the operand to a compatible type.

illegal operands for this processor

This error is issued when the compiler encounters an assembly-language instruction that refers to operands that do no exist for the 68K family of microprocessors.

illegal operation

An operator, such as == or + was illegally applied to a struct or union. This error is also signaled when an operator is not defined for a data type.

illegal operator

This error is signaled when the compiler encounters an illegal operator, as in Listing 2.54.

Listing 2.54 *Illegal operator*

```
...
int operator .(int i);
...
```

illegal operator overloading

A common cause for this error is trying to overload an operator that cannot be overloaded, or trying to overload a preprocessor directive.

See Also ARM p.329, 13.4 Overloaded Operators.

illegal precompiled header compiler flags or target

The compiler gives this error when a precompiled header file uses the wrong compiler target. For example, you get this error if you ar compiling code for a Power Mac OS computer with a precompiled header that has a first line of `#include <MacHeaders68k>`.

Fix Check your pre-compiled header or `.pch` file for flags or data of a different CPU type than your current target. Also check the prefix file in the C/C++ Language settings panel.

illegal precompiled header version

The compiler gives this error when a precompiled header file is old or defective.

Fix Check all the precompiled header files included with your project. If the error is not apparent, check the header specified in the **Prefix File** field in the C/C++ Language settings panel. For more information consult the the *C, C++, and Assembly Language Reference*.

illegal register list

This error occurs when the compiler encounters an illegal register list in an assembly function. An example is shown in Listing 2.55.

Listing 2.55 Illegal register list

```
...
movem.l  d0-d0,–(sp)
...
```

illegal SOM function overloading

You cannot overload member functions in a SOM class. For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SOMObjects Developer Toolkit* (IBM).

illegal SOM function parameters or return type

You cannot use `long double` parameters or return type in member functions for SOM classes. For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SOMObjects Developer Toolkit* (IBM).

Fix Rewrite the function using a different parameter or return type.

illegal storage class

The compile issues this error when an illegal storage class is used.

If the compiler points to a static member, it may mean that you have defined a member as `static`, ARM (p. 179) says “A data or function member of a class may be declared static in the class declaration” — but not in the definition.

Likewise, you may not declare variables as `auto` in global scope as in Listing 2.56.

See Also ARM , p.179.

Listing 2.56 Illegal storage class

```
...
auto int x;
      //error:  auto is not allowed in global scope
...
```

illegal string constant

This error is displayed when a string constant is encountered that does not terminate before the end of a line.

Fix Terminate the string before the end of a line. If this error is not apparent, it is likely being caused by a previous error. Correct all previous errors and recompile.

illegal struct/union/enum/class definition

The compiler issues this error when an illegal struct, union, enum, or class definition is encountered.

Fix Examine the illegal struct, union, enum, or class definition to check for any syntax errors.

illegal template argument(s)

The compiler gives this error when it finds a template which contains one or more illegal arguments, as shown in Listing 2.57.

Fix This error is often the result of the omission of a space in nested templates creating a right shift operator.

Listing 2.57 Illegal template argument

```
map<double, double, less<double>> aMap;
    // illegal argument
map<double, double, less<double> > aMap;

template <class T> class aClass;
aClass<int,int> *aClassptr;
    // illegal argument
```

illegal template declaration

The compiler gives this error when a malformed template declaration is encountered, such as the template declaration in Listing 2.58.

Listing 2.58 Illegal template declaration

```
template <class T> T i;         //  error
```

illegal token

The compiler issues this error when an Illegal preprocessor token is found. For example, the `@` symbol in Listing 2.59 is an illegal token.

Fix Remove the illegal token. If the illegal token is not apparent, the error may be caused by a previous syntax error. Fix all previous errors and recompile.

Listing 2.59 Illegal token

```
  // ...
if( @ ) // ...
  // ...
```

illegal type

The compiler generates this error message when an illegal type is encountered as in Listing 2.60.

Listing 2.60 Illegal type

```
static void func(int i)
{
 delete i; // <-- illegal type
}
```

illegal type cast

This error occurs when the code attempts to typecast data to an incompatible data type.

illegal type qualifier(s)

The compiler issues this error when an illegal type qualifier, for this type in this scope, is encountered. For example, the double `const` qualifier in Listing 2.61 below will produce an illegal type qualifier error.

Fix Remove the illegal type qualifier. If this error is not apparent, it may be caused by a previous error. Correct all previous errors and recompile.

Listing 2.61 Illegal type qualifier

```
...
const const int x;   //  double const
...
```

illegal use of #pragma outside of SOM class definition

You can use four SOM pragmas only within the definition of the SOM class that the apply to: `SOMReleaseOrder`,`SOMClass-Version`, `SOMMetaClass`, `SOMCallStyle`. For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SOMObjects Developer Toolkit* (IBM).

illegal use of #pragma parameter

This message is displayed when a previous `#pragma` parameter does not match a function. For example, `func()` in Listing 2.62 below does not match the function `func1`.

NOTE: *In addition, If the Illegal Pragmas option is selected in the C/C++ Warnings settings panel, undefined `#pragmas` are marked as warnings.*

See Also For more on the `#pragmas` supported by Metrowerks C/C++, consult the *C, C++, and Assembler Language Reference*, Chapter 5, “Pragmas & Predefined Symbols.” For more on the C/C++ Warnings settings panel, *C, C++, and Assembler Language Reference*, Chapter 2, “C and C++ Language Notes.”

Listing 2.62 Illegal usage of #pragma parameter

```
// ...
#pragma parameter __A0 func
char *func1()
{
  // ...
}
```

illegal use of 'HandleObject'

The compiler gives this error when an illegal usage of a class that is derived from the `HandleObject` class is encountered.

illegal use of 'this'

This error is signaled when the compiler encounters C++ code that uses `this` in a non-member function.

illegal use of abstract class ('_aClass')

The compiler gives this error when you attempt to instantiate from an abstract class. An abstract class is defined with at least one pure virtual method. A pure virtual method is declared as

```
// pure virtual method
virtual type MethodName ( arguments ) = 0;
```

An abstract class requires you to make a subclass that provides methods to replace any pure virtual methods.

Fix Abstract classes must be subclassed before being instantiated. Define a non-abstract subclass and derive your object from that subclass.

See Also ARM p. 214, 10.3 Abstract Classes.

illegal use of direct parameters

This error is issued when a function, or another expression, references a direct parameter that is not supported. For example, the example below attempts to use __X. This is illegal because a direct parameter must be either `__D0` to `__D2`, `__A0`, `__A1`, or `__FP0` to `__FP3`.

illegal use of incomplete struct/union/class

This error is signaled when an incomplete struct, union, or class is used illegally. For example, in Listing 2.63, an attempt is made to create an incomplete object.

Fix To avoid the errors, you can try to include `bar.h` so you get the full class declaration. Sometimes this does not work because of circular references in the include files. Another option is to avoid in-line (include file) references to member variables or methods of these partial classes. Don't inline the offending function. Put it in a separate implementation file that includes both `foo.h` and `bar.h`.

Listing 2.63 Illegal use of incomplete struct

```
...
struct A x;
//     ^ cannot create an incomplete object
...
```

This error often happens when you attempt to use classes that have been partially declared, usually using forward declarations as follows:

Listing 2.64 Using a class that has been partially declared

```
// foo.h
class Bar;           // empty forward declaration

class aClass {
public:
  // trouble coming up...
  void DoIt(int x) { mBar->DoStuff(x); }
        ...
private:
  Bar* mBar;        // you declare it
};
```

```
bar.hclass Bar {    // actual declaration
                    // of the class
  public:
       void DoStuff(int x);
        ...
};
```

illegal use of inline function

This error is signaled when an inline function is used illegally. For example, in Listing 2.65 an attempt is made to take the address of an inline function.

Listing 2.65 *Illegal use of inline function*

```
pascal Handle NewHandle(Size byteCount) = 0xA122;
...
&NewHandle;    // error:   cannot take address of
               //          inline function
...
```

illegal use of keyword

This error occurs when a keyword is used illegally. In some cases, this error is caused by a previous syntax error or missing symbol. For example, in Listing 2.66, the illegal use of keyword error is caused by a missing colon.

Fix Fix all previous error messages. If error still persists, verify that you are using the keyword correctly as in Listing 2.67

Listing 2.66 *Illegal use of keyword*

```
...
  switch(theMenu)
  {
    case APPLE_MENU_ID    // error:  missing ':'
      switch(theItem)
      {
        case ABOUT_ITEM :
        ...
        ...
      }
    break;    // illegal use of keyword error here
              // caused by above missing colon
```

Listing 2.67 Illegal use of direct parameters

```
...
int func(int x:__X) {}
    // error: direct parameter __X not recognized
...
```

illegal use of non-static member function

This error is signaled when an attempt is made to access a non-static member without having an object of that class, as in Listing 2.68.

Listing 2.68 Illegal usage of non-static member function

```
struct stType {
  stTypef();
};
// ...
  stTypef();
      // error: cannot call without a stType object
// ...
```

illegal use of precompiled header

This error is given when the compiler encounters a precompiled header file included illegally. A precompiled header file is used illegally when more than one precompiled header file is `#included` in the source code file (as in Listing 2.69).

Fix Check all the precompiled header files included with your project. If the error is not apparent, check the header specified in the **Prefix File** field in the C/C++ Language settings panel.

Listing 2.69 Illegal usage of precompiled header: too many

```
#include <MacHeaders>
#include <MacHeaders>
      // error: only one precompiled header
      //      file allowed
...
```

This error often occurs when the precompiled header file has already been `#included` in the **Prefix File** field in the C/C++ Language settings panel. Or, when a precompiled header is `#included` after a function, variable, or type declaration, as in Listing 2.70.

Listing 2.70 Illegal usage of precompiled header: declaration

```
long l;
#include <MacHeaders>
        // error: precompiled header included
        //        following declaration.
...
```

illegal use of pure function

The compiler generated this message when it encountered an improper usage of a pure virtual function. A pure virtual function has no definition. For example in Listing 2.71 the constructor attempts to call the pure function `myFun()`.

See Also ARM, p. 214, 10.3, Abstract Classes.

Listing 2.71 Illegal use of pure function

```
class pure {
  public:
  virtual int myFun() = 0;
  pure() { myFun(); }
};
```

illegal use of ‘void’

The compiler gives this error when an operator is incorrectly applied to a void type, or a variable is declared as a `void` type, as in Listing 2.72.

Listing 2.72 Illegal use of ‘void’

```
int  myInt;
long myNumber;
void myFoo;
//error: a variable cannot be declared as void
```

illegal use of register variable

This error occurs when a register is used illegally. For example, in Listing 2.73, an attempt is made to take the address of a register variable.

Listing 2.73 Illegal use of register variable

```
// ...
register int i;
f(&i);       // error: cannot take address of i
// ...
```

inconsistent linkage: ‘extern’ object redeclared as ‘static’

The compiler will generate this error message in C++ if you try to re-declare or define an extern object as static. This is shown in Listing 2.74.

Listing 2.74 Inconsistent linkage: ‘extern’ object redeclared as ‘static’

```
void f();
static void f(); // <<< err
```

introduced method method is not specified in release order list

If you use the `SOMReleaseOrder` pragma for a SOM class, the pragma must list all the new methods that the class declares (but not the methods it overrides). For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SOMObjects Developer Toolkit* (IBM).

Fix There are two ways to fix this problem:

- Include the method in the `SOMReleaseOrder` pragma’s list.
- Remove the `SOMReleaseOrder` pragma. The compiler assumes the release order is the same as the order in which the functions appear in the class declaration. However, when you release a version of the class, use the pragma, since you’ll need to modify its list in later versions of the class.

J, K, L

These are C/C++ compiler error messages that begin with *J, K,* or *L.*

label ‘_Lgt’ redefined

The compiler generates this error when an attempt is made to redefine a label, in this case *_Lgt*, that has already been defined.

Fix Remove or rename one of the labels.

Local data >32k

This error is issued when the local data totals exceed 32K. The local data, usually a declared array, is stored on the stack and has a limit of 32K.

Fix You can overcome this restriction by defining an array as static or using dynamic allocation to move the storage from the stack to the heap.

local data > 224 bytes

(Mac OS PPC) This error is caused in assembly functions which have no stack frame. In such a function there is a limit of 224 bytes of local variables.

Fix To resolve this create a stack frame, using the `fralloc`/`frfree` directives.

local variable '<name>' was not assigned to a register

(Mac OS PPC) The compiler generates this error when a `register` variable was named but not assigned as a `register`. In assembly functions, any variable declared `register` is guaranteed to be in a `register` and its name may be used anywhere a `register` is valid.

Fix This usually is because there are already too many `register` variables being used. Remove some previously assigned `register` variables to resolve this error.

M, N, O

These are C/C++ compiler error messages that begin with *M*, *N*, or *O*.

macro '_Mc' redefined

The compiler generates this error when an attempt is made to redefine a macro, in this case *_Mc*, that has already been defined.

Fix Remove or rename one of the macros.

macros too complex or recursive

This error is signaled when a macro cannot be expanded because it is too complex.

Fix You can resolve this error by studying the macro and redesigning it with less complexity

no parameters allowed in SOM class constructors

The constructor for a SOM class cannot contain constructors. For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SOMObjects Developer Toolkit* (IBM).

no static members allowed in SOM classes

A SOM class cannot contain static data members. For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SOMObjects Developer Toolkit* (IBM).

non-const ‘&’ reference initialized to temporary

The compiler generates this message when the initial value for a reference type is not an lvalue of that type. The compiler will create a temporary for the initialization. However, there is no storage for this temporary, as in Listing 2.75.

See Also : “not an lvalue” on page 73

Listing 2.75 Non-const ‘&’ reference initialized to temporary

```
long  &r = 40000;
// the proper method to use  is
long x;
long &y = x;
y = 40000;
```

new SOM callstyle method method must have explicit ‘Environment *’ parameter

If you create a SOM class that uses new IDL callstyle, each of the class’s methods must contain an Environment pointer as its first argument. For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SOMObjects Developer Toolkit* (IBM).

Fix There are two solutions:

- Add an Environment pointer to the method's arugment list as its first argument.
- Use the `SOMCallStyle` pragma to declare that all of the class's methods use the older OIDL callstyle. The `SOMCallStyle` method looks like this:

  ```
  #pragma SOMCallStyle OIDL
  ```

not a struct/union/class

The compiler expected to find a struct, union, or class, but found a simple type instead, as in Listing 2.76.

Listing 2.76 Not a struct

```
long  var;

var.myfoo = 10;  // error: var is not a struct
...
```

not an lvalue

The compiler expected an expression referring to an item, such as a variable, to which it can assign a value. Another expression was found instead.

number is out of range

The compiler signals this error when a numeric value is encountered that is out of range for its data type.

See Also For a complete list of data types supported by Metrowerks C/C++, see the Metrowerks *C, C++, and Assembler Language Reference*.

P, Q, R

These are C/C++ compiler error messages that begin with *P*, *Q*, or *R*.

pascal function cannot be overloaded

CodeWarrior does not let you overload pascal functions. The Listing 2.77, when compiled, will issue this error.

Listing 2.77 Illegal pascal function overloading

```
int f(int);
pascal void f();    //  error
```

pointer/array required

This error is issued when the compiler finds a left bracket, `[`, following a variable which is neither a pointer nor an array. The left bracket can only follow a pointer or array name.

possible unwanted ‘;’

(Warning Message) A semicolon was found immediately following a `while`, `if`, or `for` statement. This may cause an unintended logical error, as in Listing 2.78. This warning is signaled when the **Possible Errors** option is selected in the C/C++ Warnings settings panel.

Fix Either remove the unwanted semicolon or deselect the **Possible Errors** option in the C/C++ Warnings settings panel.

See Also For more on the **Possible Errors** option, consult the *C, C++, and Assembler Language Reference*, Chapter 2, “C and C++ Language Notes.”.

Listing 2.78 Possible unwanted ‘;’

```
...
while (x < 10);  // possible unwanted ';'
   printf(“%d  “, x);
...
```

‘pointer to member’ is not supported for SOM classes

You cannot use take the address of a member of a class that’s descended from SOMObject. For example, `&foo::bar` is not allowed if `foo` is descended from SOMObject. For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SOMObjects Developer Toolkit* (IBM).

possible unwanted assignment

(Warning Message) This warning occurs when an assignment (= operator) occurs within a logical expression in a `while`, `if`, or `for` statement. This may be meant as an equality operation, as in Listing 2.79. This is signaled as an error when the **Possible Errors** checkbox is selected in the C/C++ Warnings settings panel.

Fix Either correct the unwanted assignment or deselect the `Possible Errors` checkbox under in the C/C++ Warnings settings panel.

See Also For more on the **Possible Errors** checkbox, consult *C, C++, and Assembler Language Reference*, Chapter 2, “C and C++ Language Notes.”.

Listing 2.79 Possible unwanted assignment

```
// ...
if (x = 20) printf(“OK”);
    // possible unwanted assignment.
//...
```

possible unwanted compare

(Warning Message) This warning occurs when the compiler believes it finds an unwanted comparison as in Listing 2.80.

NOTE: *The error is signaled when the* **Possible Errors** *checkbox is selected in the C/C++ Warnings settings panel.*

Fix Either correct the unwanted comparison or deselect the **Possible Errors** checkbox under the C/C++ Warnings settings panel.

See Also For more on the `Possible Errors` checkbox, consult *C, C++, and Assembler Language Reference*, Chapter 2, “C and C++ Language Notes.”.

Listing 2.80 Possible unwanted compare

```
// ...
x == 1;  // possible unwanted comparison
// ...
```

preceding #if is missing

This error is issued when an `#endif` directive is found without a matching `#if` directive.

Fix Examine the logic behind previous nested `#if` structures to make sure you haven't included an additional `#endif` directive.

preceding '#pragma push' is missing

The compiler generates this error when it encounters a `#pragma pop` encountered that does not have a matching, preceding `#pragma push`.

For more on the pragmas available in CodeWarrior, consult the *C, C++, and Assembler Language Reference*.

preprocessor syntax error

The compiler encounters an illegal preprocessor directive, as in Listing 2.81.

Fix Check the syntax of the directive in question. If the error is not apparent, it is likely being caused by a previous error.

Listing 2.81 Preprocessor Error

```
// ...
#include file
// ...
```

reference to label '_lbl' is out of range

This error is given when an assembly function contains a branch whose destination is out of range, as in Listing 2.82.

Listing 2.82 Reference to label '_lbl' is out of range

```
// ...
bra.s label    // error: label too far away
// ...
```

return value expected

This error is generated when a function declared to return a value, does not contain a return value. For example, the `var()` function in Listing 2.83 should return an `int` or be declared as `void`.

Fix Declare the function in question as `void`, or return a value.

WARNING! *In C++ a function declared without a return value is implied to return an* `int` *type as in Listing 2.84.*

Listing 2.83 Return value expected

```
...
int var() {}  //  error: no return value
...
```

Listing 2.84 Return value from main() function expected.

```
main()
{
  cout << “working”;
      //<--  no return error here.
}
```

RTTI option is disabled

This compiler error is displayed when run-time type identification is attempted when the `RTTI` pragma or the **Enable RTTI** option in the C/C++ Language settings panel is off

Fix Turn on the **Enable RTTI** option in the C/C++ Language settings panel.

sizeof() is not supported for SOM classes

You cannot use a class or object descended from SOMObject in a `sizeof()` expression. For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SO-MObjects Developer Toolkit* (IBM).

S, T

These are C/C++ compiler error messages that begin with *S* or *T*.

SOM class access qualification only allowed to direct parent or own class

When you invoke a method with explicit scope (such as `obj->B::func()`), the specified class (`B`) must be the same class as the object (`obj`) or a direct parent of the object's class.

For example, if class `A` is the parent of class `B` which is the parent of class `C`, then

```
C* obj = new C;

obj->C::func();  // OK: C is obj’s class
obj->B::func();  // OK: B is a direct parent
                 //     of obj’s class
obj->A::func();  // ERROR: A is NOT a direct
                 //        parent of obj’s class
```

For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SOMObjects Developer Toolkit* (IBM).

SOM class arrays are not supported

You cannot create arrays of SOM objects. For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SOMObjects Developer Toolkit* (IBM).

Fix Store the SOM objects some other way (such as a linked list).

SOM class data members must be private

All the data members of a SOM class must have private access. For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SOMObjects Developer Toolkit* (IBM).

Fix Make sure you don't declare any data members in a protected or public access area of your class.

SOM classes cannot be class members

You cannot declare a SOM class as a member of another class. For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SOMObjects Developer Toolkit* (IBM).

Fix Declare a pointers to the SOM object as a member, instead

SOM classes can only inherit from other SOM based classes

A SOM class can inherit only from classes that are descendants of SOMObject. If you use multiple inheritance, you cannot mix SOM classes and regular classes together. For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SOMObjects Developer Toolkit* (IBM).

Fix Make sure all the SOM class's base classes are descended from SOMObject. If you don't want to create a SOM class, make sure *none* of the base classes are descended from SOMObject.

SOM classes inheritance must be virtual

When you declare a SOM class, all its base classes must be virtual. For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SOMObjects Developer Toolkit* (IBM).

Fix Make sure the `virtual` keyword appears before each of the class's bases in the class's declaration.

SOM class has no release order list

(Warning Message) You created a SOM class without a `SOMReleaseOrder` list, and the **Extended Error Checking** option is on. The compiler assumes the release order is the same as the order in which the functions appear in the class declaration. For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SOMObjects Developer Toolkit* (IBM).

Fix There are two ways to avoid this warning:

- In the C/C++ Warning preferences panel, turn off the **Extended Error Checking** option.
- Include a `SOMReleaseOrder` pragma for the class which gives the release order for all the class's member functions.

SOM class must have one non-inline member function

A SOM class must have at least one member function that isn't inline. MacSOM uses this class to determine which translation unit implements the class. For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SOMObjects Developer Toolkit* (IBM).

Fix Make sure the class contains at least one member function that isn't inline. If necessary, create an empty one.

SOM runtime function 'func' not defined (should be defined in somobj.hh)

The compiler expects to find certain runtime functions in the `somobj.hh` header file, but the compiler can't find one of them. The `somobj.hh` file may have been corrupted, or you may have edited the file incorrectly. For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SOMObjects Developer Toolkit* (IBM).

Fix Replace the `somobj.hh` header file on your hard disk with a copy from the CodeWarrior CD. Modify `somobj.hh` only if you're familiar with MacSOM.

SOM runtime function 'func' has unexpected type

The compiler expects runtime functions in the `somobj.hh` header file to be defined in a certain way, but the return type for one of the functions is wrong. The `somobj.hh` file may have been corrupted, or you may have edited the file incorrectly. For more information on SOM objects, see the *Metrowerks C, C++, and Assembly Language Reference*, or the *SOMObjects Developer Toolkit* (IBM).

Fix Replace the `somobj.hh` header file on your hard disk with a copy from the CodeWarrior CD. Modify `somobj.hh` only if you're familiar with MacSOM.

string too long

The character string in question is too long.

Fix Normally this error message is displayed because a terminating quotes mark was omitted from the string. A solution is to turn on the **Color Syntax** option in the Editor preference panel.

struct/union/class member '_stType' redefined

This error appears when an attempt is made to redefine a struct, union, enum, or a class member that has already been defined.

Fix Typically this happens when you use a name you have already assigned. Remove or rename one of the struct, union, enum, or class members.

struct/union/enum/class tag '_stType' redefined

This error appears when an attempt is made to redefine a struct, union, enum, or a class tag that has already been defined.

Fix Typically this happens when you use a name you have already assigned. Remove or rename one of the struct, union, enum, or class tags.

template redefined

This error is given when an attempt is made to redefine a template that has already been defined, as in Listing 2.85.

Fix Remove or rename one of the struct, union, enum, or class tags.

Listing 2.85 Template redefined

```
template <class T> class aClass { ... };
...
template <class T> class aClass { ... };
      //  error
```

template parameter mismatch

The compiler gives this error when the member function template parameter list does not match the class parameter list, as in Listing 2.86.

Listing 2.86 template parameter mismatch

```
template <class T> class aClass { void f() };
template <class T,class U>
```

```
  void aClass<T>::f() { ... };
    //  error
```

the file ‘filename’ cannot be opened

The compiler cannot find a file name provided in an `#include` directive, as in the second `#include` directive in Listing 2.87.

Fix It is possible that the file name specified in the `#include` directive is spelled wrong or is not on a valid access path. Switch to the Finder and use the **Find** command to find the file in question.

It is also possible that the `#include` file is specified as a system include, `<...>`, when it should be specified as a user include, `"..."`. If this is the case, select the **Treat #include <...> as #include "..."** checkbox in the Access Path settings panel.

See Also For more on access paths and the **Treat #include <...> as #include "..."** preference, consult the *C, C++, and Assembly Language Reference*.

Listing 2.87 File cannot be opened

```
#include "AbstractHeader.h"
#include "Foo.h"    // This file doesn’t exist
#include <QDoffscreen.h>
```

too many initializers

This error is given when the number of initialization values is greater than the number of items specified in the declaration of the initialized structure.

Fix Typically you encounter this error when initializing elements in an array, structure or class and you attempt to assign more values than elements declared. Adjust the number of elements in the array, class or structure or use the correct number of initializers.

too many macro arguments

The compiler issues this error when an attempt is made to define a macro with more than 32 arguments, as in Listing 2.88.

Listing 2.88 Too many macro arguments

```
#define macro(arg1,...,arg33) ...
```

type mismatch ‘A_type’ and ‘B_type’

The compiler issues this error when expects to find one data type, but finds another instead.

This error may also occur if one of your functions has the same name as a Metrowerks macro. For example,Listing 2.89 gives a type mismatch error.

Fix If the data types are involved are castable, typecast the offending data type to the correct type.

In Listing 2.89, `Length` is a macro defined in the precompiled headers `MacHeaders68K` and `MacHeadersPPC`:

```
#define Length(s) (*(unsigned char *)(s))
```

To fix this error, rename your variable to something other than a predefined macro.

Listing 2.89 Type mismatch

```
class MyLine {
public:
        MLine(double theLen);
        double Length(void);
// ...
```

U, V, W, X, Y, Z

These are C/C++ compiler error messages that begin with *U, V, W,, X, Y,* or *Z*.

undefined identifier ‘_var’

This error occurs when an identifier is used that has not been defined.

Fix This is often caused from a variable not being declared within the scope it appears in. Also, check for spelling errors.

undefined label '_Lbl'

The compiler generates this error when a `goto` statement specifies a label that has not been defined within the function.

Fix Remove the `goto`, or create the necessary label within the scope where the error occurs.

undefined preprocessor directive

This error is signaled when a preprocessor directive not recognized by Metrowerks C/C++ is used.

Fix A list of preprocessor directives recognized by Metrowerks C are listed in the *C, C++, and Assembler Language Reference*. Consult this list to make sure the directive you are using exists and is spelled correctly.

unexpected end of file

The end of a source code file was reached before a language item was completed.

Fix This error may be caused by a misplaced or unbalanced right brace. Check the penultimate line of the source code file in question. If the error is not apparent, fix all previous errors and recompile.

unexpected end of line

The end of a source code line was reached before a language item was completed.

Fix This error may be caused by anything from a misplaced semicolon to a missing symbol. Check the source code line in question. If the error is not apparent, fix all previous errors and recompile.

unexpected token

This error occurs when the compiler finds an unexpected token.

Fix If the error is not apparent, it is likely being caused by a previous syntax error or missing symbol. Fix all previous errors and recompile.

unknown PowerPC instruction mnemonic

(Mac OS PPC) This error message is used to report an illegal instruction name.

Fix Correct the mnemonic in the assembler instruction.

unimplemented C++ feature

The compiler generates this error when it encounters a C++ feature that is not yet supported by Metrowerks C++.

unterminated #if / macro

This error is displayed when an `#if` directive is found with no matching `#endif` directive, or a macro definition is not complete.

unterminated comment

The end of a source code file was reached before a comment was completed.

variable '_var' is not initialized before being used

The compiler encounters an expression using a variable that has neither been assigned a value nor initialized. For example the code in Listing 2.90 would cause this error.

Fix Initialize or assign a value to the variable before using it in an expression.

Listing 2.90 Variable is not initialized before being used.

```
static int f()
{
  int i;
  return i;
}
```

variable '_var' is not used in function

(Warning Message) A variable declared in a function is not used in the function body. This warning is signaled as an error if the **Un-**

used Variables checkbox is selected in the C/C++ Warnings settings panel.

Fix Either remove the unused variable, or deselect the **Unused Variables** checkbox.

See Also For more on the **Unused Variables** checkbox, see the *C, C++, and Assembly Language Reference*.

virtual functions cannot be pascal functions

This error appears when a virtual function is declared as a `pascal` function. For example, in Listing 2.91, the class `var` illegally declares `func()` as a `pascal` function.

Listing 2.91 Virtual functions cannot be pascal functions

```
class aClass {
  pascal int func();  // illegal declaration
};
```

3

Linker Error Messages

This chapter gives an alphabetical list of the most common linker errors which may be encountered while using Metrowerks CodeWarrior compilers for both the PowerPC-based and 68K-based Macintosh.

Linker Errors

In this list, errors with variable initial text (such as a class or function name) come first. Errors beginning with a non-alphabetic symbol character come next. After that, errors are listed alphabetically.

At the beginning of the error message's description, there is the code model for the error message: Be OS for Be OS code, Mac OS 68k for 680x0 code, Mac OS PPC for PowerPC code, and Mac OS CFM68K for the code fragment manager version of 680x0 code.

Symbol Names

These are linker error messages that begin with a symbol name, the name of a variable or function.

'_foo' 16-bit code reference to '_bar' is out of range.

(Be OS, Mac OS 68K and PPC) The target symbol of a 16-bit PC-relative jump must be located with 32K of the jump statement.

Fix Rearrange the files in your project to move the target closer to the source.

'_filename: symbol1' 16-bit data reference to 'symbol2' is out of range

(Mac OS 68K) *Symbol1*, in the file *filename*, assumes a data object, *symbol2*, is within a 32K range (a near, 16-bit reference), but *symbol2* is more than 32K away from *symbol1*.

Either *symbol2* must be made closer to *symbol1* to allow a 16-bit reference or *symbol1* must use a far, 32-bit reference to *symbol2*.

Fix Fixing this linker error may require explicitly placing *symbol1* and *symbol2* in the same segment. In the Processor preferences panel, try setting the Code Model to Large, selecting the Far Data and Far Strings checkboxes.

'_foo' 8-bit (16-bit) computed reference out of range: 'bar'-'fa'

(Mac OS 68K) A computed reference cannot be resolved because the objects are too far away from each other. This will most likely come from a converted MPW library.

'_foo' has 16-bit code reference to non-code: 'bar'

(Be OS, Mac OS 68K and PPC) An assembly code statement makes a 16-bit conditional branch to a data symbol, instead of a label.

Fix Replace the data symbol with a label. If you really want to branch into a data symbol, use a non-conditional branch.

'_foo' has 16-bit data reference to non-data symbol 'bar'

(Mac OS 68K) *bar* is referenced as data but defined as code.

'_foo' has illegal computed reference between segments: 'bar'-'fa'

(Mac OS 68K) There is an illegal reference type in one of the object files. This will most likely be caused by a converted MPW library.

'_foo' has illegal single segment 16-bit reference to 'bar'

(Mac OS 68K) This error message is given when you try to use more than one segment in a non-extended resource.

Fix To fix this linker error get rid of the segmentation or to switch to an extended resource by selecting the **Extended Resource** checkbox in the 68K Project preferences.

See Also: For information on code models and code resources, see *Targeting Mac OS*.

'_foo' has illegal single segment 32-bit reference to 'bar'.

(Mac OS 68K) Single segments cannot use multiple segments or jump tables. This error message is displayed when some code was compiled with smart or large code model and the project type doesn't support it.

NOTE: *Often, the problem is from using an ANSI Library since the ANSI library was built with smart code model. If you want to use an ANSI Library, you have to make a multi-segment code resource by setting the extended resource option in your 68k project preference.*

Fix Smart or large code can only be used in multi-segment projects, like applications and multi-segment code resource. It can't be used with single-segment resources. If you want to make a multi-segment code resource, turn on the **Extended Resource** option in the 68K Project settings panel

See Also For information on code models and code resources, see *Targeting Mac OS*.

'_foo' is undefined or is not an exportable object.

(Mac OS 68K) This error message is given if the user has an entry in the `.exp` file, and the entry cannot be found or cannot be exported (i.e. marked as `#pragma internal`).

'_filename' is not a valid library file.

(Mac OS PPC and Be OS) The named file had the proper file type for a CodeWarrior library, but the contents are not valid. The library file in question may be damaged.

'_filename' is not a valid PEF shared library.

(Mac OS PPC and BeOS) The named file had the proper file type for a shared library, but the contents are not valid. The shared library in question may be damaged.

'_filename' is not a valid XCOFF file.

(Mac OS PPC and Be OS) The named file had the proper file type for an XCOFF library or shared library, but the contents are not valid. The XCOFF file in question may be damaged.

'_foo' referenced from '_bar' is undefined.

(Mac OS 68K) The Linker will generate this error message because a referenced code or data module is not defined anywhere. This means that the undefined function does not exist in your code. This could be because it is a library that you did not include, a source file you did not include or because you forgot to define the function in your code.

Fix Locate or create the referenced function and make sure it is in your project.

A, B, C

These are linker error messages that begin with *A*, *B*, or *C*.

An error occurred while importing the shared library.

(Mac OS CFM68K) This general error message is generated by the PEF Importer to flag a read error.

An error occurred while reading from the .exp file.

(Mac OS CFM68K) This general error message is generated PEF Linker is to flag a error while reading the `.exp` file.

An error occurred while linking the PEF container.

(Mac OS CFM68K) This is a general purpose error message given by the PEF linker for CFM68K. It is used to flag write errors.

An error occurred while trying to open the .exp file.

(Mac OS CFM68K) The linker generates this general purpose error message if reading of the `.exp` file fails.

An error occurred while trying to write the .exp file.

(Mac OS CFM68K) This general error message is generated PEF Linker is to flag a error while writing the `.exp` file.

application/resource must have a main entry point

(Mac OS PPC and Be OS) An application or code resource must have a main entry point specified in the PEF settings panel. If you attempt to link a PowerPC project that does not have a main entry point, this link error occurs. For more on entry points and the PEF preferences, consult “Setting the PEF Preferences (PowerPC Only)” in *Targeting Mac OS*.

Cannot load object resource for ‘bar’

(Mac OS 68K) An object resource cannot be loaded. This is probably caused by either a corrupt project file or a memory problem.

Cannot use CFM68K library.

(Mac OS 68K) This error message is generated when you try to use a CFM68k library in a traditional Mac OS application or resource

Cannot use non-CFM68K library.

(Mac OS CFM68K) This error message is generated when you try to use a a traditional Mac OS library in CFM68k application or resource

Can’t copy resource file ‘filename’.

(Mac OS PPC and Be OS) An I/O error occurred while copying one of the resource files into the final executable file. The specific I/O error is indicated as well.

Can’t find source file ‘foo’ — statement locations will be lost.

(Warning) The linker could not find the file named *foo* while importing XCOFF object code. When you debug the code, the debugger may not be able to display the source code for the file, or the debugger may display the source code but won’t be able to display dashes for breakpoints.

Can’t import PEF shared library ‘filename’.

(Mac OS PPC and Be OS) An I/O error occurred while reading the named shared library file during Make. The specific I/O error is indicated as well.

Can’t import XCOFF file ‘filename’.

(Mac OS PPC and Be OS) An I/O error occurred while reading the named XCOFF file during Make. The specific I/O error is indicated as well.

Can’t parse debug information in ‘string’

(Warning) The linker could not understand a piece of debug information while importing XCOFF code. The debugger may not display the symbol associated with the string correctly.

Can’t read export file ‘filename’.

(Mac OS PPC and Be OS) An I/O error occurred while reading the `.exp` file for this project. The specific I/O error is indicated as well.

Can’t read library file ‘filename’.

(Mac OS PPC and Be OS) An I/O error occurred while reading the named library file during linking. The specific I/O error is indicated as well.

Can't read temp file ‘filename’

(MPW) You use the `-tmp` directive, and the linker cannot read from or write to one of the temporary files that the linker created in the folder you specified.

Fix Make sure the name of the folder is correctly spelled, the path is correct, the folder’s volume is mounted, and that you have access privileges to it.

Can’t write application ‘filename’.

(Mac OS PPC and Be OS) An error occurred while the linker was writing the executable file, which might be an application, shared library, or code resource. The specific error is indicated as well. It can be an I/O error such as “Disk Full” or it can be an “Out Of Memory” error.

Fix Memory errors can usually be fixed by increasing the partition size of CodeWarrior.

Can’t write export file ‘filename’.

(Mac OS PPC and Be OS) An I/O error occurred while the linker was writing the `.exp` file. The specific error is indicated as well. If you select “Use .exp file” from the PEF settings panel of the Preferences, but the file *projectname*`.exp` does not exist, the linker will write a default `.exp` file containing all global symbols.

See Also: “Export Symbols” in *Targeting Mac OS*.

Can’t write library file ‘filename’.

(Mac OS PPC and Be OS) An error occurred while the linker was writing the library file. The specific error is indicated as well. This can be an I/O error such as “Disk Full” or it can be an “Out Of Memory” error.

Fix “Out Of Memory” errors can usually be fixed by increasing the partition size of CodeWarrior.

Can’t write link map ‘filename’.

(Mac OS PPC and Be OS) An error occurred while the linker was writing the link map file. The specific error is indicated as well. This can be an I/O error such as “Disk Full” or it can be an “Out Of Memory” error.

Fix “Out Of Memory” errors can usually be fixed by increasing the partition size of CodeWarrior.

Can’t write SYM file ‘filename’.

(Mac OS PPC and Be OS) An error occurred while the linker was writing the SYM file. The specific error is indicated as well. This can be an I/O error such as “Disk Full” or it can be an “Out Of Memory” error.

Fix “Out Of Memory” errors can usually be fixed by increasing the partition size of CodeWarrior.

code resource must not have a termination routine

(Mac OS PPC and Be OS) A native or fat code resource cannot have a termination entry point, specified in the PEF settings panel.

cross-TOC call from ‘symbol1’ to ‘symbol2’ has no TOC reload slot

(Mac OS PPC and Be OS) The linker requires that a `NOP` instruction be placed after any call to an external routine, as a place-holder for a TOC-reload instruction needed when calling a routine from a shared library. This error can occur if you import some assembly language code that does not contain the `NOP`, but calls a routine which is imported from a shared library.

D, E, F

These are linker error messages that begin with *D*, *E*, or *F*.

entry-point ‘symbol’ is not a descriptor

(Mac OS PPC and Be OS) The initialization, main, and termination entry points are usually external routines, that the linker references through a descriptor (sometimes called a TVector) in the data area. This message is only a warning. For certain kinds of code resources (like plugins), you may want to use a variable as the entry-point instead of a function.

Entry Point ‘_pt’ is undefined.

(Mac OS CFM68K) In the CFM68K settings panel, the user can specify three entry points: Initialization, Main, and Termination. This message occurs when you specify an entry point that doesn't exist.

Error while operation

(Mac OS 68K) Linker errors in this form occur when the linker encounters file I/O problems. *Operation* may one of the following:

- saving resources
- writing to SYM file
- setting SYM file position
- getting file info
- creating SYM file
- opening SYM file
- reading or parsing SEG file

- creating new file
- creating or copying resource fork
- opening file's resource fork
- writing file's resource fork

These linker errors can be caused by anything from defective media to a volume being full. The solution depends on the nature of the error.

export symbol ‘symbol’ is undefined

(Mac OS PPC and Be OS) The symbol named in the `.exp` file does not exist.

G, H, I

These are linker error messages that begin with *G*, *H*, or *I*.

Global Object ‘obj’ was already declared in File: ‘fname’.

(Mac OS 68K) The linker will generate this error if you define a global data or code object in more that one file

Fix Remove all but one declaration, or declare all but one to be external.

HUNK_DEINIT_CODE not yet supported.

(Mac OS 68K) This error message is generated when an unsupported object type is encountered.

NOTE: *You will not get this error unless your object file has been corrupted.*

ignored duplicate resource ‘type’ (id) from ‘filename’

(Mac OS PPC and Be OS) The named resource was already copied from another resource file in the project.

ignored: ‘symbol’ class in ‘filename1’ previously defined in ‘filename2’

(Mac OS PPC and Be OS) The linker will permit multiple definitions of a symbol that is defined in a library. The order of files in the project window determines which definition of the symbol will be used. Subsequent definitions are ignored.

Illegal computed reference for ‘foo’

(Mac OS 68K) The linker encountered an illegal computed reference for *foo*.

Illegal relocation for ‘foo’

(Mac OS 68K) An illegal relocation information.

Illegal object data

(Mac OS 68K) This error message is generated when the linker encounters illegal data in a library.

Illegal object data in ‘objectFile’

(Mac OS 68K) The linker does not recognize the data format in file *objectFile*. Usually, this error is issued when you attempt to link a PowerPC library with a 68K project.

Fix Make sure the file is in a format recognized by the CodeWarrior project manager. To fix this error, build the library’s source code with a 68K compiler and linker or find a 68K version of the PowerPC library.

Illegal object file version, an error occurred while writing the library’s object data.

(Mac OS 68K) An error occurred during the file I/O.

Invalid object code.

(Mac OS PPC and Be OS) You tried to link a 68K library with a PowerPC program.

Fix To fix this error, build the library’s source code with a PowerPC compiler and linker or find a PowerPC version of the 68K library.

Invalid PEF shared library.

(Mac OS PPC and Be OS) The named file had the proper file type for a CodeWarrior library, but the contents are not valid. The library file in question may be damaged.

J, K, L

These are linker error messages that begin with *J*, *K*, or *L*.

Library Linker: Cannot have resource files in library.

(Mac OS 68K) This library linker error message is shown if you have a `.rsrc` file in your library project.

NOTE: *This error is often generated by including a ResEdit file in the project.*

Library Linker: Cannot load object resource.

(Mac OS 68K) The library linker generates this message when there is not enough memory. Alternatively the resource file maybe damaged.

Library Linker: Illegal object data.

(Mac OS 68K) This error is generated when the linker encounters illegal data in a library.

Library Linker: Out of memory.

(Mac OS 68K) This library linker error message is shown when there is not enough memory.

library must not have any resource files

(Mac OS PPC and Be OS) You can't add any resource files in a project whose type is library. If the library requires resources, the resource files must be added to the projects which use the library.

Library resource cannot be read.

(Mac OS 68K) This error message is displayed whenever you have a library file error.

Link aborted - Too many errors.

(Mac OS 68K) The linker will generate this error message more than 100 linker errors are encountered.

Link Error: Code resource cannot have more than one segment.

(Mac OS 68K) This error appears if you try to have more that one code segment in s single segment resource

Link Error: Object resource not found.

(Mac OS 68K) This linker error occurs when an object resource cannot be loaded. This could be a corrupt project file or memory problem.

Link Error: Runtime Object resource not found.

(Mac OS 68K) This linker error will not occur unless you have modified the compiler's resources. If you receive this linker error, contact Metrowerks Technical Support.

Link failed

(Mac OS PPC and Be OS) A fatal link error such as "Out Of Memory" has occurred. The specific link error is indicated as well.

Local Object 'foo' is redeclared

(Mac OS 68K) The local object *foo* is declared more than once within one object file.

M, N, O

These are linker error messages that begin with *M*, *N*, or *O*.

'main' is undefined.

(Mac OS 68K) This Linker error message occurs when there is no `main()` function defined in the program.

Fix You need to fix this error differently, depending on whether your project is for an application, C code resource or Pascal code resource.

Application In an executable application this error is usually the result of not including the source file that includes the `main()` function in your project.

C code resource In a multi-segment project, your `main()` must be in Segment 1. If your `main()` is not in Segment 1 then the CodeWarrior start-up code tries to call your `main()` via an A4-based reference. However, A4 is not setup until your `main()` has a chance to call `SetCurrentA4()`.

Pascal code resource Code resources can only be built from units. In the units `INTERFACE` section, you have to indicate the code resources main entry point. This entry point is indicated with the `$MAIN` directive. You need to specify the missing `main` identifier with the `$MAIN` directive.

missing vtable: '_vt_foo' Check that all virtual functions and static members are defined

(Mac OS PPC and Be OS) The linker generates this error message because the compiler usually tries to generate only one instance of a virtual function table. This table is usually created together with a certain static member or a virtual function definition. If you forget to define that member you will get this error message.

NOTE: *This message is specific to a C++ project. The error usually means you are making an instance of a derived class without declaring the base classes or including the base class' header file in your derived class header files.*

multiply-defined: 'symbol' in 'filename1' previously defined in 'filename2'

(Mac OS PPC and Be OS) The linker will not permit multiple definitions of a symbol if neither symbol comes from a library.

NOTE: *All multiply-defined symbols are reported in a single message.*

Near data section or jump table is greater than 32KB.

(Mac OS CFM68K) With CFM68K the near data section uses A5-relative addressing, so is limited to 32KB below A5. The jump table is above A5 and also limited to 32KB.

Near data segment is bigger than 64k

(Mac OS CFM68K) The project being linked has more than the 64K limit of global data allowed using near, 16-bit references. To allow more than 64K of global data, your project needs to use far, 32-bit references.

Fix Select the **Far Data** and **Far Strings** checkboxes in the 68K Processor preferences panel.

Not a ‘CPU_Type’ Library

This Library importer encountered an incorrect library type for the project target.

Fix Replace the offending library with the correct CPU version.

Not enough memory for linker.

(Mac OS 68K) This error message is given if there is not enough memory for the linker.

output code size exceeds 64K limit; please contact sales@metrowerks.com for info on unlimited linker

(BeOS) The BeOS ships with a linker that can build applications and shared libraries that are smaller than 64K in size. If you get this error, check our release notes or call technical support for the workarounds to build several shared libraries, or call Metrowerks sales to purchase an unlimited linker.

P, Q, R, S, T

These are linker error messages that begin with *P*, *Q*, *R*, *S*, or *T*.

Symbol data error in ‘bar’

(Mac OS 68K) The linker found illegal data in the object file *bar*. This means that the object data is not compatible with the current CodeWarrior compiler. Remove all binary information and recompile your project.

Syntax error in exports file ‘fname’ , line ‘n’.

(Mac OS CFM68K) The export file must contain only identifiers, comments (indicated by a # character and terminated by the end of line) and white space (spaces or tabs)

syntax error on line ‘n’ of export file ‘filename’

(Mac OS PPC and Be OS) The export file must contain only identifiers, comments (indicated by a # character and terminated by the end of line) and white space (spaces or tabs).

The global data module ‘foo’ is undefined

(Mac OS 68K) Where foo is one of the following internal start-up code variables:

- `__codereftype__`
- `__maincodexnum__`
- `__headersize__`
- `__LoadSeg__`
- `__codexreftype__`
- `__Startup__`

This linker error will not occur unless you have modified the compiler’s resources. If you receive this linker error, contact Metrowerks Technical Support.

The main entry point for applications must be an executable module.

(Mac OS CFM68K) The CFM68k linker generates this error if the user puts a non-function as the main entry point for an application (which is illegal), for Shared Libraries

NOTE: *You can have a data object as the main entry point.*

The __segloader routine cannot be found.

(Mac OS CFM68K) The __`segloader()` routine is used to load segments under CFM68K, it is found in the `MWCFM68KRuntime.Lib` library.

The virtual function table '_vt_foo' is undefined, make sure that all static members and virtual functions are defined.

(Mac OS 68K) The linker generates this error message because the compiler usually tries to generate only one instance of a virtual function table. This table is usually created together with a certain static member or a virtual function definition. If you forget to define that member you will get this error message

NOTE: *This message is specific to a C++ project. The error usually means you are making an instance of a derived class without declaring the base classes or including the base class' header file in your derived class header files.*

TOC size of 'n' bytes exceeds 64K limit

(Mac OS PPC and Be OS) The TOC is a part of the data area used to indirectly address other data. The TOC contains one 4-byte entry for each variable, floating-point constant, or string constant that is referenced in the program.

Fix You can reduce the TOC requirements of your program with a couple of different option:

- The **Pool Strings** option in the C/C++ Language settings panel, or pragma `pool_strings`, pools the string constants from each file into a single data object which needs only 1 TOC entry.
- The **Store Static Data in TOC** option in the PPC Processor settings panel stores small static integer variables and floating-point constants directly in the TOC, instead of allocating space for them elsewhere and storing pointers to them in the TOC. If you have lots of small static variables (under 4 bytes), turn this

option on to save TOC space. If you have lots of large floating-point constants (4 to 8 bytes), turn this option off to save TOC space.

too many link errors

(Mac OS PPC and Be OS) The linker stops reporting errors after about 100 errors are encountered.

U, V, W, X, Y, Z

These are linker error messages that begin with *U, V, W, X, Y,* or *Z*.

Unable to launch the Application.

(Mac OS 68K) This linker error is given if a file I/O problem occurs when CodeWarrior performs a **Run**.

Unable to load multi-segment driver header

(Mac OS 68K) This linker error will not occur unless you have modified the compilers resources. If you receive this linker error, contact Metrowerks Technical Support.

undefined: ‘symbol1’ class’ referenced from ‘symbol2’ in ‘filename’

(Mac OS PPC and Be OS) The named symbol was referenced but never defined. As a special case, you'll get an undefined message for the symbol `__procinfo` (data) if you make a Native code resource that is not one of the known resource types: `CDEF`, `MDEF`, `MBDF`, `LDEF`, `WDEF`, `cdev`, `XCMD`, or `XFCN`.

NOTE: *Multiple references to the same undefined symbol are reported in a single message.*

Fix To avoid this error, you must define a global variable `__procinfo` of type `unsigned long` initialized to the proper `MixedMode` flags placed in the `RoutineDescriptor` that becomes part of the Native code resource. Consult the comments in the `MixedMode.h` for details on `RoutineDescriptor` flags.

unsupported XCOFF relocation (x, y, z) in ‘filename’

(Mac OS PPC and Be OS) The named XCOFF file was valid but contained some relocation information that is not supported by the CodeWarrior linker.

Fix Make sure that the XCOFF file was assembled without symbolic debugging information. If the error still occurs, contact Metrowerks Tech Support.

4

Magic Cap Linker Error Messages

This chapter gives an alphabetical list of the linker errors which may be encountered while using Metrowerks CodeWarrior compilers for the Magic Cap application code generation.

Magic Cap Linker Errors

In this list, errors with variable initial text (such as a class or function name) come first. Errors beginning with a non-alphabetic symbol character come next. After that, errors are listed alphabetically.

Symbol Names

These are Magic Cap linker error messages that begin with a symbol name, the name of a variable or function.

‘ClassName reference#’ had no name when it was referenced earlier, but is being defined with the name ‘name’

(Warning Message) This warning message is generated because another object instance referred to this object, but without a name. This object is now being defined with a name.

‘ClassName reference#’ had no name when it was defined earlier, but is being referenced with the name ‘name’

(Warning Message) The linker generates this warning when a previously defined object instance had no name, however now it is being referred to with a name.

‘file’ is not a package file

You added a file to the project that isn’t a package and the linker tried to create a bundle that contains it. Remove the file or replace it with a package.

‘_method’ has both an automatic field accessor and a compiled method

This error message is generated because you created a compiled method (operation) for a field that had an automatic getter or setter.

‘_function1’ has illegal cross-reference to ‘_function2’

This error appears if you had tried calling a function in another segment directly. You can't use A5-based jump tables.

Fix Don't make cross-segment calls; use Magic Cap method dispatcher instead or, use segmentation directives to force all methods and functions into one segment.

‘_function’ has illegal data reference to ‘_var’.

This error appears because Magic Cap can't have global data, static locals, C++ vtables or Object Pascal methods,

‘_function1’ has illegal 32-bit reference to ‘_function2’.

This error message appears if you chose **Large** from the **Code Model** menu or turned on the **Far Data** option in the 68K Processor settings panel, and you attempted to to access a global or a cross-segment function.

‘_method’ not in .Def file, but compiled

(Warning Message) This warning is generated when the linker encountered a compiled method (operation) that was never defined in the class definition files.

‘ClassName reference#’ was defined earlier, but is being referenced as a ‘differentClassName’

(Warning Message) This warning appears when this object instance was defined earlier, it was an instance of a class different than the one now being referred to.

‘ClassName objectname reference#’ was defined earlier, but is being referenced without a name

(Warning Message) An warning message is displayed because the object instance being referred to had a name when it was defined, but the reference lacks a name.

‘ClassName reference#’ was referenced earlier, but is being defined as a ‘differentClassName’

(Warning Message) The linker generates this warning when a previous object instance referred to this object, but the previous object believes this object is an instance of a different class.

‘ClassName objectName reference#’ was referenced earlier, but is being defined without a name

(Warning Message) This warning is given when a previous object instance referred to this object, but then this object had a name, now it doesn't.

A, B, C, D, E, F

These are Magic Cap linker error messages that begin with *A*, *B*, *C*, *D*, *E*, or *F*.

class ‘aClass’ is not defined, cannot make instances

This error message appears when you defined an object in your object instance definition files that does not match any known class.

class ‘aClass’ is abstract, cannot make instances

This linker error message is generated when you attempted to create an instance of an abstract class.

Fix Try creating an instance of a subclass of the abstract class if one is available that suits your needs, or create your own subclass.

can not find class ‘className’

The Magic Linker generated this message when it encountered a reference to a class that had not been defined or was in the system precompiled classes file.

cannot use the same reference number twice for '_var1 _id' and '_var2 _id'

This message is generated because you have assigned the same reference number twice.

NOTE: *Object instance reference numbers in your instance definition files are unique.*

field name wrong ('_fld1' , expected '_fld2')

This message is generated when the linker encounters a field name in your object instance definition that does not match the name as defined in the class definition.

Fix This could be due to a misspelling, or your fields being out of order. Fields must be used in the order they were specified in the class definition.

G, H, I, J, K, L

These are Magic Cap linker error messages that begin with *G, H, I, J, K,* or *L*.

instance ' _obj _id' has too many fields or undefined extra data

This error message is displayed because the linker found more fields for this object instance than were actually defined in the class definition. Alternatively, you have entered extra data for a class that does not use extra data.

instance 'Obj id ' is too big or has undefined extra data ('x' vs. 'y')

The linker generates this error message when it finds that there is more data for this object instance than the class definition actually called for.

Fix This can be caused by forgetting to update fields for the object instance that were changed in the class definition, or vice versa. Check to make sure that the object instance's fields are in synch with the class definition.

instance '_obj _id' was defined but not referenced

(Warning Message) This warning appears when an object in your object instance definition files was defined, but was never referenced by another object.

NOTE: *This may mean that the object will exist, but may never be used by your package, thus wasting space.*

instance '_obj _id' was referenced but not defined

The linker gives this error message when an object in your object definition files was referenced, but was never defined

length for field: '_fld' too long for a '_type'

This error message appears when a field value in your object instance definition is too large for the type declared in the class definition.

Fix This can be due to a typo, or you have entered a value meant for another field.

list instance 'list id' has the wrong number of entries ('x' vs. 'y')

This error appears because the number of entries for this instance of a list class is not in agreement with the length field.

Fix Make sure that the length field's value has been updated to reflect any recent changes you made to the number of entries for the list class instance in your object instance definition file.

M, N, O, P, Q, R

These are Magic Cap linker error messages that begin with *M*, *N*, *O*, *P*, *Q*, or *R*.

no definition for '_obj '

The Magic Linker gave this error when it was unable to find the object instance referenced in this field

octet string is missing closing quote

(Warning Message) This warning is generated because octets should be enclosed in string quotes.

octet string is too short

(Warning Message) This linker warning appears when you have supplied less than what was expected.

octet string can not be more than 'x' characters

(Warning Message) The warning is displayed because octets can not have more than 1024 characters.

Out of memory

This linker error message is generated because the drop-in has exhausted all memory resources.

S, T, U, V, W, X, Y, Z

These are Magic Cap linker error messages that begin with S, *T*, *U*, *W*, *X*, *Y*, or *Z*.

there are 'x' unused consecutive reference numbers (between '_var1' and '_var2')

(Warning Message) This error message is given because the Magic Linker is not able to span large gaps in object reference numbers.

Fix You need to renumber the objects in your object instance definition files.

you must include a precompiled, system classes file for a package build

This error message appears because in order to build a package, a precompiled system classes file is required.

Fix This can be set in the Magic Precompiler preferences panel.

5

Magic Cap Precompiler Error Messages

This chapter gives an alphabetical list of the pre-compiler errors which may be encountered while using Metrowerks CodeWarrior compilers for the Magic Cap application code generation.

Magic Cap Precompiler Errors

In this list, errors with variable initial text (such as a class or function name) come first. Errors beginning with a non-alphabetic symbol character come next. After that, errors are listed alphabetically.

Symbol Names

These are Magic Cap linker error messages that begin with a symbol name, the name of a variable or function.

'_opDef' already has operation number '_opNum' so don't try to give it number '_opNum'

The Magic Precompiler will not let you reassign an operation's number in your class definition files. This can happen if Magic Precompiler first encounters your operation definition and then automatically assigns it an operation number, then later encounters a direct assignment of the operation number.

'_Aclass' and '_Bclass' both have the number '_cNum'

The Magic Precompiler has noticed that you have tried to give two different classes the same class number in your class definition files.

Fix Either directly specify different numbers, or let Magic Precompiler automatically assign the numbers for you.

'_par_num' does not match '_par_num' inherited definition

This error appears when the parameters or the return value of the operation you are defining does not match those of the operation you are overriding. The Magic Precompiler insists that they be identical.

Fix Make the parameters or return values match the operation you are overriding.

'_aClass' is trying to inherit from itself

This precompiler error occurs because the Magic Precompiler, like mix-in, will not let a class inherit from itself. No Smalltalk behaviors allowed here.

Fix Modify the inheritance chain so the object does not inherit from itself.

'_obj' is trying to mix with itself

The precompiler generates this message if you try to mix-in with yourself, directly or otherwise.

A, B, C, D, E, F

These are Magic Cap linker error messages that begin with *A*, *B*, *C*, *D*, *E*, or *F*.

a class already exists with the name 'className'

This error appears because Magic Precompiler encountered an attempt to give two classes the same name in your class definition files.

Fix Classes must have unique names.

cannot assign class number 0

This error occurs because class numbers of zero are not allowed.

cannot assign operation number 0

This error occurs because operation numbers of zero are not allowed.

cannot create intrinsic '_intrName' with same name as existing operation

This error appears when the precompiler encounters intrinsics and operations sharing the same names.

cannot create operation '_opName' with same name as existing intrinsic

This error appears when the precompiler encounters operations and intrinsics sharing the same names.

cannot create two intrinsics with the same name ('_intrName')

This error occurs because intrinsics must have unique names.

can't find file 'fileName'

The Magic Precompiler generated this error because it was unable to find an included file.

can not find class 'className'

The Magic Precompiler generated this message when it encountered a reference to a class that had not been defined or was in the system precompiled classes file.

cannot handle auto attributes of size '_size'

This message appears because you cannot have attributes with a type whose size is less than 2 bytes.

cannot handle class numbers greater than 'maxClasses'

This message appears because the Magic precompiler can only handle class numbers up to 2048.

NOTE: *Even if there are less than 2048 classes, you cannot directly specify a class number greater than 2048 in your class definition files.*

cannot handle operation numbers greater than '_maxOp'

This message is displayed because the Magic Precompiler will only allow up to 5300 operations.

NOTE: *The Magic Precompiler also will not allow you to directly specify operation numbers in your class definition files greater than the maximum value.*

can not inherit from '_aClass' because it is not yet defined

The Magic Precompiler gave this error when it noticed that you are inheriting from a class that has not been defined yet.

Fix Try reordering your include files such that the class being inherited from is defined before the class that uses it.

can not inherit from '_aClass' because it is sealed

Magic Precompiler generated this error when it you tried to inherit from a sealed class.

can not mix with '_aMix' because it is not yet defined

The Magic Precompiler gave this error when it noticed that you are using a mix-in that has not been defined yet.

Fix Try reordering your include files so that the mix-in is defined before the class that uses it.

cannot override a intrinsic

This error occurs because intrinsics cannot be overridden.

cannot override operation '_op' because it is sealed (with a kiss) in a superclass

Magic Precompiler generated this error when you tried to override a sealed operation (method).

cannot redefine getter '_aGet' as an operation

This precompiler error appears if you attempt to redefine a getter, or setter, as an operation.

NOTE: *Names for operations, getters and setters are global, so all must be unique.*

cannot redefine operation '_op' as an attribute

The Magic Precompiler gives this error if you name an attribute the same as an operation.

NOTE: *Names for operations and attributes are global, and thus must be unique. (Attribute names and getters are allowed to be the same.)*

cannot redefine setter '_aSet' as an operation

The Magic Precompiler gave this error when you attempted to redefine a getter, or setter, as an operation.

NOTE: *Names for operations, getters and setters are global, so all must be unique.*

class '_aClass' cannot be a mix-in more than once

This error appears because your class directly or indirectly used the same mix-in more than once.

class '_aClass' cannot use extra data, since one of its superclasses already does

This Magic Precompiler error appears because you attempted to use extra data for the class being defined, and one of the classes it inherits from already uses extra data.

Fix Either don't inherit from the class that uses extra data, or consider adding a field that references an object that can hold the data you wanted to put into extra data.

conflict between system class name and package class name '_aClass'

The Magic Precompiler gave this error when you attempted to use the name of a system class for a package class in your class definition file.

NOTE: *Classes must have unique names.*

could not find '_op' to override

This error was given when you overrode an operation that had not been previously defined.

G, H, I, J, K, L

These are Magic Cap linker error messages that begin with *G, H, I, J, K,* or *L*.

getter must have no parameters

This error was given because your getter had an argument. Getters return a value and take no parameters.

getter must have return value

This error was given because your getter had no return value. Getters return a value and take no parameters.

Header file 'file' is the wrong version (vers-num: Magic Precompiler creates version vers-num); recreate the header with this Magic Precompiler.

You're using a header file created with an older version of the Magic Cap precompiler. Recompile it with the latest version.

M, N, O, P, Q, R

These are Magic Cap linker error messages that begin with *M*, *N*, *O*, *P*, *Q*, or *R*.

Magic Precompiler could not find a critical resource

Some resources are missing from the precompiler's plugin tool. Replace the corrupted plugin with one from the CodeWarrior Tools CD.

mix-in class '_subMix' requires that you inherit from '_aMix'

The Magic Precompiler generated this error when it noticed two things. First, the class being defined uses a mix-in. Second, that the class does not inherit from the class that the mix-in was meant to work with.

NOTE: *Mix-ins are meant to work with specific classes and all their descendants (sub-classes).*

operation '_op' must not have optional parameters unless it is intrinsic

This error appears because operations cannot use C optional parameters, only intrinsics may do this.

operation '_op' must not have return value > 4 bytes
operation '_op' returns a structure

This error appears because operations cannot return values larger than 4 bytes.

Fix Structures, floats, etc. must be passed to the caller using pointers as the return value, or by passing and returning pointers in the operation's parameter list.

Out of memory

This error appears when the Magic Precompiler has exhausted all memory resources.

S, T, U, V, W, X, Y, Z

These are Magic Cap linker error messages that begin with S, *T, U, W, X, Y,* or *Z*.

setter must have exactly one parameter

This error occurs when a setter parameter is incorrect.

NOTE: *Setters take a single parameter and return no results.*

setter must have no return value

This error occurs when a setter attempts to return a value.

NOTE: *Setters take a single parameter and return no results.*

syntax error

The Magic Precompiler generated this error when it encountered incorrect syntax in a class or instance definition file.

two classes both have the name ‘className’

The Magic Precompiler encountered two classes with the same name in your class definition files.

NOTE: *Classes must have unique names.*

types in packages not supported (did you include Types.Def?)

The Magic Precompiler gave this error when it encountered a Magic Cap type definition.

Fix You either included `Types.Def` in your package classes or instances definition file or you're attempting to create new types.

you must include a precompiled, system classes file for a package build

This error is displayed when you attempted a build without the precompiled system classes file.

Fix To build a package, a precompiled system classes file is required. This can be set in the Magic Precompiler preferences panel.

you must not include a precompiled, system classes file during a precompile of the system classes

This error message is generated when you included a precompiled system classes file during a precompile.

Fix When building a precompiled system classes file, do not include another precompiled system classes file in the Magic Precompiler preferences panel.

6

Pascal Compiler Error Messages

This chapter gives an alphabetical list of the most common compiler errors which may be encountered while using Metrowerks CodeWarrior compilers for both the PowerPC-based and 68K-based Macintosh when using the Pascal programming language.

Pascal Compiler Errors

In this list, errors with variable initial text (such as a class or function name) come first. Errors beginning with a non-alphabetic symbol character come next. After that, errors are listed alphabetically.

Punctuation

These are Pascal compiler error messages that begin with punctuation marks.

‘.’ expected

A period is missing at the main `BEGIN` `END` block of a source file.

Fix Make sure the final `END` statement is properly spelled. Make sure the final `END` statement is followed by a dot (`.`) instead of a semi-colon (`;`).

‘..’ expected

The compiler can’t translate an improper array declaration. Pascal array declarations need beginning and end subscripts separated by two dots (`..`) (Listing 6.1).

Listing 6.1 Example array declarations

```
TYPE
  GoodArrayType = ARRAY [1 .. 10] OF REAL; { OK }
  BadArrayType = ARRAY [10] OF REAL; { Error }
```

‘,’ expected

The compiler can’t find a comma (`,`) to separate parameters in a routine declaration or call.

‘;’ expected

A semi-colon is missing at the end of a statement (Listing 6.2).

Listing 6.2 Example semi-colon error

```
VAR
  b : CHAR;       { OK }
  c : REAL        { Error }
  d : INTEGER;
```

‘:’ expected

The compiler expected a colon (`:`) to denote the data type of an identifier (variable, object, parameter, or routine)

‘:=’ expected

You’ve attempted to assign a value to a variable without the assignment operator (Listing 6.3).

Listing 6.3 Example of an incorrect variable assignment

```
  c := 3; { OK }
  c = 3;  { Error, “=” is a comparison operator }
```

‘=’ expected

The compiler expected the equality comparison (=) operator in a boolean expression.

‘[’ expected

The compiler expected an opening left bracket to specify an ordinal range.

‘]’ expected

A closing right bracket is missing (Listing 6.4).

Listing 6.4 *Example of a missing right bracket*

```
VAR
  c : ARRAY [1 .. 10] OF INTEGER;

BEGIN
  c[1] := 6; { OK }
  c[2  := 28; { Error }
```

‘(’ expected

The compiler expects to find an opening left parenthesis to begin an expression.

‘)’ expected

The compiler expects to find a closing right parenthesis to end an expression (Listing 6.5).

Listing 6.5 *Example of a missing closing parenthesis*

```
  b := cos(c);  { OK }
  a := sin(b;  { Error }
```

‘...’ can’t be used in this context.

You are using an ellipsis incorrectly.

‘var’ doesn’t start a variant list

You’re initializing a variant record and using using an identifier that cannot start the variant part. For example:

Listing 6.6 Initializing a variant record

```
rect = RECORD
  foo : integer;
  CASE bar : boolean OF
    true : (a,b: char);
    false: (i:integer);
END;

rec1: rect = (foo:1, bar:true, a:'a', b:'b');
    { OK }
rec2: rect = (foo:2, a:'a', b:'b');
    { ERROR: Didn’t initialize bar }
rec3: rect = (foo:3, bar:false, a:'a', b:'b');
    { ERROR: If bar is false, you must  }
    {        initialize i, not a and b. }
```

$error

(Warning) Your code contains the $error directive, which prints a user-defined warning message to the Messages Window.

A, B, C

These are Pascal compiler error messages that begin with *A*, *B*, or *C*.

a CONST parameter cannot be modified

You tried to modify a parameter declared `CONST`.

a CONST parameter cannot be passed to a VAR parameter

You cannot pass a `CONST` parameter to a VAR parameter since the compiler cannot ensure that the `CONST` parameter will remain unchanged.

a standard routine cannot be assigned

You cannot assign a standard routine to a procedure's parameter. A standard routine is one of Pascal's built-in routines, like abs. For example:

Listing 6.7 Assigning a standard routine to a procedure's parameter

```
procedure foo ( function a ( i : integer ) :
       integer );
begin
end;

{ . . . }

foo(abs);  { ERROR: cannot assign the built-in
                    function abs to a procedure's
                    parameter                    }
```

already declared

You've redeclared a variable that has already been declared in the current scope (Listing 6.8).

already declared as a data field

You already used this identifier as a data field for this class or one of its ancestors.

Listing 6.8 Example of the already declared error

```
VAR
  i : INTEGER;
  c : CHAR;
  i : REAL;
{ Error, i already declared as an INTEGER }
```

already declared UNIT

You've attempted to use two different files with the same `UNIT` name.

assignment to loop index variable

The compiler is warning you that the variable used as an index in a `FOR`...`DO` loop is assigned a value in the loop. Doing so might prevent the loop from terminating (Listing 6.9).

TIP: *To make the compiler warn about this condition, select* **Modified For-loop Indexes** *in the Pascal Warnings preferences panel.*

Listing 6.9 Example of loop index assignment

```
FOR e := 1 TO 10 DO
  e := 1; { Warning: e is the loop index }
```

bad symbol file format

This error is generated when the precompiled unit interface is corrupted.

NOTE: *This error applies for versions prior to CW5 only*

bad type for case label

This error is given when the `CASE` statement label type is not an ordinal value.

'BEGIN' expected

The compiler expects to find the beginning of a compound statement here.

by-value string parameter could be mangled

(Warning) The string you're passing to a routine is longer than the routine expects. Since this string parameter is a value parameter, copying the string could corrupt memory.

cannot be a var parameter

The compiler generated an error message because this expression cannot be passed to a by-var parameter, i.e. cannot take its address as in Listing 6.10.

Listing 6.10 Cannot be a var parameter

```
procedure foo(var bar:longint);
...
foo(3+5);
```

cannot find file

This error is generated when the compiler could not find a file while compiling or linking a project.

See Also "already declared UNIT" on page 126.

cannot mix short-circuit and non short-circuit logical operations

An error was generated because you cannot mix short-circuit operators (&,`|`) with non short-circuit operators (`and`,`or`) within the same expression

cannot pack this type

An error message was given because this type declaration cannot be made a packed type

can't be a routine type

You cannot assign a parameter of a routine type to a variable of a procedural type. The parameter may represent a nested routine, but variables of a procedural type cannot represent nested routines.

Listing 6.11 Can’t be a routine type

```
type intfunc = function ( i : integer ) : integer;

procedure foo ( function a ( i : integer ) :
    integer );
var x : intfunc;
begin
    x := a;   // ERROR
end;
```

can’t be an open array base type

You cannot declare an open array (that is, an array with no upper limit) of this type; for example, an open array of files.

can’t initialize an occurrence of this type

You cannot perform static initialization on an variable of this type.

Can’t override ‘objectMethod’.

The compiler generates this error message because this method cannot be overridden

can’t use ‘^’ for procuedural types

Don’t create a pointer to a procedure with the ^ operator. Just use the procedure’s name itself. For example:

Listing 6.12 Can’t use ^ for procedural types

```
TYPE fooproc = ^procedure;    // ERROR
     fooproc = procedure;     // OK
```

case constant defined more than once

A value in a CASE statement is repeated (Listing 6.13).

Listing 6.13 Case constant defined more than once error

```
...
CASE i OF
  1     : x := Sin(x);
  2     : x := Cos(x);
  3     : x := Exp(x);
  2     : y := 0.0;
        { Error: 2 is already defined. }
  ...
```

class 'identifier' already declared external

You declared the class to be external and you cannot redefine it.

class 'identifier' was declared forward or external

An ancestor class was declared forward or external and hasn't been fully defined yet. You must fully define a class before inheriting from it.

compiler restriction

This language feature is not supported by the Metrowerks Pascal compiler.

constant overflow

An error was encountered when the value represented by this constant exceeds the internal representation available. An example would be trying to fit 4 000 000 000 into a longint.

could not be assigned to a register

The compiler cannot assign this variable or parameter to a register. This error only occurs in in-line assembler routines.

constant string too long

The compiler generated this error message because the literal string contains to many characters. The strings length is greater than or equal to 256 characters.

D, E, F

These are Pascal compiler error messages that begin with *D*, *E*, or *F*.

division by 0

You've attempted a division operation with 0 as the divisor. Division by zero is illegal.

'DO' expected

The `DO` keyword is missing in a `WHILE` or `FOR` statement (Listing 6.14).

Listing 6.14 'DO' expected error

```
FOR a := 1 TO 10 DO { OK }
  b := b + a;

WHILE b > 100 { Error: no DO keyword }
  b := b DIV 2;
```

'END' expected

An `END` keyword is missing in a `BEGIN-END` statement block.

Fix Make sure the `END` statement is properly spelled or that the `END` statement intended to finish a `BEGIN-END` statement block is not "used" by another `BEGIN-END` block.

end of line expected

There is unnecessary text at the end of this line.

Exit statement needs a routine name

The `Exit` procedure requires an argument: the name of the routine to exit or `PROGRAM` to exit the program.

expression type must be boolean

A non-boolean expression is used where a boolean expression is expected as in Listing 6.15.

Listing 6.15 Expression type must be boolean error

```
IF (a + 10) <> 0 THEN { OK }
  b := 50;

IF (a + 10) THEN
      { Error: not a boolean expression }
  b := 50;
```

'EXTERNAL' expected

The compiler gave this error message because the `EXTERNAL` directive was expected in this declaration as in Listing 6.16.

Listing 6.16 Example of 'EXTERNAL' expected.

```
procedure foo; c; external; not extern
```

extra ',' in parameters declaration

An error was given when the routine call contains an extra comma (`,`) in the argument list as in Listing 6.17.

Listing 6.17 Example of extra , in parameters declaration

```
foo(bar,);
```

‘FILE’ expected

An error was given because this parameter must of a file type.

file name expected

The compiler gave this error when a file name was expected.

file not allowed in this context

This is an error because this component defines a file which is illegal in this context, as in Listing 6.18.

Listing 6.18 *Example of File not allowed error*

```
foo = record phyle : file; end;
```

function already has a stackframe

This assembler error message is given when, a stackframe is already defined for the routine.

function doesn’t return any value

A function does not have an assignment statement that assigns a value to the function name identifier (Listing 6.19). To specify the value a function returns, the function’s identifier must be assigned a value as if it were an ordinary variable.

Listing 6.19 *Function doesn’t return any value errors*

```
FUNCTION InchesToCms(inches : REAL) : REAL;
VAR
  temp : REAL;

BEGIN
 { Error: no assignment to InchesToCms identifier }
  temp := inches * 2.54;
END;
```

```
FUNCTION ChickenCount(hatchedEggs : INTEGER) :
    INTEGER;
BEGIN
  { Error: no assignment if condition is false }
  IF hatchedEggs = 0 THEN
    ChickenCount := 0;
END;
```

function has no initialized stackframe

This assembler error message is given because a stackframe must be defined for the routine.

G, H, I

These are Pascal compiler error messages that begin with *G, H,* or *I*.

goto a label enclosed in a 'FOR' statement.

This is an error because this label is the target for a `goto` statement from outside the `FOR` loop as in Listing 6.20.

Listing 6.20 Example of goto a label enclosed in FOR statement.

```
goto 1;
for <var> := <expr> to <expr2> do
1:
```

goto a label enclosed in a ‘WITH’ statement

This error message is given when the label is the target for a `goto` statement from outside the `WITH` statement. An example of this error is in Listing 6.21.

Listing 6.21 Example of goto a label enclosed ina WITH statement.

```
goto 1;
with <expr> do begin
  ...
  1:
  ...
  end;
```

goto between ‘CASE’ legs.

An error is given when the `goto`'s target is inside another `CASE` label, as in Listing 6.22.

Listing 6.22 Example of goto between CASE legs.

```
case <expr> of
  lab1 : begin
    ...
    goto 1;
    ...
    end;
  lab2 : begin
    ...
    2:
    ...
    end;
  end
```

goto between ‘IF’ and ‘ELSE’ parts.

An error is generated when the `goto`'s target is between an `IF` and `ELSE` as in Listing 6.23.

Listing 6.23 *Example of goto between IF and ELSE parts.*

```
if <expr> then begin
  ...
  goto 1;
  ...
  end
else begin
  ...
  1:
  ...
  end
```

identifier expected

The error message is given when the compiler expects to find an identifier.

illegal addressing mode

This assembler message is generated because this addressing mode is illegal for the operation

Illegal cast, size mismatch.

You tried to cast a variable of a structured type to another structured type of a different size. The structured types must be the same size.

illegal casting

This is an error because of an inappropriate attempt to use a value of one data type as another data type (Listing 6.24).

Listing 6.24 *Illegal casting error*

```
VAR
  a : LONGINT;
  b : ARRAY [1..3] OF CHAR;
  i : REAL;

BEGIN
  ...
  a := LONGINT(i); { OK };
  a := LONGINT(b); { Error }
  ...
```

illegal declaration

This is an error because it is illegal to use this declaration in this context as in Listing 6.25.

Listing 6.25 *Example of Illegal declaration.*

```
procedure foo ( ... ) ; c; external;
```

illegal format for real constant

An error was given because this number doesn't correspond to the notation accepted for a floating point number.

illegal function result type

This error is given when a function cannot return an item of this type, as in Listing 6.26.

Listing 6.26 Example of illegal function result type.

```
function foo : FILE;
```

illegal instruction for this processor

This assembler message is given for an unrecognized instruction.

illegal label

The compiler gives this error when the label declaration is illegal.

illegal operand

You cannot use this expression as an operand.

illegal operand type

This is an error because this operator doesn't allow operands of this type.

illegal operands for this processor

This assembler error message is generated because this kind of operand is not accepted by the processor.

illegal operation

An error is generated because this operation cannot be performed, as in Listing 6.27.

Listing 6.27 Example of illegal operation.

```
i/0 (division by literal zero)
```

illegal register list

This assembler error message is given because this instruction doesn't accept this register list.

illegal set base type

An error is given when the type cannot be used as a base to construct a set, as in Listing 6.28.

Listing 6.28 Example of illegal set base type.

```
set of real
```

illegal statement

This is an error because the statement is illegal, as in Listing 6.29.

Listing 6.29 Example of illegal statement.

```
if <expr> then
    procedure
```

illegal symbol in declaration

You have attempted to declare a variable, but its identifier is incorrect or unknown. An example of this error is in Listing 6.30

Listing 6.30 Illegal symbol in declaration error

```
VAR
  x9 : REAL;                  { OK }
  _ripe : CHAR;                 { OK }
  123e : INTEGER;
                { Error: can’t start with number }
```

illegal symbol in factor

You have included an illegal or unknown symbol in your statement, as in Listing 6.31

Listing 6.31 illegal symbol in factor

```
  x := +-85
  if a==2 then b = 3.0; (* the a==2 is illegal*)
```

illegal usage in this scope

An error message was given because an illegal operation was performed within the scope, as in Listing 6.32.

Listing 6.32 Example of illegal usage in this scope.

```
for i : <expr1> to <expr2> do
    call(i)  { where the parameter is by-var }
```

illegal usage of a selector

This is an error because `[`, `.`, `^` can only occur with the corresponding types, as in Listing 6.33.

Listing 6.33 Example of illegal usage of a selector.

```
i : integer;
  i^ := 4;
```

illegal use of inline function

This is an error because this function can not be defined as inline.

illegal use of keyword

This is an error because this keyword is illegal in this context.

‘INHERITED’ must be used in a method definition.

An error is given because the inherited directive can only occur in a method definition, not in an ordinary routine definition. Listing 6.34 is an example of this error.

Listing 6.34 Example of INHERITED must be used in a method definition.

```
procedure foo;
  begin
    inherited bar;
  end;
```

INLINE is not supported in PPC

You tried to use inline opcode routines in PowerPC code. The PPC compiler doesn’t support this feature.

‘INTERFACE’ expected

This is an error because the interface part of a `unit` is missing.

internal compiler error

This error message was generated when the compilation halted. The compiler is functioning improperly.

WARNING! *If this error is generated please contact Technical Support by sending in a bug report form. Please include as much information as to when and how the error occurred as possible.*

J, K, L

These are Pascal compiler error messages that begin with *J*, *K*, or *L*.

label error

This is an error because a label must be a number in the range of 1 to 10000

label's goto is not within 'FOR' statement.

An error was caused when a `goto` inside the `FOR` statement branched outside the loop.

label's goto is not within 'WITH' statement.

This is an error because a `goto` inside the `WITH` statement branches outside the statement.

label range error

An error is given when the case label ranges lower bound is larger than the upper one. Listing 6.35 is an example of this error.

Listing 6.35 *Example of label range error.*

```
case <expr> of
    5..3 :
    end;
```

local data > 224 bytes

You must create a stack frame for this routine, since it has more than 224 bytes of local variables. For more information, see "The Built-In Assembler" in the *Pascal Language Manual*.

local variables size > 32K

The total amount of memory used to allocate local variables has exceeded 32Kbytes. This error often occurs when declaring arrays that are too large (Listing 6.36).

Listing 6.36 *Local variables size > 32K error*

```
PROCEDURE BoomArray();

VAR
  { Error: this local array is
    greater than 32Kbytes     }
  wayTooBig : ARRAY [1 .. 100000] OF INTEGER;
```

Fix Make some of the variables global instead.

M, N, O

These are Pascal compiler error messages that begin with *M*, *N*, or *O*.

macro name is already defined

You already defined a macro with this name.

macro level error

You exceeded the maximum nesting level for macros. You cannot nest them more than 32 deep.

method not declared in '*'.

This error was given because this method wasn't declared in this class.

missing array initializer

The array initializer doesn't contain enough elements.

missing '$IFC' directive

The compiler encountered a `{$ELSEC}` or `{$ENDC}` directive without first finding a matching `{$IFC}` directive.

must be an array

This is an error because this variable's type must be an array.

must be assignable

The left hand side of an assignment statement can't be assigned a value. Listing 6.37 shows an example.

Listing 6.37 *Must be assignable error*

```
CONST
  a = 10;

BEGIN
  a := 20;
{ Error: constant values can't be reassigned }
...
```

must be a constant

You have attempted using an expression as a constant that is not a constant expression. Listing 6.38 shows an example.

Listing 6.38 *Must be a constant error*

```
CONST
  b = 10; { OK }
   x = func();
{ Error: function call is not allowed }
  ...

CASE finalValue OF
  1 : f := 10; { OK }
  b : f := 2;
{ Error: b is not a constant expression }
  ...
```

must be a function

A procedure is called where a function is expected (Listing 6.39).

Listing 6.39 *Must be a function error*

```
FUNCTION Func() : INTEGER; FORWARD;
...
PROCEDURE Func();
```

```
{ Error: expected a function definition }
BEGIN
END;
```

must be an object type.

This is an error because this variable's type must be a class.

must be an open array parameter

You used the `hi` function on an open array ; that is, an array declared with no upper limit.

must be a pointer

This is an error because this variable's type must be a pointer.

must be a procedure

A function is used where a procedure is expected (Listing 6.40).

Listing 6.40 Must be a procedure error

```
PROCEDURE Func() : INTEGER; FORWARD;
...
FUNCTION Func();
{ Error: expected a procedure definition }
BEGIN
END;
```

must be a range

An error was given because this variable's type must be a range.

must be a record

This is an error because this variable's type must be a record.

must be a scalar

An error was given because this variable's type must be an ordinal type or an enumeration.

must be a text file.

An error was given because this variable's type must be a file of type 'text'.

must be a type

This is an error because this identifier was not declared as a type.

must be a variable

This is an error because this identifier was not declared as a variable,

no parameter list

An error was generated when the parameter list was missing for this routines call.

not in program parameters

This is an ANS Pascal error message. It is generated when the file was not declared in the program header.

number is out of range

This is an error because the literal number is out of the range defined for the variable/field/parameter.

number overflow

The compiler is signalling an attempt to assign a numeric constant value to a variable that’s greater than the maximum allowed for that data type. Listing 6.41 give an example of this error

Listing 6.41 Number overflow error

```
VAR
  a : INTEGER;

BEGIN
  a := 10000000000000000000000000;
{ Error: too big! }
  ...
```

object cannot contain file component

This is an error because a file cannot be made out of classes.

object not printable

This is an error because it's impossible to print the value of this type.

‘OF’ expected

The compiler generates this message because the code is missing the keyword `OF` in a `CASE` statement

‘OTHERWISE’ expected

This error message occurs when the code is missing the keyword `OTHERWISE` in a `CASE` statement

out of range

An error message is given when the value is out of the range defined for the variable, field, or parameter.

P, Q, R

These are Pascal compiler error messages that begin with *P*, *Q*, or *R*.

parameter mismatch

The parameters in a routine’s definition and declaration do not match (Listing 6.42).

Listing 6.42 Parameter mismatch error

```
PROCEDURE Func(a : INTEGER); FORWARD;
...

PROCEDURE Func(a : CHAR); { Error: parameter
doesn’t match }
BEGIN
...
END;
```

parameter missing

A routine call does have enough parameters to match the routine's definition (Listing 6.43).

Listing 6.43 *Parameter missing error*

```
FUNCTION Times(i : INTEGER; factor : INTEGER) :
    INTEGER;
BEGIN
  ...
END;

BEGIN
  x := Times(12); { Error: need 2 parameters }
...
```

parameter's size is too small, could mangle memory

(Warning) The string you're passing to a routine is smaller than the routine expects. Since the string parameter is a VAR parameter, copying the string could mangle memory.

preprocessor nesting too deep

The compiler gives this error when there are too many nested compilation directives.

procedural variable can't get nested routine.

This is an error because you cannot assign a local routine to a procedural type variable

procedure already FORWARD

The `FORWARD` directive is used to define the same procedure twice. Listing 6.44 gives and example of this error.

Listing 6.44 *Procedure already FORWARD error*

```
PROCEDURE NoGood(); FORWARD;
...
```

```
PROCEDURE NoGood(); FORWARD;
{ Error: already defined }
```

‘PROGRAM’ expected

You must start a source file with either `UNIT` or `PROGRAM`.

program parameter redeclaration

This ANS Pascal error message is given when the file was already declared as in Listing 6.45.

Listing 6.45 *Example of program parameter redeclaration.*

```
program (output, output);
```

range base type mismatch

An error was given because it is impossible to construct a range over this type. Listing 6.46 gives an example of this error.

Listing 6.46 *Example of range base type mismatch.*

```
fprange = 1.0 .. 1.5;
```

recursion in macro not allowed

You cannot create a macro that uses recursion.

redeclaration of routine ‘routineName’

A function or procedure, named *routineName,* is declared more than once in the same scope.

redundant symbol

An error is given because this character is illegal in Pascal.

result type mismatch

The compiler generated this error message because this function's type doesn't match the declaration's type.

routine ‘identifier’ declared but undefined.

You did not implement a routine that you declared in the interface or declared `forward`.

S, T

These are Pascal compiler error messages that begin with *S*, *T*.

scalar value expected

The compiler expects a scalar value, such as a boolean, enumeration, range, char, or integer.

Segmentation directive must be placed after ‘IMPLEMENTATION’.

Segmentation directives, `{$S` *segmentName*`}`, cannot be placed in the interface part of a unit. Instead, place segmentation directives in the implementation part.

set base type mismatch

This is an error because the base type of these sets are not compatible, as in Listing 6.47.

Listing 6.47 *Example of set base type mismatch.*

```
set of colors and set of chars
```

size mismatch for universal parameter

This error is given when UNIV parameters are not the same size. UNIV parameters must match in sizes.

string too long for assignment

The string literal assigned to a string variable is longer than the string variable’s length (Listing 6.48).

Listing 6.48 String too long for assignment error

```
VAR
  s : STRING[10];

BEGIN
  { Error: s can only be 10 characters long }
  s := 'abcdefghijklmnopqrstuvwxyz';
...
```

subrange type expected

You must use an ordinal type subrange in this context.

'THEN' expected

The `THEN` keyword is missing in an `IF` statement (Listing 6.49).

Listing 6.49 'THEN' expected error

```
IF a = 'b' THEN b := a; { OK }
IF x = 0 { Error: no THEN keyword }
  y := 100;
```

'TO' or 'DOWNTO' expected

A `TO` or `DOWNTO` keyword is missing in a `FOR` statement (Listing 6.50).

Listing 6.50 'TO' or 'DOWNTO' expected error

```
FOR j := 1 TO 10 { OK }
  k := k * 2;

FOR i := 0 DO { Error: no TO or DOWNTO clause }
```

Too many array initializers

The array initializer contains too many elements.

Too many include files

This error message is displayed when the number of include files exceeds the capacity of the compiler.

Too many nested directives

This error message is generated when the compiler encounters too many levels of nesting.

Too many opcodes for inline routine

The compiler generates this error message when the inline routine is too long.

type expected

A data or object type is missing in a variable declaration (Listing 6.51).

Listing 6.51 Type expected error

```
VAR
  a : CHAR; {OK }
  i : ;  { Error: missing type }
```

type ‘identifier’ had a forward declaration, the compiler cannot change it to ‘identifier’

You didn’t define the type, and the compiler cannot change a variable of that type to a new type. To allow this type coercion turn on the **Relax pointer compatibility** option in the Pascal Language settings panel.

type mismatch

The type of a variable differs from the expression it is being assigned. Also, the type of an item in an expression is not compatible with the other types used in the expression. An example of this error is in Listing 6.52

Listing 6.52 Type mismatch error

```
  a := 'A';
  b := 'a' - 'A';
{ Error: characters aren’t numeric }
```

U, V, W, X, Y, Z

These are Pascal compiler error messages that begin with *U, V, W, X, Y,* or *Z*.

unclosed comment

A comment has not been terminated (Listing 6.53).

Listing 6.53 Unclosed comment error

```
{ Error: unclosed comment error }
{$I test.p
VAR
  i, j : INTEGER;
...
```

undeclared identifier

A variable or constant name is used in source code, but has not been declared.

undefined identifier

An error occurred when this identifier is undefined.

undefined label ‘label-number’

A label has been declared or referenced in a `GOTO` statement but is not used in the source code (Listing 6.54).

Listing 6.54 Undefined label error

```
LABEL 1000;

VAR
  i, j : INTEGER;

BEGIN
  i := 10;
  IF i > 5 THEN GOTO 1000
      { Error: 1000 never used }
END;
```

undeclared label

You must declare a label before you use it. You didn't declare this label within the scope.

undefined pointer 'identifier'

An error occurred when this identifier was assumed to be a pointer but is still undefined.

unexpected end of file

The compiler reached the end of a source code file before it could read the terminating `END` statement.

Fix Add the terminating statement to the source code file.

unknown PowerPC instruction mnemonic

This is an unknown assembler instruction for the PowerPC assembler.

'UNIT' expected

The `UNIT` expected error message is issued if neither `program` or `unit` is the first keyword of the file.

‘UNTIL’ expected

This `REPEAT` statement is missing its `UNTIL` clause. Every `REPEAT` statement must end with an `UNTIL` clause.

unit wasn‘t compiled

The unit being referenced in the project cannot be found within the project. This error often occurs when a `UNIT`’s name and its `filename` don’t match. By default, Metrowerks Pascal requires that the name in a unit’s `UNIT` statement and the unit’s filename (without the filename extension) be the same.

Fix Add the unit or the library that contains the unit to the project, or change the unit’s filename to match the name used in the unit’s `UNIT` statement.

unresolved forward class reference to ‘identifier’.

An error occurred because this identifier was assumed to be a class but is still undefined.

unresolved external class reference to ‘identifier’

The compiler cannot create an instance of an external class since it doesn't know the class’s size.

unsafe object reference.

An error is generated when the reference to an object's field is unsafe. The Memory Manager may move the object.

unterminated string

A string literal doesn’t have an ending quote. For example of this error see Listing 6.55

Listing 6.55 Unterminated string error

```
CONST
  { Error: no ending quote character }
  kDefaultTitle = 'Rabbit Food Accounting Package;
  ...
```

unused variable

The compiler is warning that a variable has been declared, but it is never used in its scope (Listing 6.56). To make this warning active, select **Unused Variables** in the Pascal Warnings settings panel.

Listing 6.56 Unused variable warning

```
FUNCTION ChunkCount(a : INTEGER) : INTEGER;

VAR
  SockStr : STRING[100]; { Warning: SockStr never
used }

BEGIN
  ChunkCount := a * 10
END;
```

value is not stored in register

This assembler error message is given when the value is not in a register, it must be in a register.

variable '*' is a loop index

This is an error because you cannot use a for loop index for this purpose.

variable used but not initialized

A variable is used in an expression without first being assigned a value (Listing 6.57).

Fix Assign a value to the variable before using it in an expression.

Listing 6.57 Variable used but not initialized warning

```
VAR
  i, j, k : INTEGER;

BEGIN
  j := i * 10; { Error: i isn't initialized yet }
  IF (Fib(i) > 100 THEN
    { Error: i isn’t initialized yet }
    k := 1;
END.
```

About CodeWarrior Documentation

Information about the people who worked on this documentation and references to other documentation you'll find useful.

About Error Reference

engineering:	Marcel Achim, Mark Anderson, Berardino Barrata, Andreas Hommel, Paul Lalonde, John McEnerney Michael Marcotty, Sylvain Vergnil
writing:	Ron Liechty, Jeff Mattson
frontline warriors:	Khurram Qureshi and the Metrowerks Technical Support team

Guide to Other CodeWarrior Documentation

If you need information about...	See this
Installing updates to CodeWarrior	*QuickStart Guide*
Getting started using CodeWarrior	*QuickStart Guide*; *Tutorials* (Apple Guide)
Using CodeWarrior IDE (Integrated Development Environment)	*IDE User's Guide*
Important, last minute, information on new features and changes	*Release Notes* folder
Creating Macintosh and Power Macintosh software	*Targeting Mac OS*; *Mac OS* folder
Creating Microsoft Win32/x86 software	*Targeting Win32*; *Win32/x86* folder
Creating Magic Cap software	*Targeting Magic Cap*; *Magic Cap* folder
Using the ToolServer with the CodeWarrior editor	*Targeting Mac OS*
Controlling CodeWarrior through AppleScript	*IDE User's Guide*
Using CodeWarrior to program in MPW	*Command Line Tools Manual*
C, C++, or 68K assembly language programming	*C, C++, and Assembly Reference*; *MSL C Reference*; *MSL C++ Reference*
Pascal or Object Pascal programming	*Pascal Language Manual*; *Pascal Library Reference*
Fixing compiler and linker errors	*Errors Reference*
Debugging	*Debugger Manual*
Fixing memory bugs	*ZoneRanger Manual*
Speeding up your programs	*Profiler Manual*
PowerPlant	*The PowerPlant Book; PowerPlant Advanced Topics;* PowerPlant reference documents
Creating a PowerPlant visual interface	*Constructor Manual*
Learning how to program for the Mac OS	*Discover Programming for Macintosh*
Learning how to program in Java	*Discover Programming for Java*
Contacting Metrowerks about registration, sales, and licensing	*Quick Start Guide*
Contacting Metrowerks about problems and suggestions using CodeWarrior software	*email Report Forms* in the *Release Notes* folder
Sample programs and examples	*CodeWarrior Examples* folder; *The PowerPlant Book*; *PowerPlant Advanced Topics; Tutorials* (Apple Guide)
Problems other CodeWarrior users have solved	*Internet newsgroup [docs]* folder
Getting more information from CodeWarrior documentation	*CodeWarrior Cross-Reference*

960926 1804