1 Open Watcom C Diagnostic Messages

The following is a list of all warning and error messages produced by the Open Watcom C compilers. Diagnostic messages are issued during compilation and execution.

The messages listed in the following sections contain references to `%s`, `%d` and `%u`. They represent strings that are substituted by the Open Watcom C compilers to make the error message more exact. `%d` and `%u` represent a string of digits; `%s` a string, usually a symbolic name.

Consider the following program, named `err.c`, which contains errors.

Example:

```
#include <stdio.h>

void main()
  {
    int i;
    float i;

    i = 383;
    x = 13143.0;
    printf( "Integer value is %d\n", i );
    printf( "Floating-point value is %f\n", x );
  }
```

If we compile the above program, the following messages will appear on the screen.

```
err.c(6): Error! E1034: Symbol 'i' already defined
err.c(9): Error! E1011: Symbol 'x' has not been declared
err.c: 12 lines, included 191, 0 warnings, 2 errors
```

The diagnostic messages consist of the following information:

1. the name of the file being compiled,
2. the line number of the line containing the error (in parentheses),
3. a message number, and
4. text explaining the nature of the error.

In the above example, the first error occurred on line 6 of the file `err.c.` Error number 1034 (with the appropriate substitutions) was diagnosed. The second error occurred on line 9 of the file `err.c.` Error number 1011 (with the appropriate substitutions) was diagnosed.

The following sections contain a complete list of the messages. Run-time messages (messages displayed during execution) do not have message numbers associated with them.

1.1 Warning Level 1 Messages

W100 *Parameter %d contains inconsistent levels of indirection*

The function is expecting something like `char **` and it is being passed a `char *` for instance.

W101 *Non-portable pointer conversion*

This message is issued whenever you convert a non-zero constant to a pointer.

W102 *Type mismatch (warning)*

This message is issued for a function return value, an assignment or operators where one type is pointer and second one is non-pointer type.

W103 *Parameter count does not agree with previous definition (warning)*

You have either not enough parameters or too many parameters in a call to a function. If the function is supposed to have a variable number of parameters, then you can ignore this warning, or you can change the function declaration and prototypes to use the ",..." to indicate that the function indeed takes a variable number of parameters.

W104 *Inconsistent levels of indirection*

This occurs in an assignment or return statement when one of the operands has more levels of indirection than the other operand. For example, a `char **` is being assigned to a `char *.`

Solution: Correct the levels of indirection or use a `void *.`

W105 *Assignment found in boolean expression*

An assignment of a constant has been detected in a boolean expression. For example: "if(var = 0)". It is most likely that you want to use "==" for testing for equality.

W106 *Constant out of range - truncated*

This message is issued if a constant cannot be represented in 32 bits or if a constant is outside the range of valid values that can be assigned to a variable.

W107 *Missing return value for function '%s'*

A function has been declared with a function return type, but no ***return*** statement was found in the function. Either add a ***return*** statement or change the function return type to ***void.***

W108 *Duplicate typedef already defined*

A duplicate typedef is not allowed in ISO C. This warning is issued when compiling with extensions enabled. You should delete the duplicate typedef definition.

W109 *not used*

unused message

W110 *'fortran' pragma not defined*

You have used the ***fortran*** keyword in your program, but have not defined a #pragma for ***fortran.***

W111 *Meaningless use of an expression*

The line contains an expression that does nothing useful. In the example "i = (1,5);", the expression "1," is meaningless.

W112 *Pointer truncated*

A far pointer is being passed to a function that is expecting a near pointer, or a far pointer is being assigned to a near pointer.

W113 *Pointer type mismatch*

You have two pointers that either point to different objects, or the pointers are of different size, or they have different modifiers.

W114 *Missing semicolon*

You are missing the semicolon ";" on the field definition just before the right curly brace "}".

W115 *&array may not produce intended result*

The type of the expression "&array" is different from the type of the expression "array". Suppose we have the declaration `char buffer[80]` Then the expression `(&buffer + 3)` will be evaluated as `(buffer + 3 * sizeof(buffer))` which is `(buffer + 3 * 80)` and not `(buffer + 3 * 1)` which is what most people expect to happen. The address of operator "&" is not required for getting the address of an array.

W116 *Attempt to return address of auto variable*

This warning usually indicates a serious programming error. When a function exits, the storage allocated on the stack for auto variables is released. This storage will be overwritten by further function calls and/or hardware interrupt service routines. Therefore, the data pointed to by the return value may be destroyed before your program has a chance to reference it or make a copy of it.

W117 *'##' tokens did not generate a single token (rest discarded)*

When two tokens are pasted together using ##, they must form a string that can be parsed as a single token.

W118 *Label '%s' has been defined but not referenced*

You have defined a label that is not referenced in a ***goto*** statement. It is possible that you are missing the ***case*** keyword when using an enumerated type name as a case in a ***switch*** statement. If not, then the label can be deleted.

W119 *Address of static function '%s' has been taken*

This warning may indicate a potential problem when the program is overlayed.

W120 *lvalue cast is not standard C*

A cast operation does not yield an lvalue in ISO C. However, to provide compatibility with code written prior to the availability of ISO compliant C compilers, if an expression was an lvalue prior to the cast operation, and the cast operation does not cause any conversions, the compiler treats the result as an lvalue and issues this warning.

W121 *Text following pre-processor directives is not standard C*

Arbitrary text is not allowed following a pre-processor directive. Only comments are allowed following a pre-processor directive.

W122 *Literal string too long for array - truncated*

The supplied literal string contains more characters than the specified dimension of the array. Either shorten the literal string, or increase the dimension of the array to hold all of the characters from the literal string.

W123 *'//' style comment continues on next line*

The compiler has detected a line continuation during the processing of a C++ style comment ("//"). The warning can be removed by switching to a C style comment ("/**/"). If you require the comment to be terminated at the end of the line, make sure that the backslash character is not the last character in the line.

Example:

```
#define XX 23 // comment start \
comment \
end

int x = XX; // comment start ...\
comment end
```

W124 *Comparison result always %d*

The line contains a comparison that is always true (1) or false (0). For example comparing an unsigned expression to see if it is >= 0 or < 0 is redundant. Check to see if the expression should be signed instead of unsigned.

W125 *Nested include depth of %d exceeded*

The number of nested include files has reached a preset limit, check for recursive include statements.

W126 *Constant must be zero for pointer compare*

A pointer is being compared using == or != to a non-zero constant.

W127 *trigraph found in string*

Trigraph expansion occurs inside a string literal. This warning can be disabled via the command line or ***#pragma warning*** directive.

Example:

```
// string expands to "(?]?????"!
char *e = "(???)???-????";
// possible work-arounds
char *f = "(" "???" ")" "???" "-" "????";
char *g = "(\?\?\?)\?\?\?-\?\?\?\?";
```

W128 *%d padding byte(s) added*

The compiler has added slack bytes to align a member to the correct offset.

W129 *#endif matches #if in different source file '%s'*

This warning may indicate a ***#endif*** nesting problem since the traditional usage of ***#if*** directives is confined to the same source file. This warning may often come before an error and it is hoped will provide information to solve a preprocessing directive problem.

W130 *Possible loss of precision*

This warning indicates that you may be converting a argument of one size to another, different size. For instance, you may be losing precision by passing a long argument to a function that takes a short. This warning is initially disabled. It must be explicitly enabled with *#pragma enable_message(130)* or option "-wce=130". It can be disabled later by using *#pragma disable_message(130).*

W131 *No prototype found for function '%s'*

A reference for a function appears in your program, but you do not have a prototype for that function defined. Implicit prototype will be used, but this will cause problems if the assumed prototype does not match actual function definition.

W132 *No storage class or type specified*

When declaring a data object, either storage class or data type must be given. If no type is specified, ***int*** is assumed. If no storage class is specified, the default depends on scope (see the *C Language Reference* for details). For instance

Example:
```
auto i;
```

is a valid declaration, as is

Example:
```
short i;
```

However,

Example:
```
i;
```

is not a correctly formed declaration.

W133 *Symbol name truncated for '%s'*

Symbol is longer than the object file format allows and has been truncated to fit. Maximum length is 255 characters for OMF and 1024 characters for COFF or ELF object files.

W134 *Shift amount negative*

The right operand of a left or right shift operator is a negative value. The result of the shift operation is undefined.

Example:
```
int a = 1 << -2;
```

The value of 'a' in the above example is undefined.

W135 *Shift amount too large*

The right operand of a left or right shift operator is a value greater than or equal to the width in bits of the type of the promoted left operand. The result of the shift operation is undefined.

Example:
```
int a = 1 >> 123;
```

The value of 'a' in the above example is undefined.

W136 *Comparison equivalent to 'unsigned == 0'*

Comparing an unsigned expression to see whether it is <= 0 is equivalent to testing for == 0. Check to see if the expression should be signed instead of unsigned.

W137 *Extern function '%s' redeclared as static*

The specified function was either explicitly or implicitly declared as ***extern*** and later redeclared as ***static.*** This is not allowed in ISO C and may produce unexpected results with ISO compliant compilers.

Example:

```
int bar( void );

void foo( void )
{
    bar();
}

static int bar( void )
{
    return( 0 );
}
```

W138 *No newline at end of file*

ISO C requires that a non-empty source file must include a newline character at the end of the last line. If no newline was found, it will be automatically inserted.

W139 *Divisor for modulo or division operation is zero*

The right operand of a division or modulo operation is zero. The result of this operation is undefined and you should rewrite the offending code to avoid divisions by zero.

Example:

```
int foo( void )
{
    return( 7 / 0 );
}
```

W140 *Definition of macro '%s' not identical to previous definition*

If a macro is defined more than once, the definitions must be identical. If you want to redefine a macro to have a different definition, you must `#undef` it before you can define it with a new definition.

W141 *message number '%d' is invalid*

The message number used in the #pragma does not match the message number for any warning message. This message can also indicate that a number or '*' (meaning all warnings) was not found when it was expected.

W142 *warning level must be an integer in range 0 to 5*

The new warning level that can be used for the warning can be in the range 0 to 5. The level 0 means that the warning will be treated as an error (compilation will not succeed). Levels 1 up to 5 are used to classify warnings. The -w option sets an upper limit on the level for warnings. By setting the level above the command line limit, you effectively ignore all cases where the warning shows up.

W143 *%s*

This is a user message generated by the ***#pragma message*** or by ***#warning*** preprocessor directive.

Example:

```
#pragma message( "my very own warning" );

or

#warning my very own warning
```

1.2 Warning Level 2 Messages

W200 *'%s' has been referenced but never assigned a value*

You have used the variable in an expression without previously assigning a value to that variable.

W201 *Unreachable code*

The statement will never be executed, because there is no path through the program that causes control to reach this statement.

W202 *Symbol '%s' has been defined, but not referenced*

There are no references to the declared variable. The declaration for the variable can be deleted.

In some cases, there may be a valid reason for retaining the variable. You can prevent the message from being issued through use of *#pragma off(unreferenced).*

W203 *Preprocessing symbol '%s' has not been declared*

The symbol has been used in a preprocessor expression. The compiler assumes the symbol has a value of 0 and continues. A `#define` may be required for the symbol, or you may have forgotten to include the file which contains a `#define` for the symbol.

1.3 Warning Level 3 Messages

W300 *Nested comment found in comment started on line %u*

While scanning a comment for its end, the compiler detected `/*` for the start of another comment. Nested comments are not allowed in ISO C. You may be missing the `*/` for the previous comment.

W301 *not used*

unused message

W302 *Expression is only useful for its side effects*

You have an expression that would have generated the warning "Meaningless use of an expression", except that it also contains a side-effect, such as ++, −−, or a function call.

W303 *Parameter '%s' has been defined, but not referenced*

There are no references to the declared parameter. The declaration for the parameter can be deleted. Since it is a parameter to a function, all calls to the function must also have the value for that parameter deleted.

In some cases, there may be a valid reason for retaining the parameter. You can prevent the message from being issued through use of *#pragma off(unreferenced).*

This warning is initially disabled. It must be specifically enabled with *#pragma enable_message(303)* or option "-wce=303". It can be disabled later by using *#pragma disable_message(303).*

W304 *Return type 'int' assumed for function '%s'*

If a function is declared without specifying return type, such as

Example:
```
foo( void );
```

then its return type will be assumed to be ***int***

W305 *Type 'int' assumed in declaration of '%s'*

If an object is declared without specifying its type, such as

Example:
```
register count;
```

then its type will be assumed to be ***int***

W306 *Assembler warning: '%s'*

A problem has been detected by the in-line assembler. The message indicates the problem detected.

W307 *Obsolete non-prototype declarator*

Function parameter declarations containing only empty parentheses, that is, non-prototype declarations, are an obsolescent language feature. Their use is dangerous and discouraged.

Example:

```
int func();
```

W308 *The function '%s' without prototyped parameters called*

A call to an unprototyped function was made, preventing the compiler from checking the number of function arguments and their types. Use of unprototyped functions is obsolescent, dangerous and discouraged.

Example:

```
int func();

void bar( void )
{
    func( 4, "s" );     /* possible argument mismatch */
}
```

W309 *The function without prototyped parameters indirectly called*

An indirect call to an unprototyped function was made, preventing the compiler from checking the number of function arguments and their types. Use of unprototyped functions is obsolescent, dangerous and discouraged.

Example:

```
int (*func)();

void bar( void )
{
    func( 4, "s" );     /* possible argument mismatch */
}
```

W310 *Pointer truncated during cast*

A far pointer is being cast to a near pointer, losing segment information in the process.

Example:

```
char __near *foo( char __far *fs )
{
    return( (char __near *)fs );
}
```

1.4 Warning Level 4 Messages

W400 *Array subscript is of type plain char*

Array subscript expression is of plain char type. Such expression may be interpreted as either signed or unsigned, depending on compiler settings. A different type should be chosen instead of char. A cast may be used in cases when the value of the expression is known to never fall outside the 0-127 range.

Example:

```
int foo( int arr[], char c )
{
    return( arr[c] );
}
```

1.5 Warning Level 5 Messages

W500 *undefined macro '%s' evaluates to 0*

The ISO C/C++ standard requires that undefined macros evaluate to zero during preprocessor expression evaluation. This default behaviour can often mask incorrectly spelled macro references. The warning is useful when used in critical environments where all macros will be defined.

Example:

```
#if _PRODUCTI0N // should be _PRODUCTION
#endif
```

1.6 Error Messages

E1000 *BREAK must appear in while, do, for or switch statement*

A ***break*** statement has been found in an illegal place in the program. You may be missing an opening brace `{` for a ***while, do, for*** or ***switch*** statement.

E1001 *CASE must appear in switch statement*

A ***case*** label has been found that is not inside a ***switch*** statement.

E1002 *CONTINUE must appear in while, do or for statement*

The ***continue*** statement must be inside a ***while, do*** or ***for*** statement. You may have too many `}` between the ***while, do*** or ***for*** statement and the ***continue*** statement.

E1003 *DEFAULT must appear in switch statement*

A ***default*** label has been found that is not inside a ***switch*** statement. You may have too many `}` between the start of the ***switch*** and the ***default*** label.

E1004 *Misplaced '}' or missing earlier '{'*

An extra `}` has been found which cannot be matched up with an earlier `{`.

E1005 *Misplaced #elif directive*

The `#elif` directive must be inside an `#if` preprocessing group and before the `#else` directive if present.

E1006 *Misplaced #else directive*

The `#else` directive must be inside an `#if` preprocessing group and follow all `#elif` directives if present.

E1007 *Misplaced #endif directive*

A preprocessing directive has been found without a matching `#if` directive. You either have an extra or you are missing an `#if` directive earlier in the file.

E1008 *Only 1 DEFAULT per switch allowed*

You cannot have more than one ***default*** label in a ***switch*** statement.

E1009 *Expecting '%s' but found '%s'*

A syntax error has been detected. The tokens displayed in the message should help you to determine the problem.

E1010 *Type mismatch*

For pointer subtraction, both pointers must point to the same type. For other operators, both expressions must be assignment compatible.

E1011 *Symbol '%s' has not been declared*

The compiler has found a symbol which has not been previously declared. The symbol may be spelled differently than the declaration, or you may need to `#include` a header file that contains the declaration.

E1012 *Expression is not a function*

The compiler has found an expression that looks like a function call, but it is not defined as a function.

E1013 *Constant variable cannot be modified*

An expression or statement has been found which modifies a variable which has been declared with the ***const*** keyword.

E1014 *Left operand must be an 'lvalue'*

The operand on the left side of an "=" sign must be a variable or memory location which can have a value assigned to it.

E1015 *'%s' is already defined as a variable*

You are trying to declare a function with the same name as a previously declared variable.

E1016 *Expecting identifier*

The token following "->" and "." operators must be the name of an identifier which appears in the struct or union identified by the operand preceding the "->" and "." operators.

E1017 *Label '%s' already defined*

All labels within a function must be unique.

E1018 *Label '%s' not defined in function*

A ***goto*** statement has referenced a label that is not defined in the function. Add the necessary label or check the spelling of the label(s) in the function.

E1019 *Tag '%s' already defined*

All ***struct, union*** and ***enum*** tag names must be unique.

E1020 *Dimension cannot be 0 or negative*

The dimension of an array must be positive and non-zero.

E1021 *Dimensions of multi-dimension array must be specified*

All dimensions of a multiple dimension array must be specified. The only exception is the first dimension which can declared as "[]".

E1022 *Missing or misspelled data type near '%s'*

The compiler has found an identifier that is not a predefined type or the name of a "typedef". Check the identifier for a spelling mistake.

E1023 *Storage class of parameter must be register or unspecified*

The only storage class allowed for a parameter declaration is ***register.***

E1024 *Declared symbol '%s' is not in parameter list*

Make sure that all the identifiers in the parameter list match those provided in the declarations between the start of the function and the opening brace "{".

E1025 *Parameter '%s' already declared*

A declaration for the specified parameter has already been processed.

E1026 *Invalid declarator*

A syntax error has occurred while parsing a declaration.

E1027 *Invalid storage class for function*

If a storage class is given for a function, it must be ***static*** or ***extern.***

E1028 *Variable '%s' cannot be void*

You cannot declare a ***void*** variable.

E1029 *Expression must be 'pointer to ...'*

An attempt has been made to de-reference (*) a variable or expression which is not declared to be a pointer.

E1030 *Cannot take the address of an rvalue*

You can only take the address of a variable or memory location.

E1031 *Name '%s' not found in struct/union %s*

The specified identifier is not one of the fields declared in the ***struct*** or ***union.*** Check that the field name is spelled correctly, or that you are pointing to the correct ***struct*** or ***union.***

E1032 *Expression for '.' must be a 'structure' or 'union'*

The compiler has encountered the pattern "expression" "." "field_name" where the expression is not a ***struct*** or ***union*** type.

E1033 *Expression for '->' must be 'pointer to struct or union'*

The compiler has encountered the pattern "expression" "->" "field_name" where the expression is not a pointer to ***struct*** or ***union*** type.

E1034 *Symbol '%s' already defined*

The specified symbol has already been defined.

E1035 *static function '%s' has not been defined*

A prototype has been found for a ***static*** function, but a definition for the ***static*** function has not been found in the file.

E1036 *Right operand of '%s' is a pointer*

The right operand of "+=" and "−=" cannot be a pointer. The right operand of "−" cannot be a pointer unless the left operand is also a pointer.

E1037 *Type cast must be a scalar type*

You cannot type cast an expression to be a ***struct, union,*** array or function.

E1038 *Expecting label for goto statement*

The ***goto*** statement requires the name of a label.

E1039 *Duplicate case value '%s' found*

Every case value in a ***switch*** statement must be unique.

E1040 *Field width too large*

The maximum field width allowed is 16 bits.

E1041 *Field width of 0 with symbol not allowed*

A bit field must be at least one bit in size.

E1042 *Field width must be positive*

You cannot have a negative field width.

E1043 *Invalid type specified for bit field*

The types allowed for bit fields are ***signed*** or ***unsigned*** varieties of ***char, short*** and ***int***

E1044 *Variable '%s' has incomplete type*

A full definition of a ***struct*** or ***union*** has not been given.

E1045 *Subscript on non-array*

One of the operands of "[]" must be an array.

E1046 *Incomplete comment started on line %u*

The compiler did not find `*/` to mark the end of a comment.

E1047 *Argument for # must be a macro parm*

The argument for the stringize operator "#" must be a macro parameter.

E1048 *Unknown preprocessing directive '#%s'*

An unrecognized preprocessing directive has been encountered. Check for correct spelling.

E1049 *Invalid #include directive*

A syntax error has been encountered in a `#include` directive.

E1050 *Not enough parameters given for macro '%s'*

You have not supplied enough parameters to the specified macro.

E1051 *Not expecting a return value for function '%s'*

The specified function is declared as a ***void*** function. Delete the ***return*** statement, or change the type of the function.

E1052 *Expression has void type*

You tried to use the value of a ***void*** expression inside another expression.

E1053 *Cannot take the address of a bit field*

The smallest addressable unit is a byte. You cannot take the address of a bit field.

E1054 *Expression must be constant*

The compiler expects a constant expression. This message can occur during static initialization if you are trying to initialize a non-pointer type with an address expression.

E1055 *Unable to open '%s'*

The file specified in an `#include` directive could not be located. Make sure that the file name is spelled correctly, or that the appropriate path for the file is included in the list of paths specified in the `INCLUDE` environment variable or the "-I" option on the command line.

E1056 *Too many parameters given for macro '%s'*

You have supplied too many parameters for the specified macro.

E1057 *Modifiers disagree with previous definition of '%s'*

You have more than one definition or prototype for the variable or function which have different type modifiers.

E1058 *Cannot use typedef '%s' as a variable*

The name of a typedef has been found when an operand or operator is expected. If you are trying to use a type cast, make sure there are parentheses around the type, otherwise check for a spelling mistake.

E1059 *Invalid storage class for non-local variable*

A variable with module scope cannot be defined with the storage class of ***auto*** or ***register***

E1060 *Invalid type*

An invalid combination of the following keywords has been specified in a type declaration: ***const, volatile, signed, unsigned, char, int, short, long, float*** and ***double***

E1061 *Expecting data or function declaration, but found '%s'*

The compiler is expecting the start of a data or function declaration. If you are only part way through a function, then you have too many closing braces "}".

E1062 *Inconsistent return type for function '%s'*

Two prototypes for the same function disagree.

E1063 *Missing operand*

An operand is required in the expression being parsed.

E1064 *Out of memory*

The compiler has run out of memory to store information about the file being compiled. Try reducing the number of data declarations and or the size of the file being compiled. Do not `#include` header files that are not required.

For the 16-bit C compiler, the "-d2" option causes the compiler to use more memory. Try compiling with the "-d1" option instead.

E1065 *Invalid character constant*

This message is issued for an improperly formed character constant.

E1066 *Cannot perform operation with pointer to void*

You cannot use a "pointer to void" with the operators +, −, ++, −−, += and −=.

E1067 *Cannot take address of variable with storage class 'register'*

If you want to take the address of a local variable, change the storage class from ***register*** to ***auto***

E1068 *Variable '%s' already initialized*

The specified variable has already been statically initialized.

E1069 *String literal not terminated before end of line*

A string literal is enclosed by double quote " characters.

The compiler did not find a closing double quote " or line continuation character \ before the end of a line or before the end of the source file.

E1070 *Data for aggregate type must be enclosed in curly braces*

When an array, struct or union is statically initialized, the data must be enclosed in curly braces {}.

E1071 *Type of parameter %d does not agree with previous definition*

The type of the specified parameter is incompatible with the prototype for that function. The following example illustrates a problem that can arise when the sequence of declarations is in the wrong order.

Example:

```
/* Uncommenting the following line will
   eliminate the error */
/* struct foo; */

void fn1( struct foo * );

struct foo {
    int     a,b;
};

void fn1( struct foo *bar )
{
    fn2( bar );
}
```

The problem can be corrected by reordering the sequence in which items are declared (by moving the description of the structure `foo` ahead of its first reference or by adding the indicated statement). This will assure that the first instance of structure `foo` is defined at the proper outer scope.

E1072 *Storage class disagrees with previous definition of '%s'*

The previous definition of the specified variable has a storage class of ***static.*** The current definition must have a storage class of ***static*** or ***extern***

Alternatively, a variable was previously declared as ***extern*** and later defined as ***static***

E1073 *Invalid option '%s'*

The specified option is not recognized by the compiler.

E1074 *Invalid optimization option '%s'*

The specified option is an unrecognized optimization option.

E1075 *Invalid memory model '%s'*

Memory model option must be one of "ms", "mm", "mc", "ml", "mh" or "mf" which selects the Small, Medium, Compact, Large, Huge or Flat memory model.

E1076 *Missing semicolon at end of declaration*

You are missing a semicolon ";" on the declaration just before the left curly brace "{".

E1077 *Missing '}'*

The compiler detected end of file before finding a right curly brace "}" to end the current function.

E1078 *Invalid type for switch expression*

The type of a switch expression must be integral.

E1079 *Expression must be integral*

An integral expression is required.

Example:

```
int foo( int a, float b, int *p )
{
  switch( a ) {
    case 1.3:       // must be integral
      return p[b];  // index not integer
    case 2:
      b <<= 2;      // can only shift integers
    default:
      return b;
  }
}
```

E1080 *Expression must be arithmetic*

Both operands of the "*", "/" and "%" operators must be arithmetic. The operand of the unary minus must also be arithmetic.

E1081 *Expression must be scalar type*

A scalar expression is required.

E1082 *Statement required after label*

The C language definition requires a statement following a label. You can use a null statement which consists of just a semicolon (";").

E1083 *Statement required after 'do'*

A statement is required between the ***do*** and ***while*** keywords.

E1084 *Statement required after 'case'*

The C language definition requires a statement following a ***case*** label. You can use a null statement which consists of just a semicolon (";").

E1085 *Statement required after 'default'*

The C language definition requires a statement following a ***default*** label. You can use a null statement which consists of just a semicolon (";").

E1086 *Expression too complicated, split it up and try again*

The expression contains too many levels of nested parentheses. Divide the expression up into two or more sub-expressions.

E1087 *Missing matching #endif directive*

You are missing a to terminate a `#if, #ifdef` or `#ifndef` preprocessing directive.

E1088 *Invalid macro definition, missing)*

The right parenthesis ")" is required for a function-like macro definition.

E1089 *Missing) for expansion of '%s' macro*

The compiler encountered end-of-file while collecting up the argument for a function-like macro. A right parenthesis ")" is required to mark the end of the argument(s) for a function-like macro.

E1090 *Invalid conversion*

A ***struct*** or ***union*** cannot be converted to anything. A ***float*** or ***double*** cannot be converted to a pointer and a pointer cannot be converted to a ***float*** or ***double***

E1091 *%s*

This is a user message generated with the `#error` preprocessing directive.

E1092 *Cannot define an array of functions*

You can have an array of pointers to functions, but not an array of functions.

E1093 *Function cannot return an array*

A function cannot return an array. You can return a pointer to an array.

E1094 *Function cannot return a function*

You cannot return a function. You can return a pointer to a function.

E1095 *Cannot take address of local variable in static initialization*

You cannot take the address of an ***auto*** variable at compile time.

E1096 *Inconsistent use of return statements*

The compiler has found a ***return*** statement which returns a value and a ***return*** statement that does not return a value both in the same function. The ***return*** statement which does not return a value needs to have a value specified to be consistent with the other ***return*** statement in the function.

E1097 *Missing ? or misplaced :*

The compiler has detected a syntax error related to the "?" and ":" operators. You may need parenthesis around the expressions involved so that it can be parsed correctly.

E1098 *Maximum struct or union size is 64K*

The size of a ***struct*** or ***union*** is limited to 64K so that the compiler can represent the offset of a member in a 16-bit register.

E1099 *Statement must be inside function. Probable cause: missing {*

The compiler has detected a statement such as ***for, while, switch,*** etc., which must be inside a function. You either have too many closing braces "}" or you are missing an opening brace "{" earlier in the function.

E1100 *not used*

unused message

E1101 *Cannot #undef '%s'*

The special macros `__LINE__`, `__FILE__`, `__DATE__`, `__TIME__`, `__STDC__`, `__FUNCTION__` and `__func__`, and the identifier "defined", cannot be deleted by the `#undef` directive.

E1102 *Cannot #define the name 'defined'*

You cannot define a macro called `defined`

E1103 *## must not be at start or end of replacement tokens*

There must be a token on each side of the "##" (token pasting) operator.

E1104 *Type cast not allowed in #if or #elif expression*

A type cast is not allowed in a preprocessor expression.

E1105 *'sizeof' not allowed in #if or #elif expression*

The ***sizeof*** operator is not allowed in a preprocessor expression.

E1106 *Cannot compare a struct or union*

A ***struct*** or ***union*** cannot be compared with "==" or "!=". You must compare each member of a ***struct*** or ***union*** to determine equality or inequality. If the ***struct*** or ***union*** is packed (has no holes in it for alignment purposes) then you can compare two structs using `memcmp`

E1107 *Enumerator list cannot be empty*

You must have at least one identifier in an ***enum*** list.

E1108 *Invalid floating-point constant*

The exponent part of the floating-point constant is not formed correctly.

E1109 *Cannot take sizeof a bit field*

The smallest object that you can ask for the size of is a char.

E1110 *Cannot initialize variable with storage class of extern*

A storage class of ***extern*** is used to associate the variable with its actual definition somewhere else in the program.

E1111 *Invalid storage class for parameter*

The only storage class allowed for a parameter is ***register***

E1112 *Initializer list cannot be empty*

An initializer list must have at least one item specified.

E1113 *Expression has incomplete type*

An attempt has been made to access a struct or union whose definition is not known, or an array whose dimensions are not known.

E1114 *Struct or union cannot contain itself*

You cannot have a ***struct*** or ***union*** contain itself. You can have a pointer in the ***struct*** which points to an instance of itself. Check for a missing "*" in the declaration.

E1115 *Incomplete enum declaration*

The enumeration tag has not been previously defined.

E1116 *An id list not allowed except for function definition*

A function prototype must contain type information.

E1117 *Must use 'va_start' macro inside function with variable parameters*

The `va_start` macro is used to setup access to the parameters in a function that takes a variable number of parameters. A function is defined with a variable number of parameters by declaring the last parameter in the function as "...".

E1118 ****FATAL*** %s*

A fatal error has been detected during code generation time. The type of error is displayed in the message.

E1119 *Internal compiler error %d*

A bug has been encountered in the C compiler. Please report the specified internal compiler error number and any other helpful details about the program being compiled to compiler developers so that we can fix the problem.

E1120 *Parameter number %d - invalid register in #pragma*

The designated registers cannot hold the value for the parameter.

E1121 *Procedure '%s' has invalid return register in #pragma*

The size of the return register does not match the size of the result returned by the function.

E1122 *Illegal register modified by '%s' #pragma*

For the 16-bit C compiler: The BP, CS, DS, and SS registers cannot be modified in small data models. The BP, CS, and SS registers cannot be modified in large data models.

For the 32-bit C compiler: The EBP, CS, DS, ES, and SS registers cannot be modified in flat memory models. The EBP, CS, DS, and SS registers cannot be modified in small data models. The EBP, CS, and SS registers cannot be modified in large data models.

E1123 *File must contain at least one external definition*

Every file must contain at least one global object, (either a data variable or a function). This message is only issued in strict ANSI mode (-za).

E1124 *Out of macro space*

The compiler ran out of memory for storing macro definitions.

E1125 *Keyboard interrupt detected*

The compile has been aborted with Ctrl/C or Ctrl/Break.

E1126 *Array, struct or union cannot be placed in a register*

Only scalar objects can be specified with the ***register*** class.

E1127 *Type required in parameter list*

If the first parameter in a function definition or prototype is defined with a type, then all of the parameters must have a type specified.

E1128 *Enum constant is out of range %s*

All of the constants must fit into appropriate value range.

E1129 *Type does not agree with previous definition of '%s'*

You have more than one definition of a variable or function that do not agree.

E1130 *Duplicate name '%s' not allowed in struct or union*

All the field names in a ***struct*** or ***union*** must be unique.

E1131 *Duplicate macro parameter '%s'*

The parameters specified in a macro definition must be unique.

E1132 *Unable to open work file: error code = %d*

The compiler tries to open a new work file by the name "__wrkN__.tmp" where N is the digit 0 to 9. This message will be issued if all of those files already exist.

E1133 *Write error on work file: error code = %d*

An error was encountered trying to write information to the work file. The disk could be full.

E1134 *Read error on work file: error code = %d*

An error was encountered trying to read information from the work file.

E1135 *Seek error on work file: error code = %d*

An error was encountered trying to seek to a position in the work file.

E1136 *not used*

unused message

E1137 *Out of enum space*

The compiler has run out of space allocated to store information on all of the ***enum*** constants defined in your program.

E1138 *Filename required on command line*

The name of a file to be compiled must be specified on the command line.

E1139 *Command line contains more than one file to compile*

You have more than one file name specified on the command line to be compiled. The compiler can only compile one file at a time. You can use the Open Watcom Compile and Link utility to compile multiple files with a single command.

E1140 *_leave must appear in a _try statement*

The ***_leave*** keyword must be inside a ***_try*** statement. The ***_leave*** keyword causes the program to jump to the start of the ***_finally*** block.

E1141 *Expecting end of line but found '%s'*

A syntax error has been detected. The token displayed in the message should help you determine the problem.

E1142 *Too many bytes specified in #pragma*

There is an internal limit on the number of bytes for in-line code that can be specified with a pragma. Try splitting the function into two or more smaller functions.

E1143 *Cannot resolve linkage conventions for routine '%s' #pragma*

The compiler cannot generate correct code for the specified routine because of register conflicts. Change the registers used by the parameters of the pragma.

E1144 *Symbol '%s' in pragma must be global*

The in-line code for a pragma can only reference a global variable or function. You can only reference a parameter or local variable by passing it as a parameter to the in-line code pragma.

E1145 *Internal compiler limit exceeded, break module into smaller pieces*

The compiler can handle 65535 quadruples, 65535 leaves, and 65535 symbol table entries and literal strings. If you exceed one of these limits, the program must be broken into smaller pieces until it is capable of being processed by the compiler.

E1146 *Invalid initializer for integer data type*

Integer data types (int and long) can be initialized with numeric expressions or address expressions that are the same size as the integer data type being initialized.

E1147 *Too many errors: compilation aborted*

The compiler stops compiling when the number of errors generated exceeds the error limit. The error limit can be set with the "-e" option. The default error limit is 20.

E1148 *Expecting identifier but found '%s'*

A syntax error has been detected. The token displayed in the message should help you determine the problem.

E1149 *Expecting constant but found '%s'*

The #line directive must be followed by a constant indicating the desired line number.

E1150 *Expecting \"filename\" but found '%s'*

The second argument of the #line directive must be a filename enclosed in quotes.

E1151 *Parameter count does not agree with previous definition*

You have either not enough parameters or too many parameters in a call to a function. If the function is supposed to have a variable number of parameters, then you are missing the ", ..." in the function prototype.

E1152 *Segment name required*

A segment name must be supplied in the form of a literal string to the __segname() directive.

E1153 *Invalid __based declaration*

The compiler could not recognize one of the allowable forms of __based declarations. See the *C Language Reference* document for description of all the allowable forms of __based declarations.

E1154 *Variable for __based declaration must be of type __segment or pointer*

A based pointer declaration must be based on a simple variable of type __segment or pointer.

E1155 *Duplicate external symbol %s*

Duplicate external symbols will exist when the specified symbol name is truncated to 8 characters.

E1156 *Assembler error: '%s'*

An error has been detected by the in-line assembler. The message indicates the error detected.

E1157 *Variable must be 'huge'*

A variable or an array that requires more than 64K of storage in the 16-bit compiler must be declared as ***huge.***

E1158 *Too many parm sets*

Too many parameter register sets have been specified in the pragma.

E1159 *I/O error reading '%s': %s*

An I/O error has been detected by the compiler while reading the source file. The system dependent reason is also displayed in the message.

E1160 *Attempt to access far memory with all segment registers disabled in '%s'*

The compiler does not have any segment registers available to access the desired far memory location.

E1161 *No identifier provided for '-D' option*

The command line option "-D" must be followed by the name of the macro to be defined.

E1162 *Invalid register pegged to a segment in '%s'*

The register specified in a #pragma data_seg, or a ***__segname*** expression must be a valid segment register.

E1163 *Invalid octal constant*

An octal constant cannot contain the digits 8 or 9.

E1164 *Invalid hexadecimal constant*

The token sequence "0x" must be followed by a hexadecimal character (0-9, a-f, or A-F).

E1165 *Unexpected ')'. Probable cause: missing '('*

A closing parenthesis was found in an expression without a corresponding opening parenthesis.

E1166 *Symbol '%s' is unreachable from #pragma*

The in-line assembler found a jump instruction to a label that is too far away.

E1167 *Division or remainder by zero in a constant expression*

The compiler found a constant expression containing a division or remainder by zero.

E1168 *Cannot end string literal with backslash*

The argument to a macro that uses the stringize operator '#' on that argument must not end in a backslash character.

Example:

```
#define str(x) #x
str(@#\)
```

E1169 *Invalid __declspec declaration*

The only valid __declspec declarations are "__declspec(thread)", "__declspec(dllexport)", and "__declspec(dllimport)".

E1170 *Too many storage class specifiers*

You can only specify one storage class specifier in a declaration.

E1171 *Expecting '%s' but found end of file*

A syntax error has been detected. The compiler is still expecting more input when it reached the end of the source program.

E1172 *Expecting struct/union tag but found '%s'*

The compiler expected to find an identifier following the ***struct*** or ***union*** keyword.

E1173 *Operand of __builtin_isfloat() must be a type*

The __builtin_isfloat() function is used by the ***va_arg*** macro to determine if a type is a floating-point type.

E1174 *Invalid constant*

The token sequence does not represent a valid numeric constant.

E1175 *Too many initializers*

There are more initializers than objects to initialize. For example int X[2] = { 0, 1, 2 }; The variable "X" requires two initializers not three.

E1176 *Parameter %d, pointer type mismatch*

You have two pointers that either point to different objects, or the pointers are of different size, or they have different modifiers.

E1177 *Modifier repeated in declaration*

You have repeated the use of a modifier like "const" (an error) or "far" (a warning) in a declaration.

E1178 *Type qualifier mismatch*

You have two pointers that have different "const" or "volatile" qualifiers.

E1179 *Parameter %d, type qualifier mismatch*

You have two pointers that have different const or "volatile" qualifiers.

E1180 *Sign specifier mismatch*

You have two pointers that point to types that have different sign specifiers.

E1181 *Parameter %d, sign specifier mismatch*

You have two pointers that point to types that have different sign specifiers.

E1182 *Missing \\ for string literal*

You need a '\' to continue a string literal across a line.

E1183 *Expecting '%s' after '%s' but found '%s'*

A syntax error has been detected. The tokens displayed in the message should help you to determine the problem.

E1184 *Expecting '%s' after '%s' but found end of file*

A syntax error has been detected. The compiler is still expecting more input when it reached the end of the source program.

E1185 *Invalid register name '%s' in #pragma*

The register name is invalid/unknown.

E1186 *Storage class of 'for' statement declaration not register or auto*

The only storage class allowed for the optional declaration part of a ***for*** statement is ***auto*** or ***register***

E1187 *No type specified in declaration*

A declaration specifier must include a type specifier.

Example:

```
auto i;
```

E1188 *Symbol '%s' declared in 'for' statement must be object*

Any identifier declared in the optional declaration part of a ***for*** statement must denote an object. Functions, structures, or enumerations may not be declared in this context.

Example:

```
for( int i = 0, j( void ); i < 5; ++i ) {
    ...
}
```

E1189 *Unexpected declaration*

Within a function body, in C99 mode a declaration is only allowed in a compound statement and in the opening clause of a ***for*** loop. Declarations are not allowed after ***if, while,*** or ***switch*** statement, etc.

Example:

```
void foo( int a )
{
    if( a > 0 )
        int j = 3;
}
```

In C89 mode, declarations within a function body are only allowed at the beginning of a compound statement.

Example:

```
void foo( int a )
{
    ++a;
    int j = 3;
}
```

E1190 *'else' without 'if'*

The ***else*** must follow ***if***

E1191 *expression contains consecutive operand(s)*

More than one operand found in a row.

E1192 *'%s' unexpected in constant expression*

'%s' not allowed in constant expression.

E1193 *floating-point constant too small to represent*

The Open Watcom C compiler cannot represent the floating-point constant because the magnitude of the negative exponent is too large.

Example:

```
float f = 1.2e-78965;
```

E1194 *floating-point constant too large to represent*

The Open Watcom C compiler cannot represent the floating-point constant because the magnitude of the positive exponent is too large.

Example:

```
float f = 1.2e78965;
```

E1195 *unmatched left parenthesis '('*

The expression contains a left parenthesis "(" without a matching right parenthesis.

E1196 *expression contains extra operand(s)*

The expression contains operand(s) without an operator.

E1197 *unmatched right parenthesis ')'*

The expression contains a right parenthesis ")" without a matching left parenthesis.

E1198 *no expression between parentheses '( )'*

There is a matching set of parenthesis "()" which do not contain an expression.

E1199 *expecting ':' operator in conditional expression*

A conditional expression exists without the ':' operator.

E1200 *expecting '?' operator in conditional expression*

A conditional expression exists without the '?' operator.

E1201 *expecting first operand in conditional expression*

A conditional expression exists without the first operand.

E1202 *expecting second operand in conditional expression*

A conditional expression exists without the second operand.

E1203 *expecting third operand in conditional expression*

A conditional expression exists without the third operand.

E1204 *binary operator '%s' missing right operand*

There is no expression to the right of the indicated binary operator.

E1205 *binary operator '%s' missing left operand*

There is no expression to the left of the indicated binary operator.

E1206 *expecting operand after unary operator '%s'*

A unary operator without being followed by an operand.

E1207 *%s*

This is a internal error message generated by compiler.

1.7 Informational Messages

I2000 *Not enough memory to fully optimize procedure '%s'*

The compiler did not have enough memory to fully optimize the specified procedure. The code generated will still be correct and execute properly. This message is purely informational.

I2001 *Not enough memory to maintain full peephole*

Certain optimizations benefit from being able to store the entire module in memory during optimization. All functions will be individually optimized but the optimizer will not be able to share code between functions if this message appears. The code generated will still be correct and execute properly. This message is purely informational. It is only printed if the warning level is greater than or equal to 4.

The main reason for this message is for those people who are concerned about reproducing the exact same object code when the same source file is compiled on a different machine. You may not be able to reproduce the exact same object code from one compile to the next unless the available memory is exactly the same.

I2002 *'%s' defined in: %s(%u)*

This informational message indicates where the symbol in question was defined. The message is displayed following an error or warning diagnostic for the symbol in question.

Example:

```
static int a = 9;
int b = 89;
```

The variable 'a' is not referenced in the preceding example and so will cause a warning to be generated. Following the warning, the informational message indicates the line at which 'a' was declared.

I2003 *source conversion type is '%s'*

This informational message indicates the type of the source operand, for the preceding conversion diagnostic.

I2004 *target conversion type is '%s'*

This informational message indicates the target type of the conversion, for the preceding conversion diagnostic.

I2005 *Including file '%s'*

This informational message indicates that the specified file was opened as a result of `#include` directive processing.

I2006 *operator '%s'*

This informational message indicates the operator, for the preceding diagnostic.

I2007 *first operand type is '%s'*

This informational message indicates the type of the first operand, for the preceding diagnostic.

I2008 *second operand type is '%s'*

This informational message indicates the type of the second operand, for the preceding diagnostic.

1.8 Pre-compiled Header Messages

H3000 *Error reading PCH file*

The pre-compiled header file does not follow the correct format.

H3001 *PCH file header is out of date*

The pre-compiled header file is out of date with the compiler. The current version of the compiler is expecting a different format.

H3002 *Compile options differ with PCH file*

The command line options are not the same as used when making the pre-compiled header file. This can effect the values of the pre-compiled information.

H3003 *Current working directory differs with PCH file*

The pre-compiled header file was compiled in a different directory.

H3004 *Include file '%s' has been modified since PCH file was made*

The include files have been modified since the pre-compiled header file was made.

H3005 *PCH file was made from a different include file*

The pre-compiled header file was made using a different include file.

H3006 *Include path differs with PCH file*

The include paths have changed.

H3007 *Preprocessor macro definition differs with PCH file*

The definition of a preprocessor macro has changed.

H3008 *PCH cannot have data or code definitions.*

The include files used to build the pre-compiled header contain function or data definitions. This is not currently supported.

1.9 Miscellaneous Messages and Phrases

M4000 *Code size*

String used in message construction.

M4001 *Error!*

String used in message construction.

M4002 *Warning!*

String used in message construction.

M4003 *Note!*

String used in message construction.

M4004 *Parameter %d:*

String used in message construction.