Build Tools Message Reference Manual

Document Number: CWSCBTMREF
Rev. 10.9.0, 06/2015

Contents

Section number **Title** **Page**

Chapter 1
Introduction

Chapter 2
Compiler Front-end

Chapter 1
Introduction

The Build Tools Message Reference Manual documents the messages generated by your CodeWarrior build tools.

This information helps you fix problems the build tools encounter as they build your projects.

Note that this manual documents just those messages that are generated by compiler, assembler, and linker.

In this chapter:

1.1 Accompanying Documentation

The **Documentation Roadmap** page describes the documentation included in this version of *CodeWarrior Development Studio for StarCore 3900FP DSP Architectures*. You can access **Documentation Roadmap** by:

- a shortcut link on the Desktop that the installer creates by default, or
- opening `START_HERE.html` in `CWInstallDir\SC\Help` directory.

Chapter 2
Compiler Front-end

This chapter documents the messages generated by the compiler's front-end (FE) component.

For easy reference, these messages are categorized into the following symbolic and alphabetical categories:

2.1 Symbols

In this category:

- '{' expected
- '}' expected
- '<' expected
- < expected (you may have accidentally used a <: token)
- '>' expected

2.2 A-F

In this category:

- => Trying with application file one: %s
- a cycle in the call graph is overriding this function's always_inline request
- A pointer/array type was expected for this operation instead of typename
- allocation/deallocation functions shall be global scope or class members
- alternates in a single argument to gcc style inline assembler exceeded maximum number supported (number)
- ambiguous ?: expression, typeA can be converted to typeB and vice versa
- ambiguous access from type to type
- ambiguous access to class/struct/union member
- ambiguous access to name found 'symbol1' and 'symbol2'
- ambiguous access to overloaded function
- ambiguous use of partial specialization
- argument expected while expanding macro name (got number, wanted number
- arguments to gcc style inline assembler exceeded maximum number supported (number)
- ASCII shift state expected
- assigning a non-int numeric value to an unprototyped function
- branch out of range
- call of non-function
- calling convention ignored due to incompatible compiler options
- cannot construct type's base class type
- cannot construct type's direct member name
- cannot convert typeA to typeB
- cannot create object file filename
- cannot delete pointer to const
- cannot destroy const object
- cannot enable instance manager here
- cannot execute instrumenter name (name) (command line: name) name
- cannot find matching deallocation function for variable

- cannot instantiate obj
- cannot load main source file filename
- cannot locate instrumenter name (PATH=pathname, AMC_HOME=name)
- cannot mix structured exception handling and C++ exception handling
- cannot modify name after declarations have begun
- cannot pass const/volatile data object to non-const/volatile member function
- cannot pass 'void' or 'function' parameter
- cannot query option name
- cannot throw class with ambiguous base class (name)
- cannot use precompiled header name while instrumenting
- cannot write file '%u' (%u)
- case constant defined more than once
- case has an empty range of values
- 'catch' expected
- character cannot be represented in a 'long long'
- character is out of range
- class has no default constructor
- class type expected
- classname is not a class name
- combining character detected at beginning of identifier
- compatible IDB list changed; you may want to remove object code and rebuild this target
- complex types are not implemented
- component Compiler: Cannot find component %s => Skipping...
- component Compiler: Cannot find arg line component %s
- component %s already exists, Appending information to existing definition
- const member aVar is not initialized
- 'const' or '&' variable needs initializer
- constness casted away
- could not create or write file name for instrumentation (name)
- could not generate intrinsic name due to incompatible arguments or compiler options
- could not read file name for instrumentation (name)
- data object object redefined
- data type is incomplete
- declaration syntax error
- declarator expected
- default label defined more than once
- deleting a void pointer is undefined
- empty array must be last class/struct member
- end of line expected
- exception handling option is disabled

- exception specification list mismatch
- expression has no side effect
- expression syntax error
- expression too complex
- extended universal character used in identifier
- function already has a stackframe
- function call does not match prototype
- function call name is ambiguous
- function defined 'inline' after being called
- function has no prototype
- function hides inherited virtual function func2
- function nesting too complex
- function result is a pointer/reference to an automatic variable

2.3 G-L

In this category:

- GCC constraint name is not supported at this time
- GCC style assembler processing could not match any of the constraints in name
- identifier expected
- identifier name redeclared
- identifier name redeclared was declared as: typeA now declared as: typeB
- ignored attribute name due to conflict with calling convention
- illegal #pragma
- illegal '&' reference
- illegal access from classname to protected/private member membername
- illegal access from name to protected/private base class classname
- illegal access from name to protected/private member class::member
- illegal access qualifier
- illegal access to local variable from other function
- illegal access to protected/private member
- illegal access/using declaration
- illegal addressing mode
- illegal argument list
- illegal array definition
- illegal assignment to constant
- illegal bitfield size
- illegal bitfield type name

- illegal character constant
- illegal class member access
- illegal combination of operands in inline statement at line number
- illegal const/volatile '&' reference initialization
- illegal constant expression
- illegal constructor/destructor declaration
- illegal ctor initializer
- illegal data in precompiled header
- illegal data size
- illegal default argument(s)
- illegal empty declaration
- illegal exception specification
- illegal explicit conversion from type_A to type_B
- illegal explicit template instantiation
- illegal explicit template specialization
- illegal 'friend' declaration
- illegal function definition
- illegal function overloading
- illegal function return type
- illegal implicit const/volatile pointer conversion from typeA to typeB
- illegal implicit conversion from Type_A to Type_B
- illegal implicit enum conversion from Type_A to Type_B
- illegal implicit member pointer conversion
- illegal initialization
- illegal 'inline' function definition
- illegal instruction for this processor
- illegal jump past initializer
- illegal macro argument name argument
- illegal macro name
- illegal macro name; name is a C++ keyword
- illegal multi-line string constant
- illegal multiway transfer out of __try...__finally statement
- illegal name overloading
- illegal namespace
- illegal non-type template argument
- illegal number
- illegal operand
- illegal operand in inline statement at line number
- illegal operand operand
- illegal operands for this processor
- illegal operands operand symbol operand

- illegal operation
- illegal operator
- illegal 'operator' declaration
- illegal operator overloading
- illegal optimization level for this limited version of CodeWarrior
- illegal option name
- illegal or unsupported __attribute__
- illegal or unsupported __declspec
- illegal partial specialization
- illegal partial specialization argument list
- illegal precompiled header compiler flags or target
- illegal precompiled header version
- illegal redefinition of UUID for typename
- illegal register list
- illegal return value in void/constructor/destructor function
- illegal static const member argument initialization
- illegal storage class
- illegal string constant
- illegal struct/union/enum/class definition
- illegal template argument dependent expression
- illegal template argument(s)
- illegal template declaration
- illegal token
- illegal token for integral constant expression
- illegal type
- illegal type qualifier(s)
- illegal universal character
- illegal universal character sequence
- illegal use of #pragma parameter
- illegal use of __declspec(name)
- illegal use of __leave in __try...__except statement
- illegal use of abstract class classname
- illegal use of abstract class classname
- illegal use of asm inline function
- illegal use of C++ feature in EC++
- illegal use of calling convention declarator
- illegal use of const/volatile function qualifier sequence
- illegal use of data qualifier(s)
- illegal use of default template-argument
- illegal use of direct parameters
- illegal use of expression statement outside function

- illegal use of function qualifier(s)
- illegal use of 'HandleObject'
- illegal use of incomplete struct/union/class 'type'
- illegal use of inline function
- illegal use of keyword
- illegal use of namespace name
- illegal use of non-static member
- illegal use of precompiled header
- illegal use of pure function
- illegal use of register variable
- illegal use of specifier
- illegal use of structured exception handling keyword outside of _try statement
- illegal use of 'template' prefix
- illegal use of 'this'
- illegal use of typename
- illegal use of 'void'
- illegal use template argument dependent type 'T::argument'
- illegal UTF-8 sequence, treating as raw bytes
- illegal virtual function functionname in union
- illegal X::X(X) copy constructor
- illegal redefinition of %s segment <%s, %s>
- immediate operand in inline statement at line number cannot span more than 8 bits
- implicit arithmetic conversion from typeA to typeB
- implicit 'int' is no longer supported in C++
- included file filename is spelled differently on disk name
- inconsistent linkage: 'extern' object redeclared as 'static'
- inconsistent use of 'class' and 'struct' keywords
- inline function call functionname not inlined
- input constraints shall not begin with '=' or '+'
- instance variable list does not match @interface
- instrumenter name failed with exit code name (command line: name) name
- integral to pointer conversion
- integral type is not large enough to hold pointer
- internal scanner error
- invalid floating-point constant
- invalid integer constant
- invalid message number
- label Lbl redefined
- local classes shall not have member templates
- local data >32k
- logical segment %s %s is not defined

- 'long long' case range is not supported
- 'long long' switch( ) is not supported
- loss of precision in floating-point constant

2.4 M-R

In this category:

- macro Macro redefined
- macro(s) too complex
- 'main' not defined as external 'int main( )' function
- missing ')' in macro argument list
- more than one expression in non-class type conversion
- multi-byte character constant
- name has not been declared in namespace/class
- name is not a class/namespace name
- name is not a member of class type
- name is not a member of namespace name
- nested comment detected
- no UUID defined for type name
- non-const '&' reference initialized to temporary
- not a struct/union/class
- not an lvalue
- no component LCF found for component %s
- no component name selected, Skipping component...
- number expected
- number is out of range
- object differs from virtual base function object in return type only
- object name hidden by local declaration
- object objectname redefined
- out of memory
- output constraints shall begin with '=' or '+'
- override is declared without 'virtual' keyword
- override of dllimport function symbol only has application scope
- overriding segment setting %s %s in %s
- pascal function cannot be overloaded
- pointer/array required
- possible unwanted ';'
- possible unwanted assignment

- possible unwanted compare
- preceding '#pragma push' is missing
- preprocessor #error directive
- preprocessor #warning directive
- protocol list does not match @interface
- pure function functionname is not virtual
- recursive operator-> delegation
- reference to label lbl is out of range
- result of function call is not used
- RTTI option is disabled

2.5 S-Z

In this category:

- size of type is too large (maximum num bytes)
- size pad byte(s) inserted after data member name
- some instances are missing; please rebuild
- stray character 0x%x (ASCII value) in program
- string expected
- string too long
- struct/union/class size exceeds 32k
- struct/union/enum/class member name redefined
- struct/union/enum/class tag stType redefined
- super class does not match @interface
- system does not support the text encoding name, treating as ASCII
- template argument list expected
- template non-type argument object does not have external linkage
- template parameter mismatch
- template parameter/argument list mismatch
- template redefined
- template too complex or recursive
- the (elided) copy constructor is not callable because the reference parameter cannot be bound to an rvalue
- the file filename cannot be opened
- the file filename cannot be opened
- the local type name cannot be used in template arguments
- the type of the section '%s' is %s, while the definition of variable '%s' expects to be %s

- the type of the section '%s' is %s, while it is expected to be %s in file %s
- the option %s is not supported for application file
- too many errors emitted, quitting
- too many initializers
- too many macro arguments
- type mismatch typeA and typeB
- typename is missing in template argument-dependent qualified type
- typename redefined
- types that are declared in parameter lists go out of scope at the end of the function declaration/definition
- undefined identifier var
- undefined label Lbl
- undefined macro name used in #if or #elif conditional
- undefined preprocessor directive
- unexpected argument while expanding macro name (wanted number)
- unexpected end of file
- unexpected end of line
- unexpected token
- unimplemented assembler instruction/directive
- unimplemented C++ feature
- unions cannot be used as base classes
- unions cannot have base classes
- unions cannot have nontrivial class members
- unions cannot have reference members
- unions cannot have static data members
- unknown assembler instruction mnemonic
- unknown escape sequence
- unknown x86 assembler instruction mnemonic
- unterminated #if / macro
- unterminated comment
- unterminated comment
- using '#pragma once [on]' in a precompiled header
- using implicit copy assignment for class with const or reference member variable
- using non-POD classes in variable argument lists is undefined
- value is not stored in register
- variable could not be assigned to a register
- variable is not a struct/union/class member
- variable length array types can only be used in local or function prototype scope
- variable length arrays cannot be used in function template prototypes or local template typedefs
- variable var is not initialized before being used

- variable/argument name is not used in function
- virtual functions cannot be pascal functions
- whitespace expected after integer constant (at location)

2.6 Compiler Messages in Detail

This chapter describes the messages generated by the compiler's front-end (FE) component in detail.

2.6.1 C/C++ Error 10100

illegal character constant

This error is signaled when an attempt is made to assign an illegal character constant. This error also occurs when a string constant does not terminate before the end of a line. The following code example contains three examples of illegal character constants.

```
// each of these assignments contain illegal character constants

char FooCh;

...

...

FooCh = '';

FooCh = '\x';

FooCh = 'ddjdjdj';
```

Fix

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

2.6.2 C/C++ Error 10101

illegal string constant

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

Fix

The solutions are:

- Terminate the string before the end of a line.
- If the source of the error is not apparent, correct all previous errors and recompile.

2.6.3 C/C++ Error 10102

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. The solutions are:

- Check the line of the source code file in question.
- If the error is not apparent, fix all previous errors and recompile.
- Try preprocessing the file to examine the token stream, especially at end of buffer.

2.6.4 C/C++ Error 10103

unterminated comment

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

Fix

A comment opened with /* must be closed with */

Close the comment and recompile your code.

2.6.5 C/C++ Error 10104

undefined preprocessor directive

This error is signaled when a preprocessor directive not recognized by Freescale C/C++ is encountered.

Fix

Preprocessor directives recognized by Freescale C/C++ are listed in the C, C++, and Assembler Language Reference. Select **Help > Online Manuals** to access these documents.

2.6.6 C/C++ Error 10105

illegal token

The compiler found an illegal preprocessor token.

For example:

```
int a = 1a;
```

Fix

The solutions are:

- Remove the illegal token.
- Fix all previous errors and recompile.

2.6.7 C/C++ Error 10106

string too long

The character string in question is too long. This error message is displayed most often when a terminating quote mark was omitted from the string.

Fix

Turn on the Color Syntax option in the Editor settings panel, find the location of the missing terminating quote mark.

2.6.8 C/C++ Error 10107

identifier expected

The compiler expected to find an identifier, but instead found another token.

```
long Waggle;

long temp, short; // short is placed where identifier should be

Point myPt;
```

Fix

The compiler issues this error when it finds a short where an identifier should be located. 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 and recompile.

2.6.9 C/C++ Error 10108

macro *Macro* redefined

An attempt was made to redefine a macro that has already been defined.

Fix

Remove or rename one of the macros, or insert #undef *Macro* before the redefinition.

2.6.10 C/C++ Error 10109

illegal argument list

An illegal macro argument list was found.

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

Fix

Remove illegal arguments.

```
#define macro(arg)
```

2.6.11 C/C++ Error 10110

too many macro arguments

An attempt was made to define a macro. The macro definition included more than 128 arguments.

```
#define macro(arg1,...,arg129) ...
```

Fix

Rewrite the macro to use 128 or fewer arguments.

2.6.12 C/C++ Error 10111

macro(s) too complex

A macro cannot be expanded because it is too complex (or possibly recursive).

Fix

Redesign the macro with less complexity.

2.6.13 C/C++ Error 10112

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.

2.6.14 C/C++ Error 10113

end of line expected

This error occurs under many circumstances and may be the result of an error on a previous line of code. For example, if you turn on the ANSI Keywords Only option in the 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. The code listing below shows another example where more tokens are 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.

2.6.15 C/C++ Error 10114

'(' 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.

To prevent this error while typing in source code, select the Balance While Typing checkbox in the Editor Settings preference panel.

2.6.16 C/C++ Error 10115

')' expected

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

NOTE

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

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 Settings preference panel. When selected, this preference sounds an alert if an non-matching right parenthesis is typed.

2.6.17 C/C++ Error 10116

',' expected

The compiler did not find a comma where it expected to find one. For example, the compiler expects to find a comma in a function call.

```
void Myfunc(int);

x = Myfunc(1; // right parenthesis is missing
```

The error compiler syntactically expected a comma because a right parenthesis is missing.

Fix

The solutions are:

- Add a parenthesis to end the function call.
- If the source of the error is not apparent, correct all previous errors and recompile.

2.6.18 C/C++ Error 10117

preprocessor syntax error

The compiler encounters an illegal preprocessor directive.

```
#include file
```

Fix

The solutions are:

- Check the syntax of the directive in question.
- If the source of the error is not apparent, correct all previous errors and recompile.
- Try preprocessing the file to examine the token stream, especially at end of buffer.

2.6.19 C/C++ Error 10118

preceding #if is missing

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.

Try preprocessing the file to examine the token stream, especially at end of buffer.

2.6.20 C/C++ Error 10119

unterminated #if / macro

An #if directive is found with no matching #endif directive, or a macro definition is not complete.

Fix

The solutions are:

- Terminate #if with #endif
- Complete the macro definition
- Try preprocessing the file to examine the token stream, especially at end of buffer.

2.6.21 C/C++ Error 10120

unexpected token

The compiler found an unexpected token.

Fix

Fix all previous errors and recompile.

2.6.22 C/C++ Error 10121

declaration syntax error

The compiler encountered an improper declaration, perhaps a misused keyword.

For example:

```
long char;
```

Fix

Examine the declaration in question. If the error is not apparent, it may be caused by a previous error. Correct all previous errors and recompile.

2.6.23 C/C++ Error 10122

identifier name redeclared

The variable was declared more than once.

```
void f(int a,int a);
```

Fix

Delete or rename (shown below) one of the duplicate variables.

```
void f(int a,int b);
```

2.6.24 C/C++ Error 10123

';' expected

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

```
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 and recompile.

2.6.25 C/C++ Error 10124

illegal constant expression

The compiler encountered 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, correct all previous errors and recompile.

2.6.26 C/C++ Error 10125

']' 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.

NOTE

Select the **Balance While Typing** checkbox in the **Edit > Preferences... > Editor Settings** preference panel. This preference enables you to balance brackets as you type them.

2.6.27 C/C++ Error 10126

illegal use of 'void'

An operator is incorrectly applied to a void type, or a variable is declared as a void type.

```
int myInt;

long myNumber;

void myFoo; //error: a variable cannot be declared as void
```

Fix

Use a legal variable type as shown: `int myFoo;`

2.6.28 C/C++ Error 10127

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.

2.6.29 C/C++ Error 10128

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.

2.6.30 C/C++ Error 10129

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). If the last member of a struct is an empty array, it is not supported by CodeWarrior C/C++; as demonstrated in the code example below.

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

Fix

As a work around for the code example shown above, the code should change to:

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

NOTE

Remember to change your allocation routines so that they allocate the right size for these structs.

For example, the proper allocation routine would be:

```
sizeof(DataBase) - (nb_elements -1) * sizeof(Data)
```

as shown in the code example below.

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

2.6.31 C/C++ Error 10130

'}' 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.

NOTE

Select the Balance While Typing checkbox in the **Edit > Preferences... > Editor Settings** preference panel. This preference enables you to balance brackets as you type them.

2.6.32 C/C++ Error 10131

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.

2.6.33 C/C++ Error 10132

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.

2.6.34 C/C++ Error 10133

struct/union/enum/class member *name* 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.

You can preprocess your source file and search the preprocessed file to find the previous use.

2.6.35 C/C++ Error 10134

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.

2.6.36 C/C++ Error 10135

'{' 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.

NOTE

Select the Balance While Typing checkbox in the **Edit > Preferences... > Editor Settings** preference panel. This preference enables you to balance brackets as you type them.

2.6.37 C/C++ Error 10136

illegal use of incomplete struct/union/class 'type'

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

```
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:

```
// foo.h
class Bar; // empty forward declaration
class aClass {
public:
// next line is invalid
void DoIt(int x) { mBar->DoStuff(x); }
...
private:
Bar* mBar; // you declare it
};
Class Bar { // actual definition of class
```

```
public:
void DoStuff(int x);
...
};
```

Fix

To avoid the errors, include bar.h to get the full class declaration. Sometimes this does not work because of circular references in the include files. Another option is to avoid inline 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.

2.6.38 C/C++ Error 10137

struct/union/class size exceeds 32k

This error appears when the size of a class, union, or struct is greater than 32k. A struct, class, or union (usually a declared array), is stored on the stack if it is a local variable 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.

2.6.39 C/C++ Error 10138

illegal bitfield type *name*

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

```
long err:33;
```

Fix

Check the bitfield declaration in question to ensure that a named bitfield is not declared with zero, a bitfield is not declared with a negative size, or that too many bits are not requested.

Example of wrong type:

```
    struct X {


           int* bf : 1;  //'int*' bitfields are not allowed

    };
=>
    Error   : illegal bitfield type 'int *'

    Test.cp line 2    int* bf : 1;
```

Workaround:

```
    struct X {

           int bf : 1;

    };
```

2.6.40 C/C++ Error 10139

division by 0

When an expression tries to divide by zero or use modulo zero, a division by 0 error is signaled. When the right hand side of a division or modulo expression evaluates to zero, the behavior is undefined in C and C++.

This may be just a warning, depending on the compiler and/or the "Treat All Warnings As Errors" setting of the "C/C++ Warnings" panel.

```
int i = 10 / 0;
```

Fix

If the division by zero or modulo zero is unintentional, then remove it from the expression. Use a non-zero value in divisions.

```
int i = 10 / 2;
```

2.6.41 C/C++ Error 10140

undefined identifier var

This error occurs when an identifier is used that has not been defined. Often this is encountered when using namespace in C++.

Fix

Check for scope or namespace errors. Check for spelling errors. This may be a result of a previous un-terminated variable list in an enum, struct or class declaration.

2.6.42 C/C++ Error 10141

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.

2.6.43 C/C++ Error 10142

not an lvalue

Another expression other than a legal variable expression was found (shown in **bold**).

```
void f(int a)
{
1 = a;
}
```

Fix

The compiler expects an expression that refers to an item to which it can assign a value, such as a variable.

```
void f(int a)
{
a = 1;
}
```

2.6.44 C/C++ Error 10143

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.

NOTE

This message is no longer used.

2.6.45 C/C++ Error 10144

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.

2.6.46 C/C++ Error 10145

datatype 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; as in "forward declarations".

```
union U u = { 1 };
```

Fix

Use complete data types.

```
union U { int m; } u = { 1 };
```

SeeAlso

"illegal use of incomplete struct/union/class"

2.6.47 C/C++ Error 10146

illegal type

The compiler generates this error message when an illegal type is encountered.

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

Fix

Use an operand with an appropriate type.

```
static void func(int *ip)
{
delete ip;
}
```

2.6.48 C/C++ Error 10147

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.

2.6.49 C/C++ Error 10148

pointer/array required

The compiler found a left bracket, [, following a variable which is neither a pointer nor an array.

```
void f(int a)
{
```

```
a[0] = 1;
}
```

Fix

The left bracket can only follow a pointer or array name. Use a pointer or array operand.

```
void f(int a[])
{
a[0] = 1;
}
```

2.6.50 C/C++ Error 10149

not a struct/union/class

The compiler expected to find a struct, union, or class, but found a simple type instead, shown in the code example below.

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

Fix

Use a struct, union, or class type.

```
struct X { int myfoo; } var;
var.myfoo = 10;
```

2.6.51 C/C++ Error 10150

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 the code example below, *theColors.color* references a member, var, that does not belong to the struct *ColorValues*.

```
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 the case, either change the item referenced or modify the structure, union, or class.

2.6.52 C/C++ Error 10151

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 the following example.

```
#include "AbstractHeader.h"

#include "Poo.h" // This file doesn't exist

#include <QDoffscreen.h>
```

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 Always Search User Paths checkbox in the Access Path settings panel.

SeeAlso

For more on access paths and the option Always Search User Paths, consult the C Compilers Reference

2.6.53 C/C++ Error 10152

illegal instruction for this processor

An assembly language instruction is found that does not exist for this family of microprocessors.

Fix

Use a supported assembly language instruction.

2.6.54 C/C++ Error 10153

illegal operands for this processor

The compiler found an assembly-language instruction that refers to operands that do no exist for this family of microprocessors.

Fix

Use supported assembly operands.

2.6.55 C/C++ Error 10154

number is out of range

A numeric value was encountered that is out of range for its data type.

```
int i = 10000000000000000000000000000000000000;
```

Fix

Use a value that is representable on the target architecture.

```
int i = 100;
```

SeeAlso

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

2.6.56 C/C++ Error 10155

illegal addressing mode

An assembly-language instruction attempts to use an addressing mode that is not possible with this instruction, as shown in the code example below.

```
moveq d0,d1 //68K
```

Fix

Use an addressing mode that is supported for this instruction.

```
moveq #0,d1 // 68K
```

2.6.57 C/C++ Error 10156

illegal data size

A line in an assembly function contains an illegal data size.

```
move.Z #0,d9 //68K
```

Fix

Use a size that is supported for this instruction.

```
move.w #0,d1 // 68K
```

2.6.58 C/C++ Error 10157

illegal register list

The compiler encountered an illegal register list in an assembly function.

```
movem.l d0-d0, -(sp) //68K
```

Fix

Use a legal register list.

```
movem.l d0-d2,-(sp)     // 68K
```

2.6.59 C/C++ Error 10158

branch out of range

A branch destination in an assembly function is out of range.

```
bra.s 10000 //68K
```

Fix

Use a smaller displacement or a branch instruction that supports a wider range.

2.6.60 C/C++ Error 10159

undefined label Lbl

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

Fix

The solutions are:

- Remove the goto.
- Create the necessary label within the scope where the error occurs.

2.6.61 C/C++ Error 10160

reference to label lbl is out of range

An assembly function contains a branch whose destination is out of range.

```
bra.s label // error: label too far away
... // a lot of code
label:
```

Fix

The solutions are:

- Move label closer
- Use a branch that supports a wider range.

```
bra.s label
... // less code
label:
```

2.6.62 C/C++ Error 10161

call of non-function

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

2.6.63 C/C++ Error 10162

function call does not match prototype

A function call's arguments do not correspond to the function's prototype. The code example below shows that the call to SetFoo() passes two arguments when the function prototype requires only one.

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

Fix

Match function call with function prototype. Select the function call and choose Search > Find Definition to locate the function prototype definition.

2.6.64 C/C++ Error 10163

illegal use of register variable

A variable has been incorrectly specified as a register variable. For example, in the code below, an attempt is made to take the address of a register variable.

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

Fix

Do not use the register specifier.

```
int i;
f(&i);
```

2.6.65 C/C++ Error 10164

illegal type cast

The code attempted to typecast data to an incompatible data type.

```
extern const int i;
int *ip = const_cast<int *>(i);
```

Fix

Use an operand/type that is legal.

```
extern const int i;
int *ip = const_cast<int *>(&i);
```

2.6.66 C/C++ Error 10165

function already has a stack frame

The compiler found more than one fralloc directive in an assembly function.

Fix

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

2.6.67 C/C++ Error 10166

function has no initialized stack frame

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

Fix

Modify your assembly function so that it uses fralloc.

2.6.68 C/C++ Error 10167

value is not stored in register

This error occurs in a GCC-style __asm__("reg") declaration when the variable does not have a register storage class.

Fix

Use a register storage class.

2.6.69 C/C++ Error 10168

function nesting too complex

The compiler encountered a function that contains too many nested ({ }) blocks.

Fix

Divide the function into a series of dependent functions.

2.6.70 C/C++ Error 10169

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, the illegal use of keyword error is caused by a missing colon (shown below).

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

Fix

Fix all previous error messages. Verify that you are using the keyword correctly, you are using correct syntax, and you are using recognizable parameters.

2.6.71 C/C++ Error 10170

':' expected

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

```
switch(theMenu) {
 case APP_MENU_ID // error: missing ':'
}
switch(theItem) {
 case ABOUT_ITEM : // Correct
}
```

Fix

If the error is not apparent, an error in a previous statement may be causing the problem. Correct all previous errors and recompile.

2.6.72 C/C++ Error 10171

label Lbl redefined

An attempt was made to redefine a label (in this case, Lbl) that has already been defined.

Fix

Remove or rename one of the labels.

2.6.73 C/C++ Error 10172

case constant defined more than once

A constant used in a switch statement was defined more than once.

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

Fix

Remove one of the constant labels.

2.6.74 C/C++ Error 10173

default label defined more than once

The compiler found more than one default label in the same switch statement.

More than one default:

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

Fix

Remove one of the default labels.

2.6.75 C/C++ Error 10174

illegal initialization

A variable, or other data type, is illegally initialized within a function.

```
int arr[2] = 2;
```

Fix

Use a legal initializer.

```
int arr[2] = { 1, 2 };
```

2.6.76 C/C++ Error 10175

illegal use of inline function

An inline function was used illegally. For example in the code snippet below, an attempt is made to take the address of an inline function.

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

Fix

Use a non-inline function.

```
Handle NewHandle(Size byteCount) { ... }
...
&NewHandle;
```

2.6.77 C/C++ Error 10176

illegal type qualifier(s)

An illegal type qualifier, for this type in this scope, was encountered. For example, in the code snippet below, the double const qualifier produces an illegal type qualifier error.

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

Fix

Remove the illegal type qualifier. This error may be caused by a previous error. Correct all previous errors and recompile.

2.6.78 C/C++ Error 10177

illegal storage class

The compiler issues this error when an illegal storage class is used. This error is also generated when variables in a global scope are declared as auto (shown below).

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

Fix

Use a legal storage class.

```
static int x;
```

2.6.79 C/C++ Error 10178

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.

NOTE

Refer to the C Compilers Reference manual.

2.6.80 C/C++ Error 10179

illegal assignment to constant

An expression attempted to assign a value to a 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 variable.

2.6.81 C/C++ Error 10180

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 shown in Example 1.

Example 1 - Illegal usage of precompiled header : too many

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

This error also 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 Example 2.

Example 2 - Illegal usage of precompiled header : declaration

```
long l;
#include <Headers>
// error: precompiled header included after declaration
```

Fix

Check all the precompiled header files included with your project. Also check the header specified in the C/C++ Language Settings panel Prefix File field.

This error can also occur when building precompiled headers with the *.pch or *.pch++ file. Precompiled headers may not contain data initializations (except for const data, when "#pragma cpp_extensions on" is active) or function definitions (except for static or inline functions).

Example - Illegal usage of precompiled header : data initializations

```
int utility_func(char *x) {return strlen(x); }

// cannot define function bodies

int flag = 1;

// cannot initialize data
```

2.6.82 C/C++ Error 10181

illegal data in precompiled header

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

Fix

Remove the data from your precompiled header and precompile it again. Refer to the C Compilers reference manual for information on illegal items in a precompiled header.

2.6.83 C/C++ Error 10182

variable/argument name is not used in function

A variable declared in a function is not used in the function body. This warning is signaled as an error if the Unused Variables checkbox is selected in the C/C++ Warnings settings panel.

Fix

The solutions are:

- Remove the unused variable
- Deselect the Unused Variables checkbox

NOTE

Refer to the C Compilers Reference for more on the Unused Variables checkbox.

2.6.84 C/C++ Error 10183

illegal use of direct parameters

This error is issued when a function, or another expression, references a direct parameter that is not supported. The example below attempts to illegally use __X.

```
void f(int arg:__X);
```

Fix

A direct parameter must be either __D0 to __D2, __A0, __A1, or __FP0 to __FP3.

```
void f(int arg:__D0);  //68K
```

2.6.85 C/C++ Error 10184

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 Example 1 should return an int or be declared as void.

Example 1 - Return Value Expected

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

In C++, a function declared without a return value is implied to return an int type as in Example 2 below.

Example 2 - Return Value From main() Function Expected

```
int main()
{
 cout << "working";
 //<-- warning generated here
}
```

Fix

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

2.6.86 C/C++ Error 10185

variable var is not initialized before being used

The compiler encountered an expression using a variable that has neither been assigned a value nor initialized.

C/C++ does not define an initial value for local variables with auto storage class, upon entry to a function.

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

Fix

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

2.6.87 C/C++ Error 10186

illegal #pragma

The compiler found an unrecognized #pragma directive.

Fix

Supported #pragmas are listed in the C Compilers Reference. Consult this list to make sure the #pragma you are using exists and is spelled correctly.

2.6.88 C/C++ Error 10187

illegal access to protected/private member

A function attempted to access a private member.

The example below shows priv is declared as _private. Function func() attempts to access this member.

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

Fix

Make the protected/private member public.

```
class aClass { public  : int priv; };
void func() {
 aClass x;
 x.priv=0; // OK
}
```

2.6.89 C/C++ Error 10188

ambiguous access to class/struct/union member

A reference to a class, struct, or union member is ambiguous. This message is generated when calling a function that is defined in both a virtual base class and another base class with the same parameters.

```
struct A { int m; };
struct B { int m; };
struct C : A, B {};
int main()
{
 C  c;
 c.m = 1; // error: A::m or B::m?
}
```

Fix

You must use qualified member to define which function you want to use.

```
int main()
{
 C c;
 c.A::m = 1;
 c.B::m = 2;
}
```

2.6.90 C/C++ Error 10189

illegal use of 'this'

The compiler encountered C++ code that uses this in a non-member function.

```
void f()
{
 this->m = 0; // error: f() has no 'this' pointer
}
```

Fix

Use `this` only in non-static member functions.

```
void A::f()
{
 this->m = 0;
}
```

2.6.91 C/C++ Error 10190

unimplemented C++ feature

The compiler encountered a C++ feature that is not supported by Freescale C++.

```
export template<typename T> void f(T);
// export is currently not supported
```

2.6.92 C/C++ Error 10191

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 (MacOS only).

```
struct X : HandleObject {
  int m;
};
X  x;  // error HandleObjects can only be allocated
```

Fix

Use HandleObject appropriately.

```
struct X : HandleObject {
 int m;
};
X *xp = new X;
```

2.6.93 C/C++ Error 10192

illegal access qualifier

This error message is no longer used.

This error is signaled when an illegal qualification is encountered. The example below shows code that is illegal because foo doesn't exist.

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

2.6.94 C/C++ Error 10193

illegal 'operator' declaration

The compiler found an illegal operator declaration.

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

Fix

Use legal parameters for operator functions.

```
struct X;

int operator +(X&,int);
```

2.6.95 C/C++ Error 10194

illegal use of abstract class *classname*

The compiler detected an attempt to instantiate from an abstract class.

An abstract class is defined with at least one pure virtual method. An abstract class requires you to make a subclass that provides methods to replace any pure virtual methods.

A pure virtual method is declared as shown below.

```
virtual type MethodName ( arguments ) = 0;
```

Fix

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

2.6.96 C/C++ Error 10195

illegal use of pure function

The compiler generated this message when it encountered improper use of a pure virtual function.

A pure virtual function has no definition. The example below shows the constructor attempting to call the myFun() pure function.

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

Fix

Use a non-pure function.

```
class pure {
 virtual int myFun();
 pure() { myFun(); }
};
```

2.6.97 C/C++ Error 10196

illegal '&' reference

The compiler encountered an illegal & reference.

```
int &&g; // error: illegal reference to reference type
```

Fix

Use a legal reference type.

```
int &ref;
```

2.6.98 C/C++ Error 10197

illegal function overloading

Functions with the same name and identical arguments, but with different return types, were declared.

```
int f(int);
long f(int); // error
```

Fix

Use distinct parameter lists for overloaded functions.

```
int f(int);
long f(long);
```

2.6.99 C/C++ Error 10198

illegal operator overloading

This error message is no longer used.

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

2.6.100 C/C++ Error 10199

ambiguous access to overloaded function

An ambiguous reference was made to an overloaded function. References to overloaded functions must be unambiguous.

```
int f(char);
int f(long);
int i = f(1);  // error: f(char) or f(long)?
```

Fix

Use type casts to make a function call specific instead of ambiguous.

```
int f(char);
int f(long);
int i = f((char)1);
```

2.6.101 C/C++ Error 10200

illegal access/using declaration

Access to a base class member, from a derived class, was incorrectly declared.

```
int g;
 struct X {
  using ::g;
// error: cannot specify access to global a variables
};
```

Fix

Use the base class member's qualified-name in a public or protected part of a derived class declaration.

```
struct Base {
 int g;
};
struct X : Base {
 using Base::g;
};
```

2.6.102 C/C++ Error 10201

illegal 'friend' declaration

The compiler encountered an illegal declaration.

```
class X;
int i;
class aClass {
 friend X; // error: elaborate type specifier is required
 friend int i; // error: variables cannot be friends
};
```

Fix

Only use elaborate type specifiers for classes or functions in friend declarations.

```
class X;
int f();
class aClass {
 friend class X;
 friend int f();
};
```

2.6.103 C/C++ Error 10202

illegal 'inline' function definition

This error message is no longer used.

A function that has already been referenced is defined as inline.

2.6.104 C/C++ Error 10203

class has no default constructor

The compiler cannot construct a class because it has no default constructor.

```
struct A {
  A(int);
};
A *ap = new A[10]; // error: no default constructor
```

Fix

Provide a default constructor.

```
struct A {
  A();
  A(int);
};
A *ap = new A[10];
```

2.6.105 C/C++ Error 10204

illegal operator

The compiler encountered an illegal operator.

```
struct X;
int operator .(X,int);
```

Fix

Use a legal operator.

```
struct X;
int operator +(X,int);
```

2.6.106 C/C++ Error 10205

illegal default argument(s)

The compiler found a function that contains one or more illegal arguments.

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

// error: 'z' has no default argument
```

Fix

Provide correct default arguments. All arguments after the initial default argument must also have defaults.

```
int func(int x=1, int z=2);
```

2.6.107 C/C++ Error 10206

possible unwanted ';'

(Warning Message) A semicolon was found immediately following a while, if, or for statement. This can cause an unintended logical error.

This warning is signaled when the Possible Errors option is selected in the C/C++ Warnings settings panel.

Fix

The solutions are:

- Either remove the unwanted semicolon.
- Deselect the Possible Errors option in the C/C++ Warnings settings panel.

NOTE

For more on the Possible Errors option, consult the C, C++, and Assembler Language Reference manual.

2.6.108 C/C++ Error 10207

possible unwanted assignment

(Warning Message) An assignment (= operator) occurred within a logical expression in a while, if, or for statement. This may be meant as an equality operation.

This is signaled as an error when the Possible Errors checkbox is selected in the C/C++ Warnings settings panel.

Fix

The solutions are:

- Correct the unwanted assignment.
- For intentional assignments, enclose assignment in parenthesis.

For example:

```
while (c = check()) { ... } // generates warning

while ((c = check())) { ... } // no warning
```

- Deselect the Possible Errors checkbox under in the C/C++ Warnings settings panel.

NOTE

For more on the Possible Errors checkbox, consult the C, C ++, and Assembler Language Reference manual.

2.6.109 C/C++ Error 10208

possible unwanted compare

(Warning Message) This warning occurs when the compiler believes it finds an unwanted comparison (== operator).

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

Fix

The solutions are:

- Correct the unwanted comparison.
- Deselect the Possible Errors checkbox under in the C/C++ Warnings settings panel.

NOTE

For more on the Possible Errors checkbox, refer to the C, C ++, and Assembler Language Reference manual.

2.6.110 C/C++ Error 10209

illegal implicit conversion from Type_A to Type_B

The compiler encountered an illegal implicit conversion.

The ANSI C++ language 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)

In C++, a **void*** cannot be assigned to an object of any type other than **void*** without an explicit cast.

The example below is legal ANSI C, but is not accepted in C++.

```
void f(char *cptr, void *vptr)
{
 cptr = vptr; // Illegal in C++, legal in C
 char *ptr=(void *)0; // Illegal in C, legal in C++
}
```

Fix

Use explicit type casts.

```
void f(char *cptr, void *vptr)
{
 cptr = (char *)vptr;
 char *ptr=(char *)0;
}
```

NOTE

Refer to the ANSI C Standard 3.2.2.3 for detailed information on pointer conversions.

2.6.111 C/C++ Error 10210

local data >32k

Local data totals have exceeded 32K.

Fix

The local data, a declared array, is stored on the stack and has a limit of 32K. The solutions are:

- Define an array as static
- Use dynamic allocation to move the storage from the stack to the heap

2.6.112 C/C++ Error 10211

illegal jump past initializer

A transfer was made into a block that bypasses initializers. In certain cases, it is illegal to jump past explicit or implicit initializers. This error usually occurs whenever there is a section of code that can be jumped past in the same scope.

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

Fix

Move the initializations before the branch so that they cannot be skipped.

```
int v1 = 2;
short v2 = 3;
switch (i) {
 case 1:
```

```
 case 2:
 if( v2 == 7 ) {}
}
```

2.6.113 C/C++ Error 10212

illegal ctor initializer

This error is signaled when the compiler encounters an illegal ctor initializer. For example, the ctor initializer in the example below is illegal because there is no aClass i member.

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

Fix

Only use ctor-initializers for direct base classes or members.

2.6.114 C/C++ Error 10213

cannot construct *type's* base class *type*

The base class aClass has no ctor initializer or default constructor.

```
struct A { A(int); };
struct B : A {
 B() {}  // error: cannot construct base class A
};
```

Fix

Provide the ctor initializer in a derived constructor, or in a default constructor in the base class.

```
struct A { A(int); };
struct B : A {
```

```
 B() : A(1) {}
};
```

2.6.115 C/C++ Error 10214

cannot construct *type's* direct member *name*

The direct member, aClass, has no constructor initializer, or default constructor.

```
struct A { A(int); };
struct B {
 A m;
B() {} // error: cannot construct member 'm'
};
```

Fix

Provide a ctor initializer or a default constructor in the members class.

```
struct A { A(int); };
struct B {
 A m;
B() : m(1) {}
};
```

2.6.116 C/C++ Error 10215

#if nesting overflow

The number of nested #if processor directives exceeded the maximum number allowed.

Fix

Divide the large nested #if into a series of smaller nests.

2.6.117 C/C++ Error 10216

illegal empty declaration

A declaration is missing an identifier.

```
int ;
```

Fix

Specify a declaration.

```
int i;
```

2.6.118 C/C++ Error 10217

illegal implicit enum conversion from Type_A to Type_B

The compiler has encountered an illegal implicit conversion involving an enum.

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

NOTE

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.

Fix

Use an explicit type conversion.

```
enum ff { foo };
enum ff x = (enum ff)0;
```

2.6.119 C/C++ Error 10218

illegal use of #pragma parameter

This message is displayed when a previous #pragma parameter does not match a function. The example below shows that func() does not match the function func1.

```
#pragma parameter __A0 func
char *func1() // error: wrong name
{
```

```
 // ...
}
```

NOTE

If the Illegal Pragmas option is selected in the C/C++ Warnings settings panel, undefined #pragmas are marked as warnings.

Fix

The #pragma parameter name must match the function name.

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

2.6.120 C/C++ Error 10219

virtual functions cannot be pascal functions

A virtual function was declared as a pascal function.

```
class aClass {
 virtual pascal int func();
};
```

Fix

Do not use virtual pascal functions.

```
class aClass {
 virtual int func();
};
```

2.6.121 C/C++ Error 10220

illegal implicit const/volatile pointer conversion from typeA to typeB

(Warning Message) A const pointer was converted into a variable.

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

Fix

Use an explicit type conversion.

```
void func(const char *cptr)
{
 char *ptr=(char *)cptr;
}
```

2.6.122 C/C++ Error 10221

illegal use of non-static member

An attempt was made to access a non-static member without having an object of that class.

```
struct A {
 int m;
};
int main()
{
 return A::m;
}
```

Fix

Use a class instance to access non-static members.

```
struct A {
 int m;
};
int main(A *ap)
{
```

```
 return ap->m;
}
```

2.6.123 C/C++ Error 10222

illegal precompiled header version

The compiler found a precompiled header file that was old or defective.

Fix

If you are using a prefix file provided by Freescale, rebuild the precompiled header projects (these include MSLHeaders.mcp).

2.6.124 C/C++ Error 10223

illegal precompiled header compiler flags or target

The compiler encountered a precompiled header file that uses the wrong compiler target.

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.

2.6.125 C/C++ Error 10224

'const' or '&' variable needs initializer

The const or reference variables were not initialized.

```
const int a; // error
const int &b; // error
```

Fix

You must initialize the const or reference variables when you declare them.

```
const int a = 1;
const int &b = a;
```

2.6.126 C/C++ Error 10225

function hides inherited virtual function func2

A non-virtual member function that hides a virtual function in a superclass was declared.

One function hides another if it has the same name but 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)
};
```

NOTE

This appears as a warning only if the Hidden virtual functions option in the C/C++ Warnings settings panel was enabled.

Fix

Turn off the Hidden virtual functions option or choose another name for one of the functions. Also, ensure that all derived virtual functions have identical parameter lists as the base virtual function.

2.6.127 C/C++ Error 10226

pascal function cannot be overloaded

A pascal function was overloaded.

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

Fix

Do not use overloaded pascal functions.

```
int f(int);
void f();
```

2.6.128 C/C++ Error 10227

object differs from virtual base function *object* 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.

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

Fix

Use matching (or covariant) return types for overriding functions.

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

2.6.129 C/C++ Error 10228

non-const '&' reference initialized to temporary

The compiler found that the initializer for a non-const reference is not an appropriate lvalue.

```
long &r = 40000;
```

Fix

Use a real variable to initialize reference or use a const reference.

```
long x = 4000;

long &y = x;

const long &z = 40000;
```

2.6.130 C/C++ Error 10229

illegal template declaration

The compiler encountered a malformed template declaration.

```
template <class T> T i;

// error not a class/function/member
```

Fix

You must use a legal class, function, or member when declaring a template.

```
template <class T> T f();
```

2.6.131 C/C++ Error 10230

'<' expected

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

Fix

Check the syntax and symbols you may have used in a previous statement.

NOTE

The Balance command does not check for angle brackets.

2.6.132 C/C++ Error 10231

'>' expected

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

Fix

Check the syntax and symbols you may have used in a previous statement.

NOTE

The Balance command does not check for angle brackets.

2.6.133 C/C++ Error 10232

illegal template argument(s)

The compiler found a template that contains one or more illegal arguments.

```
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
```

Fix

Check to ensure you have correct spacing in all nested templates.

2.6.134 C/C++ Error 10233

cannot instantiate obj

The template _obj cannot be instantiated because it is undefined.

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

Fix

Define the class template before it is instantiated.

```
template <class T> class aClass { T m; };
template class aClass<int>;
```

2.6.135 C/C++ Error 10234

template redefined

An attempt was made to redefine a template that has already been defined.

Fix

Remove or rename one of the template definitions.

2.6.136 C/C++ Error 10235

template parameter mismatch

The member function template parameter list does not match the class parameter list.

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

Fix

Use matching parameter lists.

```
template <class T> class aClass { void f() };
template <class T> void aClass<T>::f() { ... };
```

2.6.137 C/C++ Error 10236

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

An attempt was made to pass a data object declared as a const to a member function that is not declared as const.

```
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
```

Fix

Use matching const/volatile member functions for const/volatile class objects.

2.6.138 C/C++ Error 10237

preceding '#pragma push' is missing

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

Fix

Add a matching "#pragma push' for each '#pragma pop'

```
#pragma push

#pragma pop
```

NOTE

For more information on all available pragmas, consult the C/C ++ Reference and Assembler Language Reference.

2.6.139 C/C++ Error 10238

illegal explicit template instantiation

The compiler encountered an illegal explicit template instantiation. The example below shows how the f() function was not correctly declared as a template function.

```
//template <class T>

void f();

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

Fix

Change the explicit template instantiation to match the defined template.

```
template <typename T> class X { T m; };

template class X<int>;
```

2.6.140 C/C++ Error 10239

illegal X::X(X) copy constructor

A class constructor function was declared with an argument of the same class type.

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

Fix

Use a reference argument.

```
class aClass {
aClass(aClass&);
};
```

2.6.141 C/C++ Error 10240

function defined 'inline' after being called

A function was declared inline after it had already been called.

```
int func( int x );
class cA {
 int i;
 public:
 cA() { i = func( 3); }
};
inline int func( int x ) { return x + 1; }
```

Fix

Declare the function prototype to be inline.

2.6.142 C/C++ Error 10241

illegal constructor/destructor declaration

An attempt was made to declare a constructor or destructor in an illegal manner.

```
struct A {
 void A(); //error: return type in ctor decl
};
```

Fix

Do not specify a return type.

```
struct A {
 A();
};
```

2.6.143 C/C++ Error 10242

'catch' expected

The try-block is not directly followed by a catch-clause.

```
int main()
{
  try {} // error: no catch
}
```

Fix

Add at least one catch-clause after a try-block.

```
int main()
{
 try {}
 catch(...) {}
}
```

2.6.144 C/C++ Error 10243

#include nesting overflow

The number of nested #includes processor directives exceeds the maximum number allowed.

Fix

Divide the large nested #includes into a series of smaller nests.

2.6.145 C/C++ Error 10244

cannot convert typeA to typeB

A type conversion was attempted without proper conversion constructors, or with incompatible types. The code in the example below attempts to convert a type long * to an int *.

```
int *ptr = new long; // <-- Error wrong type
```

Fix

Use matching types or a type cast.

```
int *ptr = new int;
```

2.6.146 C/C++ Error 10245

type mismatch typeA and typeB

The compiler found a different data type than the one it had expected.

```
int *p1;
char *p2;
int diff = p1 - p2;  // error
```

This error may also occur if one of your functions has the same name as a Freescale macro.

Fix

Use matching data types or typecasts:

```
char *p1;
char *p2;
int diff = p1 - p2;  // OK
```

2.6.147 C/C++ Error 10246

class type expected

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

```
__uuidof(int); // error: int is not a class (Windows compiler only)
```

Fix

Use a class type.

```
class X { ... };
__uuidof(X);
```

2.6.148 C/C++ Error 10247

illegal explicit conversion from type_A to type_B

The compiler encountered an explicit conversion of one type to an improper type.

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

Fix

Change to a legal conversion or provide a conversion function.

```
class D { operator long(); };
D d;
long x = (long)d;
```

2.6.149 C/C++ Error 10248

function call *func* does not match *func*

A call to a function did not match the expected arguments.

```
extern int f(int);

int i = f();  // error
```

Fix

Use matching function arguments.

```
extern int f(int);

int i = f(1);  // OK
```

NOTE

An attempt to initialize an object without a proper matching constructor also generates this message. 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.

2.6.150 C/C++ Error 10249

identifier *name* redeclared was declared as: typeA now declared as: typeB

The source code attempted to redefine an identifier. This error is often the result of reusing the name of a Macintosh declared variable or function.

```
extern long var;

int var; // error
```

Fix

Use matching types.

```
extern long var;

long var;
```

2.6.151 C/C++ Error 10250

cannot throw class with ambiguous base class (*name*)

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.

2.6.152 C/C++ Error 10251

class *type*: object has more than one final overrider: *object* and object

A virtual function was found that has no final overrider.

```
struct A {
 virtual void f();
};
struct VB1 : virtual A {
 void f();
};
struct VB2 : virtual A {
 void f();
};

struct B : VB1, VB2
 {
// error A::f() has no final overrider
};
```

NOTE

Refer to ISO C++ 10.3.

Fix

Add a final overriding function.

```
struct B : VB1, VB2 {
void f();
};
```

2.6.153 C/C++ Error 10252

exception handling option is disabled

This error occurs when the Enable C++ Exceptions option in the C/C++ Language settings panel is disabled, and you try to use EH (for example, **throw**).

Fix

Enable the exception handling option in the C/C++ Language settings panel.

2.6.154 C/C++ Error 10253

cannot delete pointer to const

The compiler encountered an attempt to delete a pointer to a const value.

```
main()
{
 const int y = 3;
 int const *ptr = &y;
 delete ptr; // <-- Error
 return 0;
}
```

NOTE

This error message is no longer used.

2.6.155 C/C++ Error 10254

cannot destroy const object

The compiler detected an attempt to destroy a const object.

NOTE

This error message is no longer used.

2.6.156 C/C++ Error 10255

const member *a Var* is not initialized

The compiler encountered a const member that was not initialized correctly.

Fix

Initialize the const member at the time of the object's construction.

2.6.157 C/C++ Error 10256

'&' reference member *var* is not initialized

A reference member was not initialized.

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

Fix

All reference types must be evaluated in the scope of the constructor.

```
class caClass {
 private:
 int x;
 public:
 const int &ref;
 caClass():ref(x) {} //<-- reference is initialized
};
```

2.6.158 C/C++ Error 10257

RTTI option is disabled

A run-time type identification was attempted when the RTTI pragma, or the Enable RTTI option in the C/C++ Language settings panel, was disabled.

Fix

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

2.6.159 C/C++ Error 10258

constness casted away

An attempt to cast away **const** and/or **volatile** was encountered.

```
const int *ip;
char *vp = reinterpret_cast<char *>(ip);
```

Fix

Use const_cast to cast away const and/or volatile.

```
const int *ip;
char *vp = reinterpret_cast<char *>(const_cast<int *>(ip));
```

2.6.160 C/C++ Error 10259

illegal const/volatile '&' reference initialization

The compiler encountered either a const or a volatile reference that was improperly initialized.

Fix

Initialize the reference during object construction.

2.6.161 C/C++ Error 10260

inconsistent linkage: 'extern' object redeclared as 'static'

The compiler encountered an extern object that was redeclared or defined as static.

```
extern   int f();
static int f();
```

Fix

Use consistent storage class specifiers.

```
static    int f();
static int f()
```

2.6.162 C/C++ Error 10261

unknown assembler instruction mnemonic

An illegal instruction name was reported.

Fix

Correct the mnemonic in the assembler instruction.

2.6.163 C/C++ Error 10263

variable could not be assigned to a register

A variable could not be assigned to a register.

2.6.164 C/C++ Error 10264

illegal exception specification

You used an exception specification where it is not allowed.

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

Fix

Remove the exception specification.

```
typedef void (*f)();
```

2.6.165 C/C++ Error 10265

exception specification list mismatch

The exception specification lists for a function declaration and a function definition do not match.

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

Fix

Use a matching exception specification.

```
void f() throw(int);
void f() throw(int) {}
```

2.6.166 C/C++ Error 10313

illegal use of specifier

A keyword was used in an illegal context. This error is mostly used for qualifiers and specifiers.

```
inline int k; // illegal use of 'inline'
const int const cc; // illegal use of 'const'
```

NOTE

This error is mostly used for qualifiers and specifiers.

Fix

Remove incorrect specifier. The correct use is shown below.

```
int k;
const int cc;
```

2.6.167 C/C++ Error 10314

template too complex or recursive

The compiler gives this error when there are too many recursive template expansions.

Fix

Decrease the complexity of your algorithm.

2.6.168 C/C++ Error 10315

illegal return value in void/constructor/destructor function

An attempt was made to return a value from a void function, or a constructor or destructor, neither of which may return a value.

Fix

Remove the illegal return.

2.6.169 C/C++ Error 10316 (Warning Message)

assigning a non-int numeric value to an unprototyped function

This warning is activated by `#pragma warn_largeargs on`, or by passing `-warn largeargs` to the command-line compilers.

The compiler will emit a warning when passing a non-integer numeric value such as a float or long to an unprototyped function when the "require prototypes" option is off.

```
#pragma warn_largeargs on
void main()
{
  f(1.0);
}
```

Fix

Provide the prototype.

```
#pragma warn_largeargs on

void f(double);

void main()

{

  f(1.0);

}
```

2.6.170 C/C++ Error 10317

implicit arithmetic conversion from typeA to typeB

A potentially dangerous implicit conversion (for example - a conversion from a wide type to a narrow type) was attempted with the "Implicit arithmetic conversion" option selected in the Target Settings C/C++ Warnings panel.

```
#pragma warn_implicitconv on

char c = 1234567; // warning: 1234567 does not fit into a 'char'
```

Fix

Use a type cast to make the conversion explicit.

```
#pragma warn_implicitconv on

char c = (char)1234567;
```

2.6.171 C/C++ Error 10318

preprocessor #error directive

The compiler encountered the #error directive. The #error directive prevents you from compiling a section of code in certain situations.

Fix

Remove the #error directive.

NOTE

Before fixing this error, check to understand why this error directive was added.

2.6.172 C/C++ Error 10319

ambiguous access to name found '*symbol1*' and '*symbol2*'

The compiler detected an ambiguous access to a name. This occurs most often when the same name is used in multiple, legally accessible namespaces.

```
namespace A {
 int a;
}
long a;
namespace B {
 using namespace ::A;
 int x = a; //error: ::a or A::a?
}
```

Fix

Use explicit name qualification, for example '::a' or 'A::a'

```
namespace A {
 int a;
}
long a;
namespace B {
 using namespace ::A;
 int x = ::a;
 int y = A::a;
}
```

2.6.173 C/C++ Error 10320

illegal namespace

A namespace name that is not unique in the enclosing namespace was used as a namespace name.

```
int a;
namespace a { // ERROR
 int b;
}
```

Fix

Use unique namespace names.

```
int a;
namespace A {
 int b;
}
```

2.6.174 C/C++ Error 10321

illegal use of namespace name

A namespace name was used as a non-namespace.

```
namespace a {
int b;
}
int g = a++; // ERROR
```

Fix

Only use namespace names to qualify namespace members.

```
namespace a {
int b;
}
int g = a::b++;
```

2.6.175 C/C++ Error 10322

illegal name overloading

An attempt was made to perform an illegal overload.

```
int b;
enum { b }; // ERROR
```

Fix

Do not overload non-function names.

```
int b;
enum { e };
```

2.6.176 C/C++ Error 10323

instance variable list does not match @interface

The compiler generates an error when the instance variable list declared in the interface does not match what is defined in the implementation.

```
@interface A
{
 int a;
}
@end
@implementation A
{
 long b; // ERROR: inconsistent with declaration
}
@end
```

Fix

Use matching instance variable lists in interface and implementation.

```
@implementation A
{
 int a;
}
@end
```

2.6.177 C/C++ Error 10324

protocol list does not match @interface

The protocol list declared in the interface does not match what is defined in the implementation.

```
@protocol a

@end

@protocol b

@end

@interface A <a>

@end

@implementation A <b>  // ERROR: inconsistent with declaration

@end
```

Fix

Use matching protocol lists in interface and implementation.

```
@implementation A <a>

@end
```

2.6.178 C/C++ Error 10325

super class does not match @interface

The super class declared in the interface does not match what is defined in the implementation.

```
@interface a

@end

@interface b

@end

@interface c : a

@end

@implementation c : b  // ERROR: inconsistent with declaration

@end
```

Fix

Use matching super classes in interface and implementation.

```
@implementation c : a
@end
```

2.6.179 C/C++ Error 10326

function result is a pointer/reference to an automatic variable

A pointer/reference to an automatic variable is used as the result of a function.

```
char *foo()
{
 char c=0;
 return c;  // error
}
```

Fix

Return a pointer to a static or an allocated object.

```
char *foo()
{
static char c=0;
 return c;
}
```

2.6.180 C/C++ Error 10327

cannot allocate initialized objects in the scratchpad

(PlayStation only) Static initialization in the scratch pad is not supported.

```
__declspec(scratchpad) int P=1234; // ERROR
```

Fix

Call a routine to initialize it.

```
__declspec(scratchpad) int P; // OK  '
...
```

```
void InitScratchPad()
{
 P = 1234;
}
```

2.6.181 C/C++ Error 10328

illegal class member access

An attempt to access a class member that doesn't exist has occurred.

This error message is no longer used.

2.6.182 C/C++ Error 10329

data object *object* redefined

The compiler generates an error when a data object is incorrectly redefined.

```
int a = 1;
int a = 2; //  ERROR: the variable name is reused incorrectly
```

Fix

Remove the redundant definition.

2.6.183 C/C++ Error 10330

illegal access to local variable from other function

A function attempted an illegal access to a local variable.

```
void bar()
{
 int a=0;
 struct nest {
   void foo()
   {
```

```
      a=2; // ERROR: 'a' is bar's local variable,
          // which cannot be accessed from nest::foo
    }
  };
}
```

Fix

Use global variables.

2.6.184 C/C++ Error 10331

illegal implicit member pointer conversion

A member function is incorrectly initialized.

```
struct X {
 void foo();
};
void (X::*f)() = X::foo; // error: not '& qualified-id'
```

Fix

Use the & qualified-id to get pointer-to-member.

```
void (X::*f)() = &X::foo;
```

2.6.185 C/C++ Error 10332

type name redefined

A typename is incorrectly redefined.

```
struct B;
typedef int B; // Error
```

Fix

Use distinct typenames.

```
struct B;
typedef int BT;
```

2.6.186 C/C++ Error 10333

object *objectname* redefined

An object is incorrectly redefined.

```
void foo() {}
void foo() {} // ERROR
```

Fix

Remove the redundant definition.

```
void foo() {}
```

2.6.187 C/C++ Error 10334

'main' not defined as external 'int main()' function

If strict ANSI is selected, this error will be generated for anything other than the standard use of main.

That is

```
int main();
```

or

```
int main(void);
```

or

```
int main(int argc, char *argv[ ]);
```

is all that is allowed by the current international standards for C and C++ languages.

Fix

Use one of the above main prototypes or disable strict ANSI option.

2.6.188 C/C++ Error 10335

illegal explicit template specialization

There is something wrong with your explicit template specialization.

```
template <typename> T f(T);
template <> int f(long); // error: does not match 'T f(T)'
```

Fix

Use a specialization that matches the previously defined templates.

```
template <typename> T f(T);
template <> long f(long);
```

2.6.189 C/C++ Error 10336

name has not been declared in namespace/class

A namespace/class member was defined that was not declared in it's own namespace.

```
namespace N {}
int N::a; // error
```

Fix

Always declare namespace/class members before they are defined elsewhere.

```
namespace N {
 extern int a;
}
int N::a;
```

2.6.190 C/C++ Error 10337

preprocessor #warning directive

The compiler encountered the #warning directive. The #warning directive informs you that unexpected things could happen in a section of code.

NOTE

Before fixing this error, you should check to see why this warning directive was added. This directive is not available

when the ANSI Strict option is enabled in the C/C++ Language settings panel.

Fix

Remove the #warning directive.

2.6.191 C/C++ Error 10338

illegal use of asm inline function

This error is generated when you try to use entries or PC-relative data in inline assembly functions.

Example:

```
extern void e();
inline asm int GetD7()
{
 move.l d7,d0
 entry e
}
```

Fix

Do not use entry in inline assembly functions.

2.6.192 C/C++ Error 10339

illegal use of C++ feature in EC++

A C++ feature was used that is not available in the EC++ language subset (for example, templates, namespaces, or multiple inheritance).

Fix

Avoid features that are not supported by EC++, or disable the EC++ compatibility option in the C/C++ language preference panel.

2.6.193 C/C++ Error 10340

illegal use template argument dependent type 'T::*argument*'

The compiler found a template dependent type that cannot be resolved.

```
template <class T> int foo(typename T::x arg)
int i = foo<int>(1);
// error: illegal use template argument
// dependent type 'T::x'
```

Fix

Use template arguments that match template parameter requirements.

```
template <class T> int foo(typename T::x arg);
struct X { typedef int x; };
int i = foo<X>(1);
```

2.6.194 C/C++ Error 10342

inline function call *functionname* not inlined

A function is not inlined (for example, when the inline level is not large enough).

Fix

Use fewer inline functions, a higher inline expansion level, or disable this warning.

2.6.195 C/C++ Error 10343

inconsistent use of 'class' and 'struct' keywords

The compiler generates an error when you inconsistently use the keywords class and struct are used in an inconsistent manner.

In the example below, X is declared as a class, and then inconsistently declared as a struct.

```
#pragma warn_structclass on

class X;

struct X { int a; }; // warning: inconsistent use of 'class' and 'struct' keywords
```

Fix

Use struct and class keywords consistently.

```
class X;

class X { int a; };
```

2.6.196 C/C++ Error 10344

illegal partial specialization

There is an error in a partial class specialization.

```
template <typename T, typename U> class X;

template <typename T> class X<T,T,T>; // error
```

Fix

Use a partial specialization that matches the original template.

```
template <typename T, typename U> class X;

template <typename T> class X<T,T>;
```

2.6.197 C/C++ Error 10345

illegal partial specialization argument list

The compiler gives this error when you have an error in the partial class specialization argument list.

This error message is no longer used.

2.6.198 C/C++ Error 10346

ambiguous use of partial specialization

The compiler found more than one partial specialization that matched the class type.

```
template <typename T, typename U> class X;
template <typename T> class X<int,T>;
template <typename T> class X<T,int>;
X<int,int> xii;
```

Fix

Remove the ambiguous partial specializations.

```
template <typename T, typename U> class X;
template <typename T> class X<int,T>;
X<int,int> xii;
```

2.6.199 C/C++ Error 10347

local classes shall not have member templates

The compiler found a local class that mistakenly contains a member template.

```
void f()
{
 struct X {
 template <class T> void f(T) {}; // error
};
   ... // use X
}
```

Fix

Move the local class to the file scope.

```
struct fX {
 template <class T> void f(T) {};
};
void f()
{
 ... // use fX
}
```

2.6.200 C/C++ Error 10348

illegal template argument dependent expression

Template argument-dependent expression is used in an illegal way.

```
template <int I> struct X {
  I  i;   // error
};
```

Fix

Use template argument-dependent expression as shown.

```
template <int I> struct X {
 static const int i2 = I+2;
};
```

2.6.201 C/C++ Error 10349

implicit 'int 'is no longer supported in C++

This error occurs when you try to use an implicit **int** in C++ code. This option is no longer supported in C++.

```
main() // Error
{}
```

Fix

Add an int return type.

```
int main()
{}
```

2.6.202 C/C++ Error 10350

size pad byte(s) inserted after data member *name*

Optional warning for struct pad bytes (ANSI only).

```
#pragma warn_padding on
struct X {
 char c;
 int i;
};
```

Fix

Insert explicit pad bytes or disable warning.

```
#pragma warn_padding on
struct X {
 char c;
 char pad[3];
 int i;
};
```

2.6.203 C/C++ Error 10351

pure function *functionname* is not virtual

A pure function is not virtual.

```
struct X {
 int f() = 0;
};
```

Fix

Add 'virtual' specifier for pure functions.

```
struct X {
 virtual int f() = 0;
};
```

2.6.204 C/C++ Error 10352

illegal virtual function *functionname* in union

Unions cannot have virtual function members.

```
union X {
 virtual int f();
};
```

Fix

Add a virtual specifier for pure functions.

```
struct X {
 virtual int f() = 0;
};
```

2.6.205 C/C++ Error 10353

cannot pass 'void' or 'function' parameter

An attempt was made to pass a void function argument.

```
void f();
void g(...);
int main() {
 g(f());
}
```

Fix

Use the appropriate arguments.

2.6.206 C/C++ Error 10354

illegal static const member *argument* initialization

This error occurs for illegal static const member initializations.

```
struct X {
 static const int *ip = 0; // error: pointer init not allowed in ISO C++
};
```

Fix

Use legal static const member initializations.

```
struct X {
 static const int i = 0;
};
```

2.6.207 C/C++ Error 10355

typename is missing in template argument-dependent qualified type

Typename keyword is missing in template argument-dependent qualified type.

NOTE

All cases are not currently diagnosed.

```
template <typename T> struct X {
 T::type f();
};
```

Fix

Use **typename** for template argument-qualified types.

```
template <typename T> struct X {
 typename T::type f();
};
```

2.6.208 C/C++ Error 10356

more than one expression in non-class type conversion

There is more than one expression in a non-class type conversion.

```
int i = int(1,2);
```

Fix

Only use one expression in non-class type conversions.

```
int i = int(1);
```

2.6.209 C/C++ Error 10357

template non-type argument *object* does not have external linkage

```
template<int *ip> int f();

static int i;

int val = f<&i>(); // error: 'i' has internal linkage
```

Fix

Only use template arguments with external linkage.

```
template<int *ip> int f();

int i;

int val = f<&i>();
```

2.6.210 C/C++ Error 10358

illegal or unsupported__attribute__

An attempt was made to use an unsupported attribute.

```
int a __attribute__((bogus)); // error
```

Fix

Only use supported attributes.

2.6.211 C/C++ Error 10360

cannot create object file *filename*

Compiler cannot create output file.

Fix

Check destination drive, free space, access rights, and other parameters.

2.6.212 C/C++ Error 10361

error writing to object file *filename*

Compiler cannot write to output file.

Fix

Check destination drive, free space, access rights, and other parameters.

2.6.213 C/C++ Error 10364

__alignof__() is not supported for SOM classes

__alignof__() does not work for SOMObject derived classes.

Fix

Do not use `__alignof__()` for SOMObject derived classes.

2.6.214 C/C++ Error 10365

illegal macro argument name *argument*

The macro argument name is not allowed.

```
#define macro(a,a) a+a
// error: cannot reuse macro argument name 'a'
```

Fix

Use unique macro argument names.

```
#define macro(a,b) a+b
```

2.6.215 C/C++ Error 10366

case has an empty range of values

This error occurs for empty case ranges (non-ANSI language extension).

```
switch(i)
{
case 1 ... 0 : // error
}
```

Fix

Make sure that the range covers at least one value.

```
switch(i)
{
case 0 ... 10 :
}
```

2.6.216 C/C++ Error 10367

'long long' switch() is not supported

The long long type is not supported as a switch type on this platform.

Fix

Use a long int type instead.

2.6.217 C/C++ Error 10368

'long long' case range is not supported

A long long is not supported as a case type on this platform.

Fix

Use a long int type instead.

2.6.218 C/C++ Error 10369

expression has no side effect

```
#pragma warn_no_side_effect on|off| reset (default: off)
#pragma warn_no_side_effect on
```

The compiler encountered a statement that generates no effect

```
void foo(int a,int b) {
 a+b; //warning: expression has no side effect
}
```

Fix

Cast the statement using **void** to suppress this warning.

```
void foo(int a,int b) {
 (void)(a+b);  //'void' cast suppresses warning
}
```

2.6.219 C/C++ Error 10370

result of function call is not used

The compiler encountered a statement that calls a function without using its result.

For detailed information, see the C Compilers Reference, `#pragma warn_resultnotused`.

2.6.220 C/C++ Error 10371

illegal non-type template argument

This error occurs for illegal non-type template arguments.

```
template<int i> int f();
static int i;
int val = f<i*3>();  // error:'i*3' is not a legal template argument
```

Fix

Use non-type template arguments that conform with the ISO C++ specs.

```
template<int i> int f();
int val = f<3>();
```

2.6.221 C/C++ Error 10372

illegal use of abstract class *classname*

Illegal attempt to create an instance of an abstract class.

An abstract class is defined with at least one pure virtual method. An abstract class requires you to make a subclass that provides methods to replace any pure virtual methods.

```
struct X {
 virtual void f() =   0  ;
};
X x; // error
```

Fix

You can create instances only of non-abstract classes.

```
struct X {
 virtual void f();
};
X x;
```

2.6.222 C/C++ Error 10373

illegal use of 'template' prefix

The template prefix is used in an illegal context.

```
struct X {
 static int m;
};
int i = X::template m;
```

Fix

Remove template keyword.

```
struct X {
 static int m;
```

```
};
int i = X::m;
```

2.6.223 C/C++ Error 10374

template parameter/argument list mismatch

Mismatch in the template parameter/argument list.

This error message is no longer used.

2.6.224 C/C++ Error 10375

cannot find matching deallocation function for *variable*

The code includes an overloaded delete operator, however the operator is not used correctly.

```
struct X {
 void operator delete(void *,char *);
};
void f(X *xp)
{
 delete xp; // error
}
```

Fix

Fix the code for correct use or rewrite the overloaded operator.

2.6.225 C/C++ Error 10376

illegal operand *operand*

The operand type is not allowed in an operation.

```
extern char *xp;

int i = -xp; // error
```

Fix

Only use operands that are supported by the operation.

```
extern char *xp;

int i = -*xp;
```

2.6.226 C/C++ Error 10377

illegal operands *operandsymboloperand*

The operand type is not allowed in a binary operation.

```
extern char *xp;

int i = xp << 1; // error
```

Fix

Only use operands that are supported by the operation.

```
extern char *xp;

int i = *xp << 1;
```

2.6.227 C/C++ Error 10378

illegal use of default template-argument

Default template arguments are not always supported (such as in function templates).

```
template <typename T = int> void f();
```

Fix

Do not use default arguments in function templates.

2.6.228 C/C++ Error 10380

__uuidof() is not supported for SOM classes

__uuidof() does not work for SOMObject derived classes.

Fix

Do not use `__uuidof()` for SOMObject derived classes.

2.6.229 C/C++ Error 10381

illegal access from *classname* to protected/private member *membername*

An illegal access to a protected/private class member was detected.

```
class X {
private:
 static int m;
};
int i = X::m; // error
```

Fix

Make the member public.

```
class X {
 public:
 static int m;
};
int i = X::m;
```

2.6.230 C/C++ Error 10382

integral type is not large enough to hold pointer

This is an optional warning for illegal T* -> integral conversions.

```
#pragma warn_ptr_int_conv on
int i;
char c = (char)&i;
```

Fix

Use an integral type that is large enough for a pointer type.

```
#pragma warn_ptr_int_conv on

int i;

long c = (long)&i;
```

2.6.231 C/C++ Error 10383

unknown x86 assembler instruction mnemonic

The compiler encountered an unknown assembler mnemonic.

This error message is no longer used.

2.6.232 C/C++ Error 10384

illegal use of const/volatile function qualifier sequence

The const/volatile function qualifier can only be used for non-static member functions.

```
extern void f() const; // error
```

Fix

Remove the function qualifier sequence.

```
extern void f();
```

2.6.233 C/C++ Error 10385

illegal optimization level for this limited version of CodeWarrior

This version of the CodeWarrior IDE is limited in the level of optimizations that are allowed.

Fix

Use a lower optimization level. Contact sales for full version of product.

2.6.234 C/C++ Error 10386

no UUID defined for type name

The UUID of a struct/class/union type is being referenced with `__uuidof(type)` but there was no corresponding `__declspec(uuid("aaaaaaaa-bbbb-cccc-dddd-eeee-ffffffffffff"))` attribute on the type.

Fix

Supply a __declspec(uuid("...")) attribute for the struct/class/union declaration.

For example:

```
#include <windows.h>
struct __declspec(uuid("b722bcc5-4e68-101b-a2bc-00aa00404770")) foo
{};
static const IID x = external_guid;
int main(void)
{if (__uuidof(foo) == x) { } else { }
}
```

2.6.235 C/C++ Error 10387

using implicit copy assignment for class with const or reference member *variable*

An assignment was attempted incorrectly for a const or reference member.

```
struct A
{
 const int mem;
 A(int i) : mem(i) { }
};
int main(void)
{
 A a(5);
 a = A(7); // error
}
```

Fix

Define an assignment operator function.

2.6.236 C/C++ Error 10388

unimplemented assembler instruction/directive

(Target-specific) The assembly code is unimplemented for this inline assembler.

Fix

Use a supported assembly code instruction.

2.6.237 C/C++ Error 10389

override of dll import function *symbol* only has application scope

You may redefine a symbol declared as `__declspec(dllimport)` to override the version from the DLL. Override the DLL's export by defining both symbols found in the export library (such as `_foo` and `__imp_foo`) such that the DLL version is never seen. The override has an affect on references from the application to itself. The DLL prelinked references to the symbol still use the original definition. Any other DLLs also use the original definition. This enables you to override standard functions such as operator new/delete, without the compiler complaining that they were declared as DLL imports (dllimport).

```
__declspec(dllimport) void* operator new(size_t);

....

void *operator new(size_t sz) { return 0L; }
```

The example above shows the function-level overrides of the standard C library (i.e., in the MSL runtime DLL which is a requirement of the ANSI standard).

Fix

The warning can be controlled with " `#pragma warn_dll_override on|off|reset`".

2.6.238 C/C++ Error 10390

illegal combination of operands in inline statement at line *number*

The inline statement incorporates a combination of operands. This usage structure is not allowed.

Fix

Rewrite this as a non-inlined statement.

2.6.239 C/C++ Error 10391

illegal operand ininline statement at line number

The inline statement includes an operand that is not allowed.

Fix

Use a supported operand.

2.6.240 C/C++ Error 10392

function call name is ambiguous

A function call matches more than one overloaded function.

```
class X {};
int operator +(X,char);
int operator +(X,long);
int i = X() + 1; // error
```

Fix

Use type casts to disambiguate the function call.

```
class X {};
int operator +(X,char);
int operator +(X,long);
int i = X() + (char)1;
```

2.6.241 C/C++ Error 10393

name is not a member of class type

The compiler generates this error when you use C++ 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

The solutions are:

- Enable the Enable C++ Exceptions option in the C/C++ Language settings panel
- Turn off the direct_destruction pragma.

2.6.242 C/C++ Error 10394

immediate operand in inline statement at line *number* cannot span more than 8bits

This error message is currently not used.

2.6.243 C/C++ Error 10395

'__except' or '__finally ' expected

The __except or __finally clause is missing, or both are present.

Fix

You can write a combination of:

```
__try...__except
```

or

```
__try...__finally
```

It is not legal to use the following combination:

```
__try...__except...__finally
```

NOTE

The __except or __finally blocks can only be specified once. Refer to the MSDN Library at http://msdn.microsoft.com/library/default.asp for detailed information on structured exception handling.

2.6.244 C/C++ Error 10396

cannot mix structured exception handling and C++ exception handling

A function uses both the SEH "__try" and the C++ "try" exception statements.

```
void func(void)
{
try
 {
   throw "error";
 }
 catch (...)
 {
__try
  {
    *(char *)0 = 0;
  }
   __except(EXCEPTION_EXECUTE_HANDLER)
  {
     printf("error caught and crash averted);
  }
 }
}
```

Fix

Separate the function into two functions.

```
void funccatch(void)
{
  __try
  {
   *(char *)0 = 0;
  }
  __except(EXCEPTION_EXECUTE_HANDLER)
  {
```

```
    printf("error caught and crash averted);
   }
}
void func(void)
{
 try
 {
  throw "error";
 }
 catch (...)
 {
  funccatch();
 }
}
```

2.6.245 C/C++ Error 10397

illegal multiway transfer out of __try...__finally statement

The error indicates that a statement in a `__try` block somehow jumps either into the `__try` block or outside the `__try...__finally` statement.

Fix

Rearrange the code so this doesn't occur.

2.6.246 C/C++ Error 10398

illegal use of structured exception handling keyword outside of _try statement

One of the keywords:

```
__leave


__except
```

```
__finally
```

was used outside of a __try statement.

Or, one of the intrinsic functions:

```
AbnormalTermination() / abnormal_termination()


GetExceptionCode() / _exception_code()


GetExceptionInformation() / _exception_info
```

was used outside the appropriate part of a __try statement.

Fix

The solutions are:

- __leave may only be used inside the __try part of __try...__finally.
- __except or __finally may only be used after __try { }.
- AbnormalTermination or _abnormal_termination may only be used inside the __finally block.
- GetExceptionCode or _exception_code may only be used inside the __except(...) filter expression or inside the __except { } code.
- GetExceptionInformation or _exception_info may only be used inside the __except(...) filter expression.

2.6.247 C/C++ Error 10399

illegal use of __leave in__try...__except statement

2.6.248 C/C++ Error 10400

unions cannot have reference members

A union cannot have reference data members.

```
union X {
 int &ref; // error
};
```

Fix

Use pointers (shown below) instead of references. You may also use a struct.

```
union X {
 int *ptr;
};
```

2.6.249 C/C++ Error 10401

unions cannot have static data members

A union cannot have static data members.

```
union X {
 static int si; // error
};
```

Fix

Use a struct instead of a union.

```
struct X {
 static int si;
};
```

2.6.250 C/C++ Error 10402

unions cannot have nontrivial class members

A union cannot have non-trivial data members.

```
struct S {
 S(int);
 int m;
};
union U {
S s; // error
};
```

Fix

Only use trivial union members.

```
struct S {
 int m;
};
union U {
S s;
};
```

2.6.251 C/C++ Error 10403

unions cannot have base classes

A union cannot have base classes.

```
struct B {
 int m;
};
union U : B { // error
 int m;
};
```

Fix

Use a struct instead of a union.

```
struct B {
 int m;
};
struct U : B {
 int m;
};
```

2.6.252 C/C++ Error 10404

unions cannot be used as base classes

A union cannot be used as a base class.

```
union U {
 int m;
};
struct A : U {  // error
 int m;
};
```

Fix

Use a struct instead of a union.

```
struct U {
 int m;
};
struct A : U {
 int m;
};
```

2.6.253 C/C++ Error 10405

name is not a member of namespace *name*

A qualified name is not defined in the specified namespace.

```
namespace A { }
A::t var; // error
```

Fix

Define the name in namespace before using it.

```
namespace A {
 typedef int t;
}
A::t var;
```

2.6.254 C/C++ Error 10406

name is not a class/namespace name

A qualifying name is not defined.

```
B::t var; // error
```

Fix

Only use namespace or class names that have been previously defined.

```
namespace B {
 typedef int t;
}
B::t var;
```

2.6.255 C/C++ Error 10407

classname is not a class name

A qualifying name is not a class name.

```
B::t var;  // error
```

Fix

Only use class names that have been previously defined.

```
class B {
 typedef int t;
}
B::t var;
```

2.6.256 C/C++ Error 10408

recursive operator-> delegation

An `operator-> ()` function is used recursively.

```
struct X {
int m;
X& operator->();
```

```
};
int i = X()->m; // error
```

Fix

Remove or change the `operator->()` function to get rid of recursion.

```
struct X {
int m;
X* operator->();
};
int i = X()->m;
```

2.6.257 C/C++ Error 10409

illegal use of *typename*

The compiler detected *typename* used in an illegal context.

```
struct X {
 typedef int t;
};
int foo(X *xp)
{
 return xp->t(1); // error
}
```

Fix

Use *typename* appropriately.

```
struct X {
 typedef int t;
};
int foo(X *xp)
{
 return X::t(1);
}
```

2.6.258 C/C++ Error 10410

empty array must be last class/struct member

An empty array member (a non-ANSI language extension) must be the last data member.

```
struct A
{
 int value;
 int array[]; // non-ANSI C extension
 int last; // error
};
```

Fix

Remove data members that follow the empty array member or move before array member.

```
struct A
{
 int value;
 int last;
 int array[]; // non-ANSI C extension
};
```

2.6.259 C/C++ Error 10411

included file *filename* is spelled differently on disk *name*

This is a warning diagnostic from `#pragma warn_filenamecaps on`. An #include statement has succeeded, but the spelling or capitalization differs from that on disk. This is useful when building code on Win32 or MacOS in preparation for porting to operating systems with case-sensitive file systems.

NOTE

The pragma only checks user files (included with '#include *file*') by default; adding " `#pragma warn_filenamecaps_system on`" extends this to check.

```
'#include <file>' statements
/* this should issue warning: included file 'Stdio.H' is spelled differently on disk
```

```
('stdio.h')*/
  #pragma warn_filenamecaps on
  #pragma warn_filenamecaps_system on
  #include <Stdio.H>
  int main(void)
  {
  }
```

Fix

Spell the #include file in the accustomed way. In some instances the file may be capitalized incorrectly on disk. In this case, rename the disk file.

```
  /* capitalize in the standard way */
  #include <stdio.h>
  int main(void)
  {
  }
```

2.6.260 C/C++ Error 10412

illegal access from *name* to protected/private member *class::member*

The compiler detected that a protected/private class member was illegally accessed.

```
class X {
 private:
 int m;
};
int i = X().m; // error
```

Fix

Make member accessible.

```
class X {
public:
 int m;
};
int i = X().m;  // error
```

2.6.261 C/C++ Error 10413

illegal access from *name* to protected/private base class *classname*

This error is currently not used.

2.6.262 C/C++ Error 10414

no matching error(s) reported in error check mode

2.6.263 C/C++ Error 10415

ambiguous ?: expression, *typeA* can be converted to *typeB* and vice versa

The compiler detected ambiguous ?: expression operands.

```
bool b;
struct X {
 X(int);
 operator int();
};
int i = b ? X(1) : 2;  // ambiguous
```

Fix

Use explicit conversions to disambiguate the ?: expression.

```
bool b;
struct X {
 X(int);
 operator int();
};
int i = b ? int(X(1)) : 2;
```

2.6.264 C/C++ Error 10416

override is declared without 'virtual' keyword

This is enabled by a pragma.

* `#pragma warn_no_explicit_virtual on | off | reset (default: off)`

The compiler will emit a warning if an overriding function is not declared with a virtual keyword. This warning is not required by the ISO C++ standard, but it can be helpful to track down unwanted overrides.

```
#pragma warn_no_explicit_virtual on

struct A {
virtual void f();
};
struct B : A {
   void f(); // Warning : override 'B::f()' is declared without 'virtual' keyword
};
```

2.6.265 C/C++ Error 10417

illegal use of calling convention declarator

The calling convention declarator must appear immediately before the name of the function. Previous releases silently ignored the incorrect placement before the return type.

```
void __stdcall foo(); // right
__stdcall void foo(); // wrong
```

NOTE

In ports of MacOS code to Windows, these declarations might cause errors:

```
pascal void foo();
```

where pascal is #defined to __stdcall. This is incorrect. An alternative format should be used:

```
#if macintosh
#define F_PASCAL(ret) pascal ret
#elif _WIN32
#define F_PASCAL(ret) ret __stdcall
#else
#error ...
#endif
```

2.6.266 C/C++ Error 10418

calling convention ignored due to incompatible compiler options

(x86 only)

The calling convention keyword (i.e. `__3dcall`, `__ssecall`, `__sse2call`) requires an option not currently enabled.

Fix

Enable the appropriate Instruction Sets option in the x86 CodeGen target settings panel and recompile.

2.6.267 C/C++ Error 10419

illegal redefinition of UUID for *typename*

`__declspec(uuid("..."))` was used with two different UUID values for the same type.

Fix

All uses must declare the same UUID.

2.6.268 C/C++ Error 10420

using '#pragmaonce[on]' in a precompiled header

When "#pragma warn_pch_portability on" and "#pragma old_pragma_once off" are used, this warning is generated when a precompiled header contains "#pragma once [on]" directives.

Fix

The method of keeping track of these files relies on their full paths, so headers must be rebuilt on a different machine.

2.6.269 C/C++ Error 10421

ambiguous access from *type* to *type*

2.6.270 C/C++ Error 10422

allocation/deallocation functions shall be global scope or class members

```
#include <stddef.h>
namespace foo {
void *operator new(size_t); // error, see ISO C++ 3.7.3.1p1
}
```

2.6.271 C/C++ Error 10423

types that are declared in parameter lists go out of scope at the end of the function declaration/definition

```
int f(struct X); // not global struct X !
```

Unlike pre-3.2 compilers, the compiler now detects this illegal situation. An undefined struct/class/union type must be forward declared before being used in a function parameter list.

A function's parameter list serves as a unique scope, so a reference to an undefined type is absorbed into the parameter list and is not added to the outer scope.

Another example of the error:

```
int f(struct X); // undefined struct X
typedef struct X {..} X ; // new struct X
int f(struct X); // new type X, not the same as the former, thus a redeclaration error
```

or

```
int f(struct X); // not global struct X!
int f(struct X);  // different forward declaration, thus a redeclaration error
```

Fix

This is probably not what you want. Usually, a forward struct declaration will solve the problem:

```
struct X;

int f(struct X);
```

2.6.272 C/C++ Error 10424

internal scanner error

This is a catatstrophic internal error that should be sent as a bug report.

Enable the "Emit #line" options in C/C++ Preprocessor target setting panel. If it is possible to preprocess the offending source file without generating the error and to reproduce the error when compiling that preprocessed file, send in the preprocess.

Otherwise, it will be necessary to send in the original source file and any headers required to reproduce the bug.

Fix

Report to www.freescale.com/support.

2.6.273 C/C++ Error 10425

expression too complex

No longer used

2.6.274 C/C++ Error 10426

too many errors emitted, quitting

The compiler detected too many errors and aborted the process.

`#pragma maxerrorcount (<num> | off)` is used to control the count of errors emitted per compilation unit.

Fix

Examine the first few errors recorded.

2.6.275 C/C++ Error 10427

out of memory

Fix

Increase memory or optimize for less memory usage.

2.6.276 C/C++ Error 10428

cannot load main source file *filename*

The main source file (the file in the project or on the command line) could not be loaded. This may be due to the file having incorrect permissions, being moved or deleted while the project is open, or other reasons.

Fix

Close the project and IDE and retry.

2.6.277 C/C++ Error 10429

the file *filename* cannot be opened

A file referenced with "#include" could not be located or opened. The compiler asks the IDE to locate the file using the directories in the "Access paths" preference panel. See the IDE Help for details on the search.

Fix

Check your project's access paths, possibly enabling "Always search user paths" in the Access Paths target setting panel, if your source uses '#include <xyz>' and you want the access path to remain in the "User Paths" list. Also, try enabling the "Interpret foreign paths" option in the Access paths panel, if the filename includes directory separators not ordinary used on the host operating system.

On OS X, includes of files in a framework's "Headers" directory will work if the framework is added to the "Frameworks" directory. Otherwise, if the file is located elsewhere in the framework, you must add an access path pointing to that directory.

2.6.278 C/C++ Error 10430

system does not support the text encoding *name*, treating as ASCII

The source text encoding is not supported by the system libraries. The compiler may have guessed this encoding from a multibyte source file (when the "C/C++ Preprocessor" panel "Source Encoding" option is set to "Autodetect"), or you may have specified it with:

```
#pragma text_encoding("<encoding>")
```

In either case, the compiler is unable to interpret non-ASCII characters in the source and will pass the text through literally.

2.6.279 C/C++ Error 10431

unterminated comment

The compiler detected an unterminated comment.

Comments cannot begin in one file and end in another. Also, comments started inside an "#if" block should usually end before the "#else", "#elif", or "#endif", though this is not required.

Fix

Close comments with proper termination symbol, for example */

2.6.280 C/C++ Error 10432

unknown escape sequence

The compiler detected an unknown escape sequence.

Examples of legal escape sequences:

\a \b \t \v \f \n \r \e \x \u \U \<octal>

New for 3.2 is \uXXXX and \UXXXXYYYY which declare universal characters (Unicode characters).

2.6.281 C/C++ Error 10433

number expected

The compiler expected a number, either a decimal, hexadecimal, octal, or binary integer or a float.

Fix

Verify the syntax used for the directive.

2.6.282 C/C++ Error 10434

string expected

The compiler expected a string constant, in the form "...".

2.6.283 C/C++ Error 10435

corresponding #line reference

A message with this text will appear after another error or warning when the source file being compiled uses #line directives (to modify the source information for the debugger or through code generated by an external tool like lex or yacc).

With this message, you will be able to easily navigate to an error caused by the original source or the generated source.

Two #pragmas can control whether error messages show the "real" source or the "substituted" source file:

```
#pragma msg_show_lineref on|off|reset
```

If enabled, show error in source pointed to by #line.

```
#pragma msg_show_realref on|off|reset
```

If enabled, show error in actual source when #line ref exists.

2.6.284 C/C++ Error 10436

location of previous definition

When a macro is redefined, this message tells where the original definition was found.

2.6.285 C/C++ Error 10437

argument expected while expanding macro *name* (got *number*, wanted *number*

Expanded error reporting when a macro invocation uses too few parameters. "got *number*" means *number* arguments were read. "wanted *number*" means the macro expects *number* parameters only. The "wanted" text may read, for example, "wanted >2" when the last argument of the macro uses varargs (...).

2.6.286 C/C++ Error 10438

unexpected argument while expanding macro *name* (wanted *number*)

Expanded error reporting when a macro invocation specifies too many parameters. "wanted *number*" means the macro only accepts *number* parameters.

Watch out for embedded C99 struct/array initializations like "{3,4}". This string counts as two macro arguments. Only parentheses can shield commas inside parameter lists. Use "({3,4})" instead.

2.6.287 C/C++ Error 10439

invalid token pasting of *name* and *name*

According to the language standard, the '##' token pasting operator can only generate other tokens. This warning indicates when a paste was found that did not result in a token. The code will still compile as it did previously, but it may be a useful hint if the compiler emits an error later on.

Example

`3 ## 4` is the token 34, or `< ## < ## =` is the token `<<=`

however `- ## 4 ## 0` is not a token.

`-40` is two tokens, `-` and `40`.

Strictly speaking, two strings cannot be token pasted into another string, since the compiler elides adjacent strings in a later phase anyway, but Codewarrior supports this usage without a warning.

Fix

This warning can be controlled with

```
#pragma warn_illtokenpasting on|off|reset
```

2.6.288 C/C++ Error 10440

illegal token for integral constant expression

The compiler detected illegal token when parsing expressions inside #if or #elif expressions. Only integer constants and operations may be used in this context.

2.6.289 C/C++ Error 10441

illegal number

The language standard allows any sequence beginning with a number and followed by numbers and letters (and '.', 'e+', 'e-', etc.) to constitute a preprocessing number. This is later converted to an integer or float constant, at which point this message may be generated.

A common problem is leaving out spaces in this incorrect expression:

```
0xce+3
```

This is not `0xce + 3`, but a single token which is neither an integer nor a float.

Fix

Use `0xce + 3` with spaces before and after the + sign.

2.6.290 C/C++ Error 10442

invalid integer constant

The compiler detected character sequences that look like integers but are not valid integer constants. For example, `0xhi`

2.6.291 C/C++ Error 10443

invalid floating-point constant

The compiler detected character sequences that look like floats but are not valid floating-point constants.

For example, `304.33eDog`

2.6.292 C/C++ Error 10444

character is out of range

Error emitted when a wide character is stored or narrowed by a translation.

Example:

```
char *foo = "\u1234";
```

The '\u1234' character requires 16 bits, but 'char' is only 8 bits wide.

NOTE

As per the C99 standard, when constructing a narrow string parsed from multibyte source text in which a sequence of two or three bytes constitutes a Unicode code point, the compiler emits the original multibyte sequence instead of truncating the Unicode character.

2.6.293 C/C++ Error 10445

character cannot be represented in a 'long long'

Error when a multi-byte character constant is too long to store in an integral type. For example, a language extension allowing `TEXT' for MacOS.

2.6.294 C/C++ Error 10446

illegal universal character

The C99 and C++ standards do not allow the use of \uXXXX or \UXXXXXXXX sequences for universal characters that fit in the ASCII character set (thus destroying any semblance of orthogonality). This message is emitted as a warning, and only when ANSI strict mode is enabled.

2.6.295 C/C++ Error 10447

ASCII shift state expected

When decoding multibyte text that uses a shift state to allow ASCII text to be interpreted either as ASCII or as an extended character set (i.e. ISO-2022), this message indicates that the source text is not in the expected format and is probably mangled.

2.6.296 C/C++ Error 10448

multi-byte character constant

The compiler detected the language extension of packing multiple characters into a single narrow character constant (for example, TEXT for MacOS). This warning is not currently enabled.

2.6.297 C/C++ Error 10449

illegal multi-line string constant

The compiler detected the deprecated and illegal behavior of allowing a string constant to span multiple lines (interpreting line breaks as `\n`).

This is illegal:

```
"first line
second line"
```

Use the format:

```
"first line\n\
second line"
```

or

```
"first line\n"
"second line"
```

instead.

2.6.298 C/C++ Error 10450

loss of precision in floating-point constant

Warning emitted when the hexadecimal floating point constant specifies more digits than the compiler can represent internally.

Example:

... constant in the form 0xXXXXpYY specifies more precision in the mantissa than ...

2.6.299 C/C++ Error 10451

nested comment detected

The compiler detected a nested comment:

/*

```
/* foo */
*/
```

Nested comment syntax is not allowed.

2.6.300 C/C++ Error 10452

illegal universal character sequence

The C99 and C++ standards do not allow the use of \uXXXX or \UXXXXXXXX sequences for universal characters that fit in the ASCII character set (thus destroying any semblance of orthogonality). This message is emitted as a warning, and only when ANSI strict mode is enabled.

2.6.301 C/C++ Error 10453

illegal UTF-8 sequence, treating as raw bytes

Source text the compiler had previously identified at UTF-8 contains illegal character sequences. As a failsafe, the bytes are passed literally to the compiler.

2.6.302 C/C++ Error 10454

the (elided) copy constructor is not callable because the reference parameter cannot be bound to an rvalue

```
struct A {
     A();
     A(A&);
};
void f(const A&);
int main()
{
     f( A() ); // error
}
```

2.6.303 C/C++ Error 10455

could not create or write file *name* for instrumentation (*name*)

Error condition when invoking CodeTEST under control of `#pragma instrument`.

2.6.304 C/C++ Error 10456

could not read file *name* for instrumentation (*name*)

Error condition when invoking CodeTEST under control of `#pragma instrument`.

2.6.305 C/C++ Error 10457

cannot locate instrumenter *name* (PATH=*pathname*, AMC_HOME=*name*)

Error condition when invoking CodeTEST under control of `#pragma instrument`.

2.6.306 C/C++ Error 10458

cannot execute instrumenter *name* (*name*) (command line: *name*) *name*

Error condition when invoking CodeTEST under control of `#pragma instrument`.

2.6.307 C/C++ Error 10459

instrumenter *name* failed with exit code *name* (command line: *name*) *name*

Error condition when invoking CodeTEST under control of `#pragma instrument`.

2.6.308 C/C++ Error 10460

cannot use precompiled header *name* while instrumenting

Error condition when invoking CodeTEST under control of `#pragma instrument`.

2.6.309 C/C++ Error 10461

compatible IDB list changed; you may want to remove object code and rebuild this target

The list of *.idb files included in a project changed.

Fix

The current target must be rebuilt to properly incorporate references to those files.

2.6.310 C/C++ Error 10462

undefined macro *name* used in #if or #elif conditional

Refer to `#pragma warn_undefmacro`

2.6.311 C/C++ Error 10463

combining character detected at beginning of identifier

Warning emitted when a Unicode combining character is detected in an unlikely position at the start of an identifier.

2.6.312 C/C++ Error 10464

extended universal character used in identifier

The C++ 98 and C99 standards define a set of Unicode characters allowed in identifiers. These apply to an old version of Unicode, so Codewarrior allows an extended set of identifier characters adapted from Unicode 3.2. This warning points out use of such characters. We expect the language standards to include these characters in the future.

2.6.313 C/C++ Error 10465

cannot locate prefix file *name*

Enhancement to error message "file cannot be opened" indicating that the prefix file specified in the C/C++ Language Panel in target settings panels cannot be loaded.

2.6.314 C/C++ Error 10466

illegal option *name*

Generated when the "illegal #pragmas" warning is enabled (accessible in the C/C++ Warnings target setting panel) and __option(xxx) is tested for a nonexistent option "xxx". As before, the value returned by the __option() is 0.

2.6.315 C/C++ Error 10467

cannot query option *name*

Generated when the "illegal #pragmas" warning is enabled (accessible in the C/C++ Warnings target setting panel) and __option(xxx) is tested for an option whose value cannot be tested, for example, for any "#pragma xxx" that is a command. As before, the value returned by the __option() is 0.

2.6.316 C/C++ Error 10468

object *name* hidden by local declaration

Refer to `#pragma warn_hiddenlocals`

2.6.317 C/C++ Error 10469

complex types are not implemented

The C99 complex data types are not supported at this point.

2.6.318 C/C++ Error 10470

variable length arrays cannot be used in function template prototypes or local template typedefs

Refer to support for C99 variable length arrays.

2.6.319 C/C++ Error 10471

variable length array types can only be used in local or function prototype scope

Refer to support for C99 variable length arrays.

2.6.320 C/C++ Error 10472

the local type *name* cannot be used in template arguments

```
template <typename T> void f(T);
        int main()
        {
             struct X { int x; } x = { 0 };
             f(x);   //<<< error
        }
```

2.6.321 C/C++ Error 10473

< expected (you may have accidentally used a <: token)

In C++, the character sequence <: is a digraph for [

Insert a space after the opening < of a template argument list if you see this error.

Example:

```
        typedef int INT;
        template <typename T> class X;
        X<::INT> *xp;   // error, is token sequence "X [ : INT > * xp ;"
=>
        Error   : '<' expected (you may have accidentally used a <: token)
        Test.cp line 3   X<::INT> *xp;
```

Workaround:

```
        typedef int INT;
        template <typename T> class X;
        X< ::INT> *xp;  // OK, whitespace separates < and :: tokens
```

2.6.322 C/C++ Error 10474

template argument list expected

Example:

```
      template <typename T> class X;
      X *xp;          // <<<
=>
      Error   : template argument list expected
      Test.cp line 2   X *xp;
```

Workaround:

```
      template <typename T> class X;
      X<int> *xp;     // OK, has a matching template argument list
```

2.6.323 C/C++ Error 10475

illegal bitfield size

Bitfield size is smaller than 0 or greater than the maximum size supported by the backend.

Example:

```
    struct X {
            int bf : 0; // 0 length named bitfield are illegal
    };
=>
    Error   : illegal bitfield size '0'
    Test.cp line 2    int bf : 0;
```

Workaround:

```
    struct X {
            int bf : 1;
    };
```

2.6.324 C/C++ Error 10476

invalid message number

In `#pragma warning on|off|reset (...)`, the message number is not recognized by the frontend or backend. Use `#pragma showmessagenumber on` and reproduce the desired warning to reveal the message number.

2.6.325 C/C++ Error 10477

could not generate intrinsic name due to incompatible arguments or compiler options

When the `__has_intrinsic(<name>)` built-in macro returns true but the compiler cannot generate intrinsic code, this message is reported.

2.6.326 C/C++ Error 10478

illegal macro name; *name* is a C++ keyword

In C++, operators like "and", "not", etc. are keywords and cannot be defined as macros. This is a warning unless ANSI strict is enabled, in which case it is an error.

2.6.327 C/C++ Error 10479

illegal macro name

The keyword "defined" may not be a macro name.

2.6.328 C/C++ Error 10480

deleting a void pointer is undefined

Using "delete (void*)x;" or similar is undefined behavior. Since it is not possible to allocate "void *" storage with "new", deleting "void *" is probably losing some information, meaning the appropriate destructors may not be called; or, "delete" is incorrectly paired with "malloc".

Example:

```
void f(void *p)
{
    delete p;       //      undefined
}
=>
Warning : deleting a void pointer is undefined
Test2.cp line 3    delete p;
```

Workaround:

```
void f(char *p)
{
```

```
        delete p;  //only use pointers to object types
    }
```

2.6.329 C/C++ Error 10481

cannot enable instance manager here

The template instance manager must be enabled before any declarations in the source, similar to the restrictions for using precompiled headers.

2.6.330 C/C++ Error 10482

some instances are missing; please rebuild

When using the instance manager, the compiler keeps track of which source files define which instances. If sources change so that instances are not available, and it is not possible to tell the IDE to rebuild files, this message will appear.

Usually this happens only when files are changing on disk or in the editor during the compile. Simply re-make the project.

Note that this behavior should never occur when building clean projects, and the message should never be reported after the next make.

2.6.331 C/C++ Error 10483

illegal or unsupported __declspec

The __declspec is recognized but not supported.

2.6.332 C/C++ Error 10484

illegal use of __declspec(*name*)

The __declspec is not used in the correct context. It is applied to a function instead of a typedef, etc.

2.6.333 C/C++ Error 10485

illegal use of function qualifier(s)

The code specified `__declspecs` or `__attribute__s` that only apply to functions.

2.6.334 C/C++ Error 10486

illegal use of data qualifier(s)

The code specified `__declspecs` or `__attribute__s` that only apply to data objects.

2.6.335 C/C++ Error 10487

pointer to integral conversion

Refer to `#pragma warn_any_ptr_int_conv on|off|reset`

2.6.336 C/C++ Error 10488

integral to pointer conversion

Refer to `#pragma warn_any_ptr_int_conv on|off|reset`

2.6.337 C/C++ Error 10489

cannot modify *name* after declarations have begun

Some compilers (at this time, only the Mach-O PowerPC) support #pragmas to modify the type of 'wchar_t' and 'bool' (#pragma ushort_wchar_t and #pragma uchar_bool).

These #pragmas may only be used in prefix text or at the top of a source file, to ensure that all references to the type use the same information.

2.6.338 C/C++ Error 10490

using non-POD classes in variable argument lists is undefined

When `#pragma warn_nonpod_vararg` is enabled and a function passes a non-"plain old data" class object -- one with virtual functions, multiple inheritance, or the like -- as an argument in a variable argument list, this warning indicates that the operation has undefined effects and is not portable.

2.6.339 C/C++ Error 10491

arguments to gcc style inline assembler exceeded maximum number supported (*number*)

In a GCC-style "asm" statement, the number of actual arguments exceeded the number allowed by the compiler.

Fix

Break up the statement into a list of shorter statements using fewer arguments, storing intermediate results in local variables.

2.6.340 C/C++ Error 10492

alternates in a single argument to gcc style inline assembler exceeded maximum number supported (*number*)

In a GCC-style "asm" statement, too many alternatives were provided for an argument.

Fix

Generally, alternatives are not used by Codewarrior, which processes the fully-substituted inline assembly directly, so disabling the alternatives when __MWERKS__ is defined may be sufficient.

2.6.341 C/C++ Error 10493

GCC style assembler processing could not match any of the constraints in *name*

In a GCC-style "asm" statement, an argument specifies a constraint that is not recognized by the compiler or which conflicts with the possible types for the supplied expression.

2.6.342 C/C++ Error 10494

GCC constraint *name* is not supported at this time

In a GCC-style "asm" statement, the supplied constraint for an operand is not supported by the compiler.

Fix

Use a different data type or simplify the instruction to use a basic constraint type (like 'g', 'r', 'f', or 'm'). CodeWarrior compilers may be able to optimize the generated inline assembly and come up with the desired operand format on its own.

2.6.343 C/C++ Error 10495

ignored attribute *name* due to conflict with calling convention

The attributes `__attribute__((pure))` and `__attribute__((const))` are ignored if a function modifies memory through a "hidden" argument returned on the stack, that is, when returning a struct.

2.6.344 C/C++ Error 10496

size of type is too large (maximum *num* bytes)

The compiler supports a maximum type size of 2 gigabytes.

Fix

If you are declaring a large array, use an empty array and modify your allocation strategy appropriately.

2.6.345 C/C++ Error 10497

whitespace expected after integer constant (at *location*)

An ambiguous number was detected, which can be fixed by inserting a space at the appropriate position.

The language standard allows any sequence beginning with a number and followed by numbers and letters ('.', 'e+', 'e-', etc.) to constitute a "preprocessing number". This is later converted to an integer or float constant, at which point this message may be generated.

A common problem is leaving out spaces in this incorrect expression:

```
0xce+3
```

This is not "0xce" + "3", but a single token that is neither an integer nor a float.

Fix

Use "0xce + 3" with a space before and after the + sign.

2.6.346 C/C++ Error 10498

illegal use of expression statement outside function

A GCC-style expression statement may only be used inside a function.

```
#include <stdio.h>
static int y = ({ int x = func(); x*2; }); // illegal, must be inside a function
```

2.6.347 C/C++ Error 10499

A pointer/array type was expected for this operation instead of *typename*

This error occurs when a type mismatch occurs in which a pointer or array is expected.

2.6.348 C/C++ Error 10500

cannot write file '%u' (%u)

2.6.349 C/C++ Error 10501

direct struct members shall not have base classes

This error occurs if a structure has a member with a base class.

Example

```
#pragma cpp_extensions on
struct X {
  int x;
};
struct Y : X {
  int y;
};
struct Z {
  int z;
  Y;    //error: Y has a base class
};
```

Fix

Do not use a member that has a base class in a structure.

2.6.350 C/C++ Error 10502

anonymous unions/structs shall not have private/protected members

This error occurs if a union or a struct has a private or a protected member.

Example

```
struct X {
  union {
  private:
    int x;    //error: private data member
  };
};
```

Fix

Do not define a private or a protected member in a structure or a union.

2.6.351 C/C++ Error 10503

anonymous unions/structs shall only define non-static data members

This warning/error occurs if an anonymous union or struct defines non-static data members.

Example

```
class username {
public:
union {
  int a;
  char *p;
  class class_username {
    int c;
    int d;
  } s;
};
=>
Warning: anonymous unions/structs shall only define non-static data
members
class username {
public:
union {
  int a;
  char* p;
```

```
  void func_username()
  {
    return a;
  }
};
};
=>
Error: anonymous unions/structs shall only define non-static data
members
```

Fix

Specify the type/function definition outside the anonymous union.

2.6.352 C/C++ Error 10504

illegal object definition in precompiled header:\n'%o'

This error occurs if define extern/static data/code objects in precompiled headers.

Fix

Remove data/code object defintion from precompiled headers. Note that you cannot define extern/static data/code objects in precompiled headers.

2.6.353 C/C++ Error 10505

illegal overloading '%o'\nwas declared as '%t'\nnow declared as '%t'

Similar to C.C++ Error 10197

2.6.354 C/C++ Error 10507

the object '%o' has already been instantiated

This error/warning occurs if already instantiated objects are explicitly instantiated.

Example

```
template<typename T> void f() {}

template void f<int>();

template void f<int>();  // error: already instantiated in line 2

=>

Warning : the object 'f<int>()' has already been instantiated

RunTest.cpp line 3   template void f<int>();
```

Fix

Remove the extra instantiation.

2.6.355 C/C++ Error 10508

static assert check '%u' failed

This error indicates a compile-time assertion failure. For example, a user-defined condition generating an error message on the assertion failure. The __static_assert() function can be used in global, local, or in class scope.

Example

```
// experimental support for __static_assert ( <const-expr> , <string-
lit> ) ;

template<typename T> void f()

{

  __static_assert(

  sizeof(T) >= sizeof(int),

  "sizeof(T) is not >= sizeof(int)"

);

//            ...

}

int main()

{

  f<int>();               // OK

  f<char>();              // Error

}

=>

Error   : static assert check 'sizeof(T) is not >= sizeof(int)' failed

    (point of instantiation: 'main()')
```

```
        (instantiating: 'f<char>()')
  Test.cp line 3   sizeof(T) is not >= sizeof(int)");
```

Fix

Correct the `__static_assert()` statement using information given in the error description.

2.6.356 C/C++ Error 10509

shell returned error %i storing object data for '%u' (whichfile=%i)

This error indicates a File I/O error.

Fix

Rebuild the project.

2.6.357 C/C++ Error 10515

illegal IPA file format

This error indicates corrupt interprocedural analysis object data.

Fix

Rebuild the project.

2.6.358 C/C++ Error 10516

illegal IPA file version

This error indicates old interprocedural analysis object data.

Fix

Rebuild the project.

2.6.359 C/C++ Error 10517

the global object '%o' (%t)\nfrom '%u'\nconflicts with the object '%o' (%t)\nin '%u'

This warning occurs if an objects has conflicting definitions in different translation units.

Example

```
foo.cpp:

extern struct X { int a, b; } x;

int main()

{

  g = 0;

  x.a = 0;

}

bar.c:

struct X { int a, b, c; } x; // error: does not match foo.cpp

=>

Warning : the global type 'X' defined in 'S:\tests\TestWin\foo.cpp'
conflicts with a type in 'S:\tests\TestWin\bar.cpp'

Warning : the global object 'x' (X) from 'S:\tests\TestWin\foo.cpp'
conflicts with the object 'x' (X) in 'S:\tests\TestWin\bar.cpp'
```

Fix

Use matching declarations in the entire program.

2.6.360 C/C++ Error 10518

the global type '%t'\ndefinded in '%u'\nconflicts with a type in '%u'

This warning occurs if a type has conflicting definitions in different translation units.

Example

```
foo.cpp:

extern struct X { int a, b; } x;

int main()

{
```

```
  g = 0;

  x.a = 0;

}

bar.c:

struct X { int a, b, c; } x; // error: does not match foo.cpp

=>

Warning : the global type 'X' defined in 'S:\tests\TestWin\foo.cpp'
conflicts with a type in 'S:\tests\TestWin\bar.cpp'

Warning : the global object 'x' (X) from 'S:\tests\TestWin\foo.cpp'
conflicts with the object 'x' (X) in 'S:\tests\TestWin\bar.cpp'
```

Fix

Use matching declarations in the entire program.

2.6.361 C/C++ Error 10519

shell returned error %i storing object data for '%u' (whichfile=%i)

This error indicates a File I/O error.

Fix

Rebuild the project.

2.6.362 C/C++ Error 10534

expected formal macro argument after '#'

This error occurs if the preprocessor detects illegal tokens after '#'. The preprocessor now detects illegal tokens following '#' in function-like macros, as required by the C99 and ISO C++ standards.

Example

```
#define x3(y) ... #z ...// error: 'z' not a parameter

#define x4(y) ... #2 ...// error: '2' not a parameter

#define x5(y) ... #  ...// error: no token after '#'
```

Fix

Specify only named parameters after '#'.

2.6.363 C/C++ Error 10535

illegal or unsupported alignment value

This error occurs if the alignment attribute is set to an invalid value.

Example

```
__declspec(align(5)) struct S2 {

   int a, b, c, d;

} S;

=>

Error: illegal or unsupported alignment value  __declspec(align(5))
struct S2 {
```

Fix

Set the alignment to a valid value. Note that currently, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, and 8192 are supported.

2.6.364 C/C++ Error 10536

structs/classes with flexibile array members cannot be used as struct/class members or array elements

This error/warning occurs if a struct/class that has a flexible array member is used as member in another struct/class.

Example

```
struct username {

  int a;

  unsigned char fa[];

} s1;

struct username2 {

  int a;

  struct  username s;
```

```
} s2;

=>

Warning: structs/classes with flexibile array members cannot be used as struct/class
members or array elements - struct  username s;
```

Fix

Do not use a struct/class that has a flexible array member as member in another struct/class. In the example given above, specify the dimensions for the fa[] array.

2.6.365 C/C++ Error 10537

flexible array member in otherwise empty struct

This error/warning occurs if a struct has a flexible array as the only member.

Example

```
struct username {

    unsigned char fa[];

} s1;

=>

Warning: flexible arrays are not allowed in unions unsigned char fa[];
```

Fix

Add another member to the struct, or specify the array with set dimensions. In the example given above, set the dimension for the fa[] array.

2.6.366 C/C++ Error 10538

cannot initialize nested flexible array members

This error occurs if a nested flexible array member is initialized.

Example

```
struct username1 {

int a;

  unsigned char fa[];

} s1;
```

```
struct username2 {

  int a;

  struct  username1 s;

} s2;

void func ()

{

  struct username2 s = { 1, {1, {2, 3}} };

}

=>

Error: cannot initialize nested flexible array members  struct
username2 s = { 1, {1, {2, 3}} };
```

Fix

Set the dimension for the nested flexible array if you need to initialize it.

2.6.367 C/C++ Error 10539

intrinsic functions cannot be defined

This error occurs if you try to define an intrinsic function.

Example

```
int __abs(int val) { return val<0 ? -val : val; }//intrinsic PPC
function

=>

Error   : intrinsic functions cannot be defined

Test.c line 1   int __abs(int val) { return val<0 ? -val : val; }
```

Fix

Do not specify any definitions for the intrinsic functions.

2.6.368 C/C++ Error 10540

the exception specification of the function '%o'\n is more restrictive than the specification of the override '%o'

Example

```
struct B {
    virtual void f() throw (int, double);
};
struct D: B {
    void f();   // error: B::f has "throw (int, double)"
};
```

Fix

Make sure that the specification of the override function is not less restrictive than its specification of the exception block.

2.6.369 C/C++ Error 10541

the target exception specification '%e'\n is more restrictive than the source specification '%e'

Example

```
void (*pf1)();
void (*pf2)() throw(int);
void f()
{
  pf1 = pf2;  //  OK:  pf1  is less restrictive
  pf2 = pf1;  //  error:  pf2  is more restrictive
}
```

Fix

Make sure that the target exception specification matches the source specification.

2.6.370 C/C++ Error 10542

(corresponding point of instantiation)

This error message indicates a point of instantiation in a template instantiation error message.

Example

```
template<typename T> int f() {
  return T(0);
}
template<typename T> int g() {
  return f<T>();
}
int main() {
  return g<void>();
}
=>
Error   : illegal implicit conversion from 'void' to 'int'
 (point of instantiation: 'main()')
  (instantiating: 'g<void>()')
   (instantiating: 'f<void>()')
Test.cpp line 2    return T(0);
    (corresponding point of instantiation for 'f<void>()')
Test.cpp line 5    return f<T>();
    (corresponding point of instantiation for 'g<void>()')
Test.cpp line 8    return g<void>();
```

2.6.371 C/C++ Error 10543

(corresponding point of instantiation for '%t')

This error message indicates a point of instantiation in a template instantiation error message.

Example

```
template<typename T> int f() {
  return T(0);
}
template<typename T> int g() {
  return f<T>();
}
int main() {
```

```
  return g<void>();
}
=>
Error   : illegal implicit conversion from 'void' to 'int'
 (point of instantiation: 'main()')
  (instantiating: 'g<void>()')
   (instantiating: 'f<void>()')
Test.cpp line 2    return T(0);
    (corresponding point of instantiation for 'f<void>()')
Test.cpp line 5    return f<T>();
    (corresponding point of instantiation for 'g<void>()')
Test.cpp line 8    return g<void>();
```

2.6.372 C/C++ Error 10544

(corresponding point of instantiation for '%o')

This error message indicates a point of instantiation in a template instantiation error message.

Example

```
template<typename T> int f() {
  return T(0);
}
template<typename T> int g() {
  return f<T>();
}
int main() {
  return g<void>();
}
=>
Error   : illegal implicit conversion from 'void' to
'int'
 (point of instantiation: 'main()')
  (instantiating: 'g<void>()')
```

```
    (instantiating: 'f<void>()')
Test.cpp line 2    return T(0);
    (corresponding point of instantiation for 'f<void>()')
Test.cpp line 5    return f<T>();
    (corresponding point of instantiation for 'g<void>()')
Test.cpp line 8    return g<void>();
```

2.6.373 C/C++ Error 10545

possible unwanted use of object address

This warning occurs if an object address is used in a conditional expression.

Example

```
#pragma warn_possunwant on
int check();
int count;
int main()
{
  if(check)   //      possible error, should be "check()"
  ++count;
}
=>
Warning : possible unwanted use of object address
Test.c line 6    if(check) // possible error, should be check()
```

Fix

Make appropriate changes according to the error description.

2.6.374 C/C++ Error 10546

'main()' cannot be called or referenced by address

This error occurs if you call the main() function by address. Note that it is illegal to call main() or to use &main.

Fix

Use another function name.

2.6.375 C/C++ Error 10547

function declaration conflicts with using declaration for '%o'

Example

```
namespace A{
  void f(){}
}
using A::f;
void f(int);
void f(){}            // error
```

Fix

Change the conflicting declaration.

2.6.376 C/C++ Error 10548

explicit template specialization without 'template<>' prefix

This error occurs if explicit template specialization of a member function does not specify template <> before the function definition.

Example

```
template <typename T>
struct templateTest
{
    void a ();
};
void templateTest<int>::a () //error - template<> is required
{
}
```

Fix

Specify template <> before the function definition in explicit template specialization of a member function.

2.6.377 C/C++ Error 10549

illegal use of 'template<>' prefix

Example

```
template<typename T>
struct A {};
template<>
struct A<int> { void f(); };
template<> // error - A<int> is not a template
void A<int>::f() {}
```

Fix

Remove the template<> prefix from normal classes.

2.6.378 C/C++ Error 10550

the 'override' function '%o' does not override any inherited functions

This message indicates that the specified function is supposed to override a function, but does not. These kinds of error/warning messages might occur if you use __declspec() or __attribute__() specifiers for improved function override checking.

Example

```
struct A {
virtual __declspec(final) void vf1();
virtual void vf2();
};
struct B : A {
void vf1();                                                    // ERROR
};
```

```
struct C : A {

__declspec(override) void vf2();                // OK

__declspec(final) void vf3();                      // OK

};

struct D : A {

__declspec(override) void vf();                  // ERROR

};

struct E {

__declspec(override) void vf();                  // ERROR

};

=>

Error   : the 'final' function 'A::vf1()' is overridden by 'B::vf1()'

Test.cpp line 7   };

Error   : the 'override' function 'D::vf()' does not override any
inherited functions

Test.cpp line 14   };

Error   : the 'override' function 'E::vf()' does not override any
inherited functions

Test.cpp line 17   };
```

Fix

Either remove the 'override' specifier, or specify a function that can be overridden.

2.6.379 C/C++ Error 10551

the 'final' function '%o' is overridden by '%o'

This message indicates that a function declared with the 'final' specifier is attempted to be overridden. These kinds of error/warning messages might occur if you use __declspec() or __attribute__() specifiers for improved function override checking.

Example

```
struct A {

  virtual __declspec(final) void vf1();

  virtual void vf2();

};

struct B : A {
```

```
void vf1();                                          // ERROR
};
struct C : A {
__declspec(override) void vf2();                  // OK
__declspec(final) void vf3();                        // OK
};
struct D : A {
__declspec(override) void vf();                   // ERROR
};
struct E {
__declspec(override) void vf();                   // ERROR
};
=>
Error   : the 'final' function 'A::vf1()' is overridden by 'B::vf1()'
Test.cpp line 7   };
Error   : the 'override' function 'D::vf()' does not override any
inherited functions
Test.cpp line 14   };
Error   : the 'override' function 'E::vf()' does not override any
inherited functions
Test.cpp line 17   };
```

Fix

Declare the function with 'override' specifier instead of the 'final'.

2.6.380 C/C++ Error 10552

environment variable '%u' not defined: using empty string as default

This warning occurs if an undefined environment variable is used via the __env_var() function.

Example

```
char* username = __env_var(username)
=>
Warning: environment variable 'username' not defined: using empty
string as default
```

Issued if the environment variable is not defined

Fix

Define the environment variable.

2.6.381 C/C++ Error 10553

local variables/parameters ('%o') shall not be used in a default argument

This error occurs if a local variable or a parameter is used as default argument.

Example

```
void f(int x, int y = x);    //error
```

Fix

Do not use a local variable or a parameter as a default argument.

2.6.382 C/C++ Error 10554

no suitable copy-ctor for class '%t'

This error occurs if a class object cannot be copied because there is no matching copy constructor.

Example

```
struct X {
  X();
  X(X& a);
};
struct Y {
  Y();
  operator X();
};
X gx = Y(); // error: cannot be copied because the copy constructor is X::X(X&a)
```

Fix

In the example given above, use X::X(const X&a) to define the constructor.

2.6.383 C/C++ Error 10555

the type '::std::type_info' that is required for 'typeid' expressions is not defined (usually defined in the <typeinfo> header file)

Example

```
int main()
{
  typeid(int); //error
}
```

Fix

Define the ::std::type_info, which is usually defined in the <typeinfo> header file.

2.6.384 C/C++ Error 10556

no members allowed after a flexible array

This error/warning occurs if a flexible array member is not the last member of a struct/class.

Example

```
class username {
int a;
    unsigned char fa[];
int b;
} ;
=>
Warning: no members allowed after a flexible array - int b;
```

Fix

Define the struct/class in such a way that no member are defined after the flexible array. Alternatively, specify dimensions for the flexible array.

2.6.385 C/C++ Error 10557

'typename' is not required/legal in this context

Example

```
template<typename T> struct Test {
T f();
};
template<typename T> typename T f() { return 0; };
                     ^^^^^^^^

=>
Warning : 'typename' is not required/legal in this context
```

Fix

Make appropriate changes according to the error description.

2.6.386 C/C++ Error 10558

calling class method using an instance

This warning occurs when an instance method is called using a class or vice versa.

2.6.387 C/C++ Error 10559

calling instance method using class method

This warning occurs when an instance method is called using a class or vice versa.

2.6.388 C/C++ Error 10560

the const or reference class member '%u' is not initialized

This error occurs when the members of struct/class are not initialized.

Example

```
struct X {
  const int a;     //error
  const int&r };  //error
X p;
```

Fix

Make sure that the members are initialized to default values.

2.6.389 C/C++ Error 10561

the enumerator '%u' does not have a matching case label

This error occurs when compiler detect potentially missing cases in switch() statements that have an enumeration type selector and no default label.

Example

```
enum X { A,B,C,D };
#pragma warn_missing_enum_case on
int foo(X x)
{
  switch(x)
  {
    case A: return 1;
    case D: return 2;
  }
return 0;
}
=>
Warning : the enumerator 'B' does not have a matching case label
Test.cp line 12    return 0;
Warning : the enumerator 'C' does not have a matching case label
Test.cp line 12    return 0;
```

Fix

Make appropriate changes according to the warning description.

2.6.390 C/C++ Error 10562

C99 this cast is incompatible with the C99 typing rules and could make the alias by type analysis fail

This warning/error occurs when compiler detects potentially dangerous pointer casts.

Example

```
extern short s;

int *ip1 = (int*)&s;

#pragma warn_c99_alias_ptr_conv on

int *ip2 = (int*)&s;

#pragma warn_c99_alias_ptr_conv_as_error on

int *ip3 = (int*)&s;

=>

Warning : C99 this cast is incompatible with the C99 typing rules and
could make the alias by type analysis fail

Test.c line 4   int *ip2 = (int*)&s;

Error   : C99 this cast is incompatible with the C99 typing rules and
could make the alias by type analysis fail

Test.c line 6   int *ip3 = (int*)&s;
```

Fix

Make appropriate changes according to the warning/error description.

2.6.391 C/C++ Error 10563

identifier '%u' redeclared as '%t'

This error indicates that an identifier has been redeclared, or is undefined.

Example

```
struct X {};

X::X() {}

X::X() {}

void X::operator >> (int) {}
```

```
=>
Error   : undefined identifier 'X::X'
Test.cpp line 3   X::X() {}
Error   : undefined identifier 'X::X'
Test.cpp line 4   X::X() {}
Error   : undefined identifier 'X::operator >>(int)'
Test.cpp line 5   void X::operator >> (int) {}
```

Fix

Make appropriate changes according to the error description.

2.6.392 C/C++ Error 10564

identifier '%u' was originally declared as '%t'

This error indicates that an identifier has been redeclared, or is undefined.

Example

```
struct X {};
X::X() {}
X::X() {}
void X::operator >> (int) {}
=>
Error   : undefined identifier 'X::X'
Test.cpp line 3   X::X() {}
Error   : undefined identifier 'X::X'
Test.cpp line 4   X::X() {}
Error   : undefined identifier 'X::operator >>(int)'
Test.cpp line 5   void X::operator >> (int) {}
```

Fix

Make appropriate changes according to the error description.

2.6.393 C/C++ Error 10565

'typename' prefix used outside of template

This warning occurs when a 'template' prefix is used outside of a template and ISO templates are enabled.

Example

```
struct X {
  typedef int T;
};
typename X::T xt;
=>
Warning : 'typename' prefix used outside of template
Test.cp line 4   typename X::T xt;
```

Fix

Make sure that template prefix is used inside a template when ISO templates are enabled.

2.6.394 C/C++ Error 10566

initializing '%t' with braces is non-standard

This error occurs if single item elements in a struct are initialized within braces.

Example

```
MyStruct  gStruct1 = { {5,6,7,8,9}, {5}, {1, 2} }; //error - {5}
```

Fix

Do not use braces when initializing single item elements in a structure.

2.6.395 C/C++ Error 10567

declaration specifier conflict: %s

This error occurs if there is a conflict in the data type specified while declaring a variable.

Example

```
short long int i;
  =>
Error   : declaration specifier conflict: 'short long'
Test.c line 1    short long int i;
```

Fix

Make appropriate changes according to the error description.

2.6.396 C/C++ Error 10568

flexible arrays are not allowed in unions

This error occurs if a flexible array is specified in a union.

Example

```
union {
  int arr[];
} u;
=>
Error   : flexibe arrays are not allowed in unions
Test.c line 2     int arr[];
```

Fix

Remove the flexible array from the union definition.

2.6.397 C/C++ Error 10569

the parameter '%u' has not been declared

This error occurs if a parameter is used without declaration.

Example

```
void f(a, b)
  int a;
{}
=>
```

```
Error   : the parameter 'b' has not been declared
Test.c line 3    {}
```

Fix

Declare the parameter.

2.6.398 C/C++ Error 10570

missing whitespace character

Example

```
#define A@B    // error
#define C @D  // OK
=>
Error   : missing whitespace character
Test.c line 1   #define A@B  // error
```

Fix

Specify a whitespace character as instructed by the error description.

2.6.399 C/C++ Error 10571

class/enum type declarations are not allowed in this context

Example

```
void f(struct X { int x; } x);
=>
Error   : class/enum type declarations are not allowed in this context
Test.cp line 1   void f(struct X { int x; } x);
```

Fix

Make appropriate changes as specified in the error description.

2.6.400 C/C++ Error 10572

local or unnamed classes shall not have static members

This error occurs if statis member is defined in an unnamed struct/class.

Example

```
struct {
static int i;
};
=>
Error   : local or unnamed classes shall not have static members
Test.cp line 2    static int i;
```

Fix

Specify a name for the struct/class that has the static member. Alternatively, make the static member, non-static.

2.6.401 C/C++ Error 10573

illegal pointer to constructor/destructor

Example

```
struct X {
  ~X();
};
int main()
{
  &X::~X;
}
=>
Error   : illegal pointer to constructor/destructor
Test.cp line 7    &X::~X;
```

Fix

Make appropriate changes as indicated in the error description.

2.6.402 C/C++ Error 10574

illegal template linkage

Example:

```
extern "C" {
  template<typename T> T f();
}
=>
Error   : illegal template linkage
Test.cp line 2    template<typename T> T f();
```

2.6.403 C/C++ Error 10575

catch(...) shall be the last handler for its try-block

This error occurs if the catch(...) is not the last handler in its try() block.

Example

```
int main()
{
try {}
  catch (...) {}
  catch (int) {}
}
=>
Error   : catch(...) shall be the last handler for its try-block
Test.cp line 5    catch (int) {}
```

Fix

Make sure that catch(...) is the last block for its try() block.

2.6.404 C/C++ Error 10576

the virtual member function '%o' is not defined and not declared pure

This error occurs if the compiler encounters an undefined virtual function that is not declared as pure.

Example

```
int main()

{

  class X { virtual void f(); } x;

}

=>

Error   : the virtual member function 'X::f()' is not defined and not
declared pure

Test.cp line 3    class X { virtual void f(); } x;
```

Fix

Make sure that that the virtual function is either defined or declared as pure.

2.6.405 C/C++ Error 10577

illegal jump into try/catch block

This occur indicates an invalid jump into try() or catch() block.

Example

```
int main()

{

  goto lab;

    try {

    lab:;

    }

catch(...) {}

}

=>
```

```
Warning : illegal jump into try/catch block

Test.cp line 3    goto lab;
```

Fix

Make sure that try() and catch() blocks are properly structured, and properly defined.

2.6.406 C/C++ Error 10578

this #pragma can only be used inside function bodies

This error occurs when a #pragma, such as #pragma loop_count ([<n1>, <n2>, <n3>, <n4>]) that is supported only inside a function body is used outside of a function.

Fix

Use the particular pragma only inside the function body.

2.6.407 C/C++ Error 10579

this #pragma could not be associated with a loop

This warning occurs if a loop #pragma is placed outside a loop.

Example

```
#define MAX 10

int func(void)

{

int i;

#pragma loop_unroll 0

for (i=0; i<10; i++) {

}

return i;

}

=>

Warning: this #pragma could not be associated with a loop
```

Fix

Use the particular pragma inside the loop body. For example,

```
for (i=0; i<10; i++) {
  #pragma loop_unroll 0
}
```

2.6.408 C/C++ Error 10580

object defined in a namespace that does not enclose the declaration's namespace

This error indicates an illegal or missing namespace declaration for the enclosed objects.

Example

```
namespace N {
  void f();
}
namespace M {
  void N::f() {}
}
=>
Warning : object defined in a namespace that does not enclose the declaration's namespace
Test.cp line 5    void N::f() {}
```

Fix

Correct the namespace declaration.

2.6.409 C/C++ Error 10581

deprecated object '%o' used

This error occurs if an object that is declared with the __attribute__((deprecated)) attribute is used.

Example

```
typedef int old_type __attribute__((deprecated));
old_type x;
void old_function(void) __attribute__((deprecated));
```

```
void foo(void)
{
  old_function();
}
=>
Warning : deprecated type 'old_type' used
Test.c line 2   old_type x;
Warning : deprecated object 'old_function()' used
Test.c line 6    old_function();
```

Fix

Do not use objects declared with the __attribute__((deprecated)) attribute.

2.6.410 C/C++ Error 10582

deprecated type '%u' used

This error occurs if a data type that is declared with the __attribute__((deprecated)) attribute is used.

Example

```
typedef int old_type __attribute__((deprecated));
old_type x;
void old_function(void) __attribute__((deprecated));
void foo(void)
{
  old_function();
}
=>
Warning : deprecated type 'old_type' used
Test.c line 2   old_type x;
Warning : deprecated object 'old_function()' used
Test.c line 6    old_function();
```

Fix

Do not use types declared with the __attribute__((deprecated)) attribute.

2.6.411 C/C++ Error 10583

non-type partial specialization arguments shall not involve template parameters except when the argument expression is a simple identifier

Example

```
template<int I, int J> struct A {};

template <int I> struct A<I, I+1> {};

=>

Warning : non-type partial specialization arguments shall not involve
template parameters except when the argument expression is a simple
identifier

Test.cp line 2   template<int I> struct A<I, I+1> {};
```

In the given example, "I + 2" is not a simple identifier.

Fix

The C++ language does not support writing this type of template code.

2.6.412 C/C++ Error 10584

the type of template parameter corresponding to a specialized non-type argument shall not be dependent on a parameter of the specialization

Example

```
template<typename T, T t> struct A {};

template<typename T> struct A<T, 1> {};

=>

Warning : the type of template parameter corresponding to a specialized
non-type argument shall not be dependent on a parameter of the
specialization

Test.cp line 2   template<typename T> struct A<T, 1> {};
```

In the given example, the type of 1 depends on T, which is not legal.

Fix

The C++ language does not support writing this type of template code.

2.6.413 C/C++ Error 10585

the partial specialization '%t' was used before it was declared

This error occurs if a partial template specialization is used before it is declared.

Example

```
template<typename T, typename U> struct A {};

A<int, int> aii;

template<typename T> struct A<T, T> {};

=>

Warning : the partial specialization 'A<int, int>' was used before it
was declared

Test.cp line 3   template<typename T> struct A<T, T> {};
```

Fix

Declare the partial specialization before use.

2.6.414 C/C++ Error 10590

Constant too big to be represented: %u

This error occurs if during the parsing of the application file, an integer constant is too big to be represented on 64 bits. This error signals an overflow.

Fix

Review application file.

2.6.415 C/C++ Error 10591

User defined calling convention '%u' is already defined

This warning occurs if the application file contains a user-defined calling convention that has the same name as a previously defined one.

Fix

Rename one of the calling conventions.

2.6.416 C/C++ Error 10592

'%u' calling convention is not defined

This warning occurs when adding an undefined calling convention to a function by name.

Fix

Review application file and make appropriate changes.

2.6.417 C/C++ Error 10593

Failed to parse application file: '%u'

This warning occurs if there are un-fixed parsing errors when the front-end component begins to process the application file.

Fix

Review previous error/warning messages and fix them.

2.6.418 C/C++ Error 10594

illegal function name in application file: '%u'\nexpecting '_%u'

This error occurs when compiler encounters certain functions with unmangled name while processing an application file at the module level. All mangled function names start with the _ character.

Fix

Use mangled function names in the application file. When using the C language, make sure that the function names begin with a _ character. When using the C++ language, review the map file.

2.6.419 C/C++ Error 10595

compound literals with destructors are not supported

Example

```
struct X { X(); ~X(); };
struct Y { X x; };
Y gy = (Y){ X() };
=>
Error   : compound literals with destructors are not supported
RunTest.cpp line 3   Y gy = (Y){ X() };
```

Fix

Make sure that the compound literals do not have destructors.

2.6.420 C/C++ Error 10599

illegal implicit conversion from %O to\n '%t'

This error indicates illegal implicit conversion.

Example:

```
void f();
void f(int);
int (*fp)() = f;
=>
Error   : illegal implicit conversion from { &f(), &f(int) } to
'int (*)()'
RunTest.cpp line 3   int (*fp)() = f;
```

Fix

Specify explicit conversion.

2.6.421 C/C++ Error 10603

illegal function overloading: more than one 'extern "C"'

This error indicates illegal function overloading.

Example

```
extern "C" {
void f();
void f(int);
}
=>
Error   : illegal function overloading: more than one 'extern "C"'
Test.cpp line 3    void f(int);
```

Fix

There can be only one extern C function. Therefore, remove the extra extern C functions, or replace them with extern C++ functions.

2.6.422 C/C++ Error 10604

illegal function overloading: cannot overload 'main()'

This error indicates that the main() function is overloaded.

Example

```
int main();
int main(int);
=>
Error   : illegal function overloading: cannot overload 'main()'
Test.cpp line 2   int main(int);
```

Fix

Specify only one main() function.

2.6.423 C/C++ Error 10605

attribute '%u' is not allowed in this context ('%u')

This error is reserved for C++1x attribute support.

2.6.424 C/C++ Error 10606

missing ')' in macro argument list

This error occurs in case of macros that have missing parenthesis.

Example

```
#define _mpylir(src1,src2) (L_dmpy(extract_l(src1),src2))
…
sum = X_add(sum, X_extend(_mpylir(DataIn[i],DataIn[i])));

=>

#   Error:                                  ^^^^^^^
#   missing ')' in macro argument list
```

Fix

Rewrite the macros not to have missing parenthesis.

2.6.425 C/C++ Error 10607

a cycle in the call graph is overriding this function's always_inline request

This warning occurs in case of detecting an infinite recursion with `__attribute__((always_inline))`.

Example

```
# pragma always_inline on

void function_always_inline1();
void function_always_inline2();
void function_always_inline1() __attribute__((always_inline))
{
            function_always_inline2();
}
void function_always_inline2() __attribute__((always_inline))
{
            function_always_inline1();
}
=>

# Warning: ^
#   a cycle in the call graph is overriding this function's always_inline request
```

Fix

Avoid recursion in function definitions with always_inline.

2.6.426 C/C++ Error 10608

output constraints shall begin with '=' or '+'

This error occurs in case of GCC-style assembly output operand does not begin with '=' or '+'.

Example

```
__ALWAYS_INLINE void _gcc_ass(Word40 __Da, Word40 __Db, Word40* __Dc, Word40* __Dd)
{
            __asm("abs.4t %2:%3, %0:%1": "d40" (*__Dc), "=d40" (*__Dd): "d40" (__Da), "d40"
(__Db));
}
=>

#   Error:                                  ^^^^^
#   output constraints shall begin with '=' or '+'
```

Fix

Any assembly output operand must begin with '=' or '+'.

2.6.427 C/C++ Error 10609

input constraints shall not begin with '=' or '+'

This error occurs in case of GCC-style assembly input operand begins with '=' or '+'.

Example

```
__ALWAYS_INLINE void _gcc_ass(Word40 __Da, Word40 __Db, Word40* __Dc, Word40* __Dd)
{
            __asm("abs.4t %2:%3, %0:%1": "=d40" (*__Dc), "=d40" (*__Dd): "=d40" (__Da),
"d40" (__Db));
}
=>

#   Error:                                                                  ^^^^^^
#   input constraints shall not begin with '=' or '+'
```

Fix

Any assembly input operand shall not begin with '=' or '+'.

2.6.428 C/C++ Error 10610

illegal redefinition of %s segment <%s, %s>

This error occurs in case of redefinition of a section with the same name but with different type.

Example

```
view Demo_View
section
            data = [
                                    MY_SECTION  : ".my_section"
                        ]
            bss = [
                                    MY_SECTION  : ".my_section"
            ]
            end section
end view
=>

#   Illegal redefinition of Bss segment <MY_SECTION, .my_section>
#
```

Fix

Use different names for sections.

2.6.429 C/C++ Error 10612

logical segment %s %s is not defined

This error occurs when assigning a section type to an undefined section name.

Example

```
view Demo_View
section
data = [
                                    MY_SECTION  : ".my_section"
                        ]
            end section
program = my_program
end view
=>

#   Logical segment Program my_program is not defined
#
```

Fix

Define all the used sections.

2.6.430 C/C++ Error 10613

the option %s is not supported for application file

This warning occurs when using an option that is not supported by the application file.

Example

```
view Demo_View
My_Const_To_Rom = TRUE
end view
=>

#   The option My_Const_To_Rom is not supported for application file
#
```

Fix

Do not use this option.

2.6.431 C/C++ Error 10614

overriding segment setting %s %s in %s

This warning occurs when setting a already set section to another segment.

Example

```
view Demo_View
section
data = [
                                    MY_SECTION  : ".my_section",
                                    MY_SECTION1 : ".my_section1"
                        ]
            end section

            data = MY_SECTION
            data = MY_SECTION1
end view
=>

#   overriding segment setting Data MY_SECTION1 in Demo_View
#
```

Fix

Delete the first assignment.

2.6.432 C/C++ Error 10615

stray character 0x%x (ASCII value) in program

This error occurs when an illegal stray character is used.

Example

```
int main(void)
{
            int a;
            a = £;
            return 0;
}
=>

#   Error:      ^
#   Stray character 0xa3 (ASCII value) in program
```

Fix

Do not use this character.

2.6.433 C/C++ Error 10616

component Compiler: Cannot find component %s => Skipping...

This warning occurs when the user tries to use a component that it is not defined in the application file.

Example

```
configuration
            component "Adder"
                        entries = [_myadd]
            end component
            use component Multiplier
end configuration
=>

#   Component Compiler: Cannot find component Multiplier => Skipping...
```

Fix

Use another component or defined the new component

2.6.434 C/C++ Error 10617

no component name selected, Skipping component...

This warning occurs when the user does not defined any component to use when compiling a self component (through application file or through command line).

Example

```
configuration
            component "Adder"
                        entries = [_myadd]
            end component
end configuration

Command line: scc -arch b4860 -Og -mb -be -O3 test.c -Wall -ma a.appli
=>

#   No component name selected, Skipping component...
```

Fix

Define a component to use (through application file or though command line: `-complib Multiplier`).

2.6.435 C/C++ Error 10618

component Compiler: Cannot find arg line component %s

This warning occurs when the user does not defined any component to use when compiling a self component (through application file or through command line).

Example

```
configuration
            component "Adder"
                        entries = [_myadd]
            end component
end configuration

Command line: scc -arch b4860 -Og -mb -be -O3 test.c -Wall -complib Multiplier -ma a.appli
=>

#   Component Compiler: Cannot find component Multiplier => Skipping...
```

Fix

Define the component in the application file or use another component already defined.

2.6.436 C/C++ Error 10619

=> Trying with application file one: %s

This warning occurs when the component defined in the command line is not found and the compiler tries to use the component in the application file.

Example

```
configuration
            component "Adder"
                        entries = [_myadd]
            end component
            use component Adder
end configuration

Command line: scc -arch b4860 -Og -mb -be -O3 test.c -Wall -complib Multiplier -ma a.appli
=>

#    => Trying with application file one: Adder
```

Fix

Remove the command line component or use one component defined in the application file.

2.6.437 C/C++ Error 10620

component %s already exists, Appending information to existing definition

This warning occurs when two components with the same name are defined in the application file.

Example

```
configuration
            component "Adder"
                        entries = [_myadd]
            end component
            component "Adder"
                        entries = [_myadd_1]
            end component
            use component Adder
end configuration

=>

#   Component Adder already exists, Appending information to existing definition
```

Fix

Merge the two components.

2.6.438 C/C++ Error 10621

no component LCF found for component %s

This error occurs when the compiler cannot figure out which is the name of the component LCF.

Example

```
Command line: scc -arch b4860 -Og -mb -be -O3 test.c -Wall -ma a.appli
=>

#   No component LCF found for component Adder
```

Fix

There are two ways to specify the name of the component LCF:

- `-scc -arch b4860 -Og -mb -be -O3 test.c -Wall -complib Adder -ma a.appli` – in this case the LCF is a file with a name generate by compiler
- `-scc -arch b4860 -Og -mb -be -O3 test.c -Wall -ma a.appli -Xcfe "-comp lcf_filename"` – in this case the LCF has tha name "`lcf_filename`"

2.6.439 C/C++ Error 10622

The type of the section '%s' is %s, while the definition of variable '%s' expects to be %s

This warning occurs if sections defined in application file and through pragmas/attributes have different types.

Fix

Rename the placing of variables in sections in order to avoid such warnings.

2.6.440 C/C++ Error 10623

The type of the section '%s' is %s, while it is expected to be %s in file %s

This warning occurs if sections defined in application file and through pragmas/attributes have different types.

Fix

Rename the placing of variables in sections in order to avoid such warnings.

2.6.441 C/C++ Error 10624

'%o' function declared 'noreturn' has a 'return' statement

This warning occurs if function defined as "noreturn" has a return statement.

Example

```
int abc(int x) __attribute__ ((noreturn))
{
       return x;
}
=>
# Warning:          ^
#   'abc(int)' function declared 'noreturn' has a 'return' statement
```

Fix

Remove the "noreturn" attribute or the return statement.

2.6.442 C/C++ Error 10625

'%o' function does return

This warning occurs if function defined as "never_return" has a return statement under a condition.

Example

```
int abc(int x)
{
#pragma never_return
       if (x)
              return x;
}
=>
# Warning: ^
#   'abc(int)' function does return
```

Fix

Remove the "never_return" pragma or the return statement.

Index

A

C

G

M

S

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number CWSCBTMREF
Revision 10.9.0, 06/2015