

## A P P E N D I X   A

# Error Messages

The error messages generated by MASM components fall into three categories:

- ⚠ Fatal errors. These indicate a severe problem that prevents the utility from completing its normal process.
- ⚠ Nonfatal errors. The utility may complete its process. If it does, its result is not likely to be the one you want.
- ⚠ Warnings. These messages indicate conditions that may prevent you from getting the results you want.

All error messages take the form:

**Utility: Filename (Line) : [Error type] (Code): Message text**

*Utility* is the program that sent the error message.

*Filename* is the file that contains the error-generating condition.

*Line* is the approximate line where the error condition exists.

*Error type* is Fatal Error, Error, or Warning.

*Code* is the unique 5- or 6-digit error code.

*Message text* is a short and general description of the error condition.

## Error Message Lists

Messages for each utility are listed below in numerical order, with a brief explanation of each error. The following two tables list the messages by utility and error code, respectively.

**Table A.1 Error Codes Listed by Utility**

Utility Name	Error Type	Code	Page
BSCMAKE	Fatal	BK1500 to BK1515	688
	Warnings	BK4500 to BK4503	691
C/C++ Expression Evaluators	All	CAN0000 to CAN0063; CXX0000, CXX0064	692
CodeView	Nonfatal	CV0000 to CV5014	700
CVPACK	Fatal	CK1000 to CK1021	716
	Warnings	CK4000 to CK4003	720
EXEHDR	Fatal	U1100 to U1140	721
Math Coprocessor	All	M6101 to M6205	722
H2INC	Fatal	HI1003 to HI1801	724
	Nonfatal	HI2000 to HI2555	727
	Warnings	HI4000 to HI4820	745
HELPMAKE	Fatal	H1000 to H1990	761
	Nonfatal	H2000 to H2003	766
	Warnings	H4000 to H4003	766
IMPLIB	Fatal	IM1600 to IM1608	767
	Nonfatal	IM2601 to IM2603	768
	Warnings	IM4600 and IM4601	768
LIB	Fatal	U1150 to U1203	769
	Nonfatal	U2152 to U2159	772
	Warnings	U4150 to U4158	773
LINK	Fatal	L1001 to L1129	775
	Nonfatal	L2000 to L2064	786
	Warnings	L4000 to L4086	791
ML	Fatal	A1000 to A1901	798
	Nonfatal	A2000 to A2901	802
	Warnings	A4000 to A6005	825
NMAKE	Fatal	U1000 to U1099; U1450 to U1455	828
	Nonfatal	U2001	838
	Warnings	U4001 to U4009	838
PWB	All	PWB3089 TO PWB3912; PWB12078 TO PWB12086	840
SBRPACK	All	SB1000 to SB1006	842

**Table A.2 Error Codes Listed by Error Code Range**

Code	Utility Name	Error Type	Page
A1000 to A1901	ML	Fatal	798
A2000 to A2901	ML	Nonfatal	802
A4000 to A6005	ML	Warnings	825
BK1500 to BK1515	BSCMAKE	Fatal	688
BK4500 to BK4503	BSCMAKE	Warnings	691
CAN0000 to CAN0063; CXX0000, CXX0064	C/C++ Expression Evaluators	All	692
CK1000 to CK1021	CVPACK	Fatal	716
CK4000 to CK4003	CVPACK	Warnings	720
CV0000 to CV5014	CodeView	Nonfatal	700
H1000 to H1990	HELPMAKE	Fatal	761
H2000 to H2003	HELPMAKE	Nonfatal	766
H4000 to H4003	HELPMAKE	Warnings	766
HI1003 to HI1801	H2INC	Fatal	724
HI2000 to HI2555	H2INC	Nonfatal	727
HI4000 to HI4820	H2INC	Warnings	745
IM1600 to IM1608	IMPLIB	Fatal	767
IM2600 to IM2603	IMPLIB	Nonfatal	768
IM4600 and IM4601	IMPLIB	Warnings	768
L1001 to L1129	LINK	Fatal	775
L2000 to L2064	LINK	Nonfatal	786
L4000 to L4086	LINK	Warnings	791
M6101 to M6205	Math Coprocessor	All	722
PWB3089 to PWB3912; PWB12078 to PWB12086	PWB	All	840
SB1000 to SB1006	SBRPACK	All	842
U1000 to U1099	NMAKE	Fatal	828
U1100 to U1140	EXEHDR	Fatal	721
U1150 to U1203	LIB	Fatal	769
U1450 to U1455	NMAKE	Fatal	828
U2001	NMAKE	Nonfatal	838
U2152 to U2159	LIB	Nonfatal	772
U4001 to U4009	NMAKE	Warnings	838
U4150 to U4158	LIB	Warnings	773

## BSCMAKE Error Messages

Microsoft Browser Database Maintenance Utility (BSCMAKE) generates the following error messages:

- ⚠ Fatal errors (BK1xxx) cause BSCMAKE to stop execution.
- ⚠ Warnings (BK4xxx) indicate possible problems in the database-building process.

## BSCMAKE Fatal Error Messages

### **BK1500 UNKNOWN ERROR**

#### **Contact Microsoft Product Support Services**

BSCMAKE detected an unknown error condition.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

### **BK1501 unknown character *character* in option *option***

BSCMAKE did not recognize the given character specified for the given option.

### **BK1502 incomplete specification for option *option***

The given option did not contain the correct syntax.

### **BK1503 cannot write to file *filename***

BSCMAKE could not write to the given file.

One of the following may have occurred:

- ⚠ The disk was full.
- ⚠ A hardware error occurred.

### **BK1504 cannot position in file *filename***

BSCMAKE could not move to a location in the given file.

One of the following may have occurred:

- ⚠ The disk was full.
- ⚠ A hardware error occurred.
- ⚠ The file was truncated. Truncation can occur if the compiler runs out of disk space or is interrupted when it is creating the .SBR file.

**BK1505 cannot read from file *filename***

BSCMAKE could not read from the given file.

One of the following may have occurred:

- 🔒 The file was corrupt.
- 🔒 The file was truncated. Truncation can occur if the compiler runs out of disk space or is interrupted when it is creating the .SBR file.

**BK1506 cannot open file *filename***

BSCMAKE could not open the given file.

One of the following may have occurred:

- 🔒 No more file handles were available. Increase the number of file handles by changing the FILES setting in CONFIG.SYS to allow a larger number of open files. FILES=50 is the recommended setting.
- 🔒 The file was locked by another process.
- 🔒 The disk was full.
- 🔒 A hardware error occurred.
- 🔒 The specified output file had the same name as an existing subdirectory.

**BK1507 cannot open temporary file *filename***

BSCMAKE could not open one of its temporary files.

One of the following may have occurred:

- 🔒 No more file handles were available. Increase the number of file handles by changing the FILES setting in CONFIG.SYS to allow a larger number of open files. FILES=50 is the recommended setting.
- 🔒 The TMP environment variable was not set to a valid drive and directory.
- 🔒 The disk was full.

**BK1508 cannot delete temporary file *filename***

BSCMAKE could not delete one of its temporary files.

One of the following may have occurred:

- 🔒 Another process had the file open.
- 🔒 A hardware error occurred.

**BK1509 out of heap space**

BSCMAKE ran out of memory.

One of the following may be a solution:

- 🔧 Reduce the memory that BSCMAKE will require by using one or more options. Use /Ei or /Es to eliminate some input files. Use /Em to eliminate macro bodies.
- 🔧 Run BSCMAKE (or PWB if you are building a database in PWB) in an MS-DOS session within Windows to use virtual memory provided under the Windows operating system.
- 🔧 Free some memory by removing terminate-and-stay-resident (TSR) software.
- 🔧 Reconfigure the EMM driver.
- 🔧 Change CONFIG.SYS to specify fewer buffers (the BUFFERS command) and fewer drives (the LASTDRIVE command).
- 🔧 Run BSCMAKEV.EXE instead of BSCMAKE.EXE.

**BK1510 corrupt .SBR file *filename***

The given .SBR file is corrupt or does not have the expected format.

Recompile to regenerate the .SBR file.

**BK1511 invalid response file specification**

BSCMAKE did not understand the command-line specification for the response file. The specification was probably wrong or incomplete.

For example, the following specification causes this error:

```
bscmake @
```

**BK1512 database capacity exceeded**

BSCMAKE could not build a database because the number of definitions, references, modules, or other information exceeded the limit for a database.

One of the following may be a solution:




- 🔧 Exclude some information using the /Em, /Es, or /Ei option.
- 🔧 Omit the /Iu option if it was used.
- 🔧 Divide the list of .SBR files and build multiple databases.

**BK1513      nonincremental update requires all .SBR files**

An attempt was made to build a new database, but one or more of the specified .SBR files was truncated. This message is always preceded by warning BK4502, which will give the name of the .SBR file that caused the error.

BSCMAKE can process a truncated, or zero-length, .SBR file only when a database already exists and is being incrementally updated.



One of the following may be a cause:

-  The database file was missing.
-  The wrong database name was specified.
-  The database was corrupted, and a full build was required.

**BK1514      all .SBR files truncated and not in database**

None of the .SBR files specified for an update was a part of the original database. This message is always preceded by warning BK4502, which will give the name of the .SBR file that caused the error.

One of the following may be a cause:

-  The wrong database name was specified.
-  The database was corrupted, and a full build was required.

**BK1515      *bscfile* : incompatible version; cannot incrementally update**

The given database (.BSC file) was not created with this version of BSCMAKE.

A database can be incrementally built only by the same version of BSCMAKE as the one used to fully build the database.

## BSCMAKE Warning Messages

**BK4500      UNKNOWN WARNING****Contact Microsoft Product Support Services**

An unknown error condition was detected by BSCMAKE.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**BK4501      ignoring unknown option *option***




BSCMAKE did not recognize the given option and ignored it.

If the given option is /r, it must be specified first on the BSCMAKE command line.

**BK4502    truncated .SBR file *filename* not in database**

The given zero-length .SBR file, specified during a database update, was not originally part of the database.

If a zero-length file that is not part of the original build of the database is specified during a rebuild of that database, BSCMAKE issues this warning. One of the following may be a cause:

-  The wrong database name was specified.
-  The database was deleted (error BK1513 will result).
-  The database file was corrupted, requiring a full build.

**BK4503    minor error in .SBR file *filename* ignored**

The given .SBR file contained an error that did not halt the build. However, the resulting .BSC file may not be correct.

Recompile to regenerate the .SBR file.

## CodeView C/C++ Expression Evaluator Errors

**CAN0000    no error condition**

No error has occurred, and this message should not appear.

You can continue debugging normally.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**CAN0001    exception executing user function**

The code being executed caused a general protection fault.

**CAN0002    error accessing user memory**

The expression attempts to reference memory that is not allocated to the program being debugged.

**CAN0003    internal error in expression evaluator**

The CodeView expression evaluator encountered an internal error.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**CAN0004    syntax error**

The syntax of the expression is incorrect.

Retype the expression with the correct syntax.



**CAN0005 operator not supported**

An unsupported C operator was specified in an expression.

You can usually write an equivalent expression using the supported C operators.

**CAN0006 missing left parenthesis**

Unbalanced parentheses were found in the expression.

Retype the expression with balanced parentheses.

**CAN0007 missing right parenthesis**

Unbalanced parentheses were found in the expression.

Retype the expression with balanced parentheses.

**CAN0008 missing \at end of string**

The double quotation mark (") expected at the end of the string literal was missing.

Retype the expression, enclosing the string literal in double quotation marks.

**CAN0009 missing ' after character constant**

The single quotation mark (') expected at the end of the character constant was missing.

Retype the expression, enclosing the character constant in single quotation marks.

**CAN0010 missing left bracket**

The expression contains unbalanced square brackets.

Retype the expression with balanced square brackets.

**CAN0011 missing right bracket**

The expression contains unbalanced square brackets.

Retype the expression with balanced square brackets.

**CAN0012 missing left curly brace**

The expression contains an unbalanced curly brace.

Retype the expression with balanced curly braces.

**CAN0013 missing operator**

An operator was expected in the expression but was not found.

Check the syntax of the expression.

**CAN0014 missing operand**

An operator was specified without a required operand.

Check the syntax of the expression.

**CAN0015 expression too complex (stack overflow)**

The expression entered was too complex or nested too deeply for the amount of storage available to the C expression evaluator.

Overflow usually occurs because of too many pending calculations.

Rearrange the expression so that each component of the expression can be evaluated as it is encountered, rather than having to wait for other parts of the expression to be calculated.

Break the expression into multiple commands.

**CAN0016 constant too big**

The CodeView C expression evaluator cannot accept an unsigned integer constant larger than 4,294,967,295 (FFFFFFFF hexadecimal), or a floating-point constant whose magnitude is larger than approximately 1.8E+308.

**CAN0017 symbol not found**

A symbol specified in an expression could not be found.

One possible cause of this error is a case mismatch in the symbol name. Since C and C++ are case-sensitive languages, a symbol name must be given in the exact case in which it is defined in the source.

**CAN0018 bad register name**

A specified register does not exist or cannot be displayed.

CodeView can display the following registers: AX, SP, DS, IP, BX, BP, ES, FL, CX, SI, SS, DX, DI, CS.

When running with MS-DOS on an 80386 machine, the 386 option can be selected to display the following registers: EAX, ESP, DS, GS, EBX, EBP, ES, SS, ECX, ESI, FS, EIP, EDX, EDI, CS, EFL.

**CAN0019 bad type cast**

The CodeView C expression evaluator cannot perform the type cast as written.

One of the following may have occurred:

- 🔍 The specified type is unknown.
- 🔍 There were too many levels of pointer types.

For example, the type cast:

```
(char far * far *)h_message
```

cannot be evaluated by the CodeView C expression evaluator.

**CAN0020 operand types bad for this operation**

An operator was applied to an expression with an invalid type for that operator.

For example, it is not valid to take the address of a register, or subscript an array with a floating-point expression.

**CAN0021 struct or union used as scalar**

A structure or union was used in an expression, but no element was specified.

When manipulating a structure or union variable, the name of the variable may appear by itself, without a field qualifier. If a structure or union is used in an expression, it must be qualified with the specific element desired.

Specify the element whose value is to be used in the expression.

**CAN0022 function call before `_main`**

The CodeView C expression evaluator cannot evaluate a function before CodeView has entered the function `_main`. The program is not properly initialized until `_main` has been called.

Execute `g main; p` to enable function calls in expressions.

**CAN0023 bad radix**

The radix specified is not recognized by the CodeView C expression evaluator. Only decimal, hexadecimal, and octal radices are valid.

**CAN0024 operation needs l-value**

An expression that does not evaluate to an l-value was specified for an operation that requires an l-value.

An l-value (so called because it appears on the left side of an assignment statement) is an expression that refers to a memory location.

For example, `buffer[count]` is a valid l-value because it points to a specific memory location. The logical comparison `zed != 0` is not a valid l-value because it evaluates to TRUE or FALSE, not a memory address.

**CAN0025 operator needs struct/union**

An operator that takes an expression of struct or union type was applied to an expression that is not a struct or union.

Components of class, structure, or union variables must have a fully qualified name. Components cannot be entered without full specification.

**CAN0026 bad format string**

A format string was improperly specified.

Check the syntax of the expression.

**CAN0027 tp addr not l-value**

Check the syntax of the expression.

**CAN0028 not struct/union element**

An expression of the form **Struct. Member** or **pStruct->Member** was specified, but *member* is not an element of the structure.

The expression may not be parenthesized correctly.

**CAN0029 not struct pointer**

The member-selection operator (->) was applied to an expression that is not a pointer to a structure.

Check that the entire expression is parenthesized correctly, or type cast the address expression to the appropriate structure pointer type.

**CAN0030 expression not evaluable**

The expression could not be evaluated as written.

This error is frequently caused by dereferencing a pointer which is not valid.

Check that the syntax of the expression is correct, and that all symbols are specified in the exact case as they are defined in the program.

**CAN0031 expression not expandable**

The CodeView C expression evaluator encountered an internal error.

You may be able to write an equivalent expression that can be evaluated correctly.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the "Microsoft Support Services" section of the introduction to this book.

**CAN0032 divide by 0**

The expression contains a divisor of zero, which is illegal. This divisor may be the literal number zero, or it may be an expression that evaluates to zero.

**CAN0033 error in OMF type information**

The executable file did not have a valid OMF (Object Module Format) for debugging by CodeView.

One of the following may have occurred:

- 🔍 The executable file was not created with the linker released with this version of CodeView. Relink the object code using the current version of LINK.EXE.
- 🔍 The executable file was not created with the high-level language released with this version of CodeView. Recompile the program with the current version of the compiler.
- 🔍 The .EXE file may have been corrupted. Recompile and relink the program.

**CAN0034 types incompatible with operator**

The operand types specified are not legal for the operation.

For example, a pointer cannot be multiplied by any value.

You may need to type cast the operands to a type compatible with the operator.


**CAN0035 overlay not resident**

An attempt was made to access an overlay that is not currently resident in RAM.

Execute the program until the overlay is loaded.

**CAN0036 bad context {...} specification**

This message can be generated by any of several errors in the use of the context-resolution operator (`{}`).


 The syntax of the context-resolution operator (`{}`) was given incorrectly.

The syntax of the context operator is:

`{ [function], [module], [dll] } expression`

This specifies the context of *expression*. The context operator has the same precedence and usage as a type-cast.

Trailing commas can be omitted. If any of `[function]`, `[module]`, or `[dll]` contain a literal comma, you must enclose the entire name in parentheses.

 The function name was spelled incorrectly, or does not exist in the specified module or dynamic-link library.

Since C is a case-sensitive language, *function* must be given in the exact case as it is defined in the source. The C expression evaluator ignores the CodeView case-sensitivity state set with the OC command or the Case Sensitive command in the Options menu.

 The module or DLL could not be found.

Check the full path name of the specified module or DLL.

**CAN0037 out of memory**

The CodeView C expression evaluator ran out of memory evaluating the expression.

**CAN0038 function argument count and/or type mismatch**

The function call as specified does not match the prototype for the function.

Retype the call with the correct number of arguments. Type cast each argument to match the prototype, as necessary.

**CAN0039 symbol is ambiguous**

The CodeView C expression evaluator cannot determine which instance of a symbol to use in an expression. The symbol occurs more than once in the inheritance tree.

You must use the scope resolution operator (::) to explicitly specify the instance to use in the expression.

**CAN0040 function requires implicit conversion**

Implicit conversions involving constructor calls are not supported by the CodeView C expression evaluator.

**CAN0041 class element must be static member or member function**

A nonstatic member of a class (or structure or union) was used without specifying which instantiation of the class to use.

Only static data members or member functions can be used without specifying an instantiation.

**CAN0042 bad line number**

This error should never occur.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**CAN0043 this pointer used outside member function**

This pointer can only be used for nonstatic member functions.

**CAN0044 use of `_based(void)` pointer requires `>` operator**

A pointer based on **void** cannot be used directly. You must form a complete pointer using the `>` operator.

**CAN0045 not a function**

An argument list was supplied for a symbol in the program that is not the name of a function.

For example, this error is generated for the expression

```
queue( alpha, beta
```

when **queue** is not a function.

**CAN0046 argument list required for member function**

An expression called a member function but did not specify any actual parameters.

**CAN0047 argument list does not match a function**

An expression called a function with an actual parameter list that did not match the formal parameter list of any function with the same name defined in the program.

Overloaded functions can be called only if there is an exact parameter match, or a match that does not require the construction of an object.

- CAN0048 calling sequence not supported**  
A function specified in the expression uses a calling sequence not supported by the CodeView C expression evaluator. You cannot call this function in a CodeView expression.
- CAN0049 obsolete OMF - please relink program**  
The program used an old OMF (Object Module Format).  
The program must be linked with LINK version 5.30 or later, and packed with CVPACK version 4.0 or later.
- CAN0050 left side of :: must be class/struct/union**  
The symbol on the left side of the scope-resolution operator (::) was not a class, structure or union.
- CAN0051 more than one overloaded symbol specified in breakpoint**  
CodeView could not determine which of more than one overloaded symbol to use as a breakpoint.
- CAN0052 member function not present**  
A member function was specified as a breakpoint but could not be found. This error can be caused by setting a breakpoint at a function that has been inlined.  
Recompile the file with inlining forced off (/Ob0) to set a breakpoint in this function.  
An expression called a function that was not defined.
- CAN0053 nonfunction symbol match while binding breakpoints**  
A symbol used as a breakpoint was not a function. This error can be caused by specifying a data member as a breakpoint.
- CAN0054 register in breakpoint expression illegal**  
A register cannot be used in a breakpoint expression.
- CAN0055 ambiguous symbol in context operator**  
A symbol in the context operator ({ }) referred to more than one symbol in the program.  
The scope resolution operator (::) may be able to resolve the ambiguity.
- CAN0056 error in line number**  
An invalid line number was specified.
- CAN0057 no code at line number**  
No code was generated for the specified line number. It cannot be used as a breakpoint.
- CAN0058 overloaded operator not found**  
A class type was specified as the left operand in an expression, but an overloaded operator was not defined for the class.

**CAN0059 left operand is class not a function name**

The left operand of a function call was a class name and could not be resolved to a function call. This error can be caused by omitting the name of a member function in an expression.

**CAN0060 register is not available**

An expression specified a register that cannot be used.

This error can be caused by trying to access a register that does not exist on the machine running CodeView, for example, accessing 80386-specific registers on an 8088-based machine.

**CAN0061 function nesting depth exceeded**

The expression contains a function nesting depth greater than the limit.

The expression should be modified to reduce the nesting depth.

**CAN0062 constructor calls not supported**

An expression made a call to a constructor.

Expressions cannot make explicit calls to constructors or make conversions that require a call to a constructor.

**CXX0063 overloaded operator -> not supported**

The expression used an overloaded class member access operator (->).

**CXX0064 can't set breakpoint on bound virtual member function**

A breakpoint was set on a virtual member function through a pointer to an object, such as:

```
pClass->vfunc( int );
```

A breakpoint can be set on a virtual function by entering the class, such as:

```
Class::vfunc( int );
```

## CodeView Error Messages

**CV0000 no error; NOERROR; No Error Condition**

You should not normally receive this error message since CV0000 indicates that no error occurred.





**CV0002 no such file or directory**

The specified file does not exist or a path does not specify an existing directory.

Check the file or directory name in the most recent command.

One of the following may have occurred:

-  The View Source (VS) command or the Open Source command from the File menu was used to view a nonexistent file.
-  An attempt was made to print to a nonexistent file or directory.

**CV0003 program terminated: restart to continue**

CodeView has detected a termination request by the program being debugged.

The program cannot be executed because it has terminated and has not been restarted. Program memory remains allocated and may still be examined at this point.

To run the program again, reload it using the Restart command.

**CV0005 I/O error**

An attempt was made to access an address that is not accessible to the program being debugged.

Check the previous command for numeric constants used as addresses and for pointers used for indirection.

**CV0007 number of arguments exceeds DOS limit of 128**

CodeView is not able to restart the program that is being debugged because the number of arguments to the executable program exceeds the limit of 128.

**CV0008 executable file format error**

The system is not able to load the program to be debugged. The file is not an executable file, or it has an invalid format for this operating system.

Try to run the program outside of CodeView to see if it is a valid executable file.

This error can be caused if there is not enough memory available to run the program.

Try making more memory available to the program.

**CV0012 out of memory**

CodeView was unable to allocate or reallocate the memory that it required because not enough memory was available.

Possible solutions include the following:

- 🔧 Recompile without symbolic information in some of the modules. CodeView requires memory to hold information about the program being debugged. Compile some modules with the /Zd option instead of /Zi, or don't use either option.
- 🔧 Remove other programs or drivers running in the system that could be consuming significant amounts of memory.
- 🔧 Decrease the settings in CONFIG.SYS for FILES and BUFFERS.

**CV0013 access denied**

A specified file's permission setting does not allow the required access.

One of the following may have occurred:

- 🔧 An attempt was made to write to a read-only file.
- 🔧 A locking or sharing violation occurred.
- 🔧 An attempt was made to open a directory instead of a file.

**CV0014 invalid address**

The command expected an address but was given an argument that could not be interpreted as a valid address.

A name or constant may have been specified without the period (.) that indicates a filename or line number.

**CV0018 no such file or directory**

The specified file does not exist or a path does not specify an existing directory.

Check the file or directory name in the most recent command.

One of the following may have occurred:

- 🔧 The View Source (VS) command or the Open Source command from the File menu was used to view a nonexistent file.
- 🔧 An attempt was made to print to a nonexistent file or directory.

**CV0022 invalid argument**

An invalid value was given as an argument.

**CV0024      too many open files**

CodeView could not open a file it needed because a file handle was not available.

Increase the number of file handles by changing the FILES setting in CONFIG.SYS to allow a larger number of open files. FILES=50 is the recommended setting.

The program being debugged may have so many files open that all available handles are exhausted. Check that the program has not left files open unnecessarily. The first four handles are reserved by the operating system.

Additional files can be made available by closing source windows. If more files are needed, set helpbuffers=0 in the [pwb] section of TOOLS.INI. As a result, online help cannot be used but several file handles will be made available.

**CV0028      no space left on device**

The disk does not have any space available for writing.

One of the following may have occurred:

- 🔍 CodeView could not find room for writing a temporary file.
- 🔍 An attempt was made to write to a disk that was full.

**CV0101      no CodeView information for *filename***

The executable file or dynamic-link library (DLL) did not contain the symbols needed by CodeView.

Be sure to compile the program or DLL using the /Zi option. If linking in a separate step, be sure to use the /CO option. Use the most current version of LINK.

**CV0102      unpacked CodeView information in *filename*: use CVPACK**

For this version of CodeView, you must process all executable files using CVPACK, which compresses the debugging information in the file.

Pass the file through CVPACK.EXE before starting CodeView.

**CV0103      relink *filename* with the current linker**

This version of CodeView expects the executable file to be in the format produced by the current version of the linker.

Make sure PWB, NMAKE, or the compiler is not running an older version of the linker.

**CV0104      CodeView information for *filename* is newer than this version of CodeView**

The executable file was compiled or linked with a version of a Microsoft compiler that is newer than the version of CodeView you are using.

Try one of the following:

- 🔧 Reinstall CodeView that came with the new compiler.
- 🔧 Remove older versions of CodeView that may be present on your hard disk.
- 🔧 Recompile the program with an older version of a Microsoft compiler.

**CV1001      invalid breakpoint command**

CodeView could not interpret the breakpoint command.

The command probably used an invalid symbol or the incorrect command format.

**CV1003      extra input ignored**

The first part of the command line was interpreted correctly.

The remainder of the line could not be interpreted or was unnecessary.

**CV1004      invalid register**

The Register (**R**) command named a register that does not exist or cannot be displayed. CodeView can access the following registers: AX, SP, DS, IP, BX, BP, ES, FL, CX, SI, SS, DX, DI, CS.

When running with MS-DOS or the Windows operating system on an 80386 or an 80486 machine, the 80386 registers option can be selected to access the following registers: EAX, ESP, DS, GS, EBX, EBP, ES, SS, ECX, ESI, FS, EIP, EDX, EDI, CS, EFL.

When debugging p-code, CodeView can also access the following registers: TL, TH, PQ.

**CV1006      breakpoint number or '\*' expected**

A breakpoint was specified without a number or asterisk.

A Breakpoint Clear (BC), Breakpoint Disable (BD), or Breakpoint Enable (BE) command requires one or more numbers to specify the breakpoints or an asterisk to specify all breakpoints.

For example, the following command causes this error:

```
bc r i k a
```

**CV1007      unable to open file**

The specified file cannot be opened.

One of the following may have occurred:

- 🔒 The file may not exist in the specified directory.
- 🔒 The filename was misspelled.
- 🔒 The file's attributes are set so that it cannot be opened.
- 🔒 A locking or sharing violation occurred.

**CV1011      no previous regular expression**

The Repeat Last Find command was executed, but a regular expression (search string) was not previously specified.

**CV1012      regular expression too long**

The regular expression was too long or complex.

Use a simpler or more general regular expression.

**CV1016      match not found**

A string could not be found that matched the search pattern.

**CV1017      syntax error**

The command contained a syntax error.

This error is probably caused by an invalid command or expression.

**CV1018      unknown symbol**

The symbolic name specified could not be found.

One of the following may have occurred:

- 🔒 The specified name was misspelled.
- 🔒 The wrong case was used when case sensitivity was turned on. Case sensitivity is toggled by the Case Sensitivity command from the Options menu or is set by the Option (O) Command-window command.
- 🔒 The module containing the specified symbol may not have been compiled with the /Zi option to include symbolic information.
- 🔒 A search was made for an undefined label or function.

**CV1021      unknown format specifier; specify one of A,B,I,IU,IX,L,LU,LX,R,RL,RT**

An unknown format specifier was given to a View Memory (VM), Memory Dump (MD), or Memory Enter (ME) command.

The valid format specifiers are:

Specifier	Display Format
A	ASCII
B	byte
I	16-bit signed decimal integer
IU	16-bit unsigned decimal integer
IX	16-bit hexadecimal integer
L	32-bit signed decimal integer
LU	32-bit unsigned decimal integer
LX	32-bit hexadecimal integer
R	32-bit single precision floating point
RL	64-bit double precision floating point
RT	80-bit 10-byte real (long double)

This error is probably due to a mistyped command.

**CV1022      invalid flag**

An attempt was made to examine or change a flag, but the flag name was not valid.

Any flags preceding the invalid name were changed to the values specified. Any flags after the invalid name were not changed.

Use the flag mnemonics displayed after entering the R FL command.

**CV1023      no code at this line number**

A line number was specified but code was not generated for that line. This error can be caused by a blank line, comment line, line with program declarations, or line moved or removed by compiler optimization.

To set a breakpoint at a line deleted by the optimizer, recompile the program with the /Od option to turn off optimization.

Note that in a multiline statement the code is associated only with one line of the statement.

This error can be caused by debugging a program whose source has been modified after it was compiled. Recompile the file before running it through CodeView.

**CV1027      invalid radix: specify 8, 10, or 16**

The Radix (N) command takes three radices: 8 (octal), 10 (decimal), and 16 (hexadecimal). Other radices are not permitted. The new radix is always entered as a decimal number, regardless of the current radix.

**CV1031 no source lines at this address**

An attempt was made to view an address that does not have source code.

This error can be caused by debugging a program whose source has been modified after it was compiled. Recompile the file before debugging it with CodeView.

**CV1039 not a text file**

An attempt was made to load a file that is not a text file. The file may be binary data.

This error can also occur if the first line of a file includes characters that are in the range of ASCII 0 to 8, 14 to 31, or 127 (0x0 to 0x8, 0xE to 0x1F, or 0x7F).

The Source window can only be used to view text files.

**CV1040 video mode changed without /S option**



The program being debugged changed screen modes, and CodeView was not set for swapping. The program output is now damaged or unrecoverable.

To be able to view program output, exit CodeView and restart it with the Swap (/S) option.

**CV1041 file error**

CodeView could not write to the disk.

One of the following may have occurred:

-  There was not enough space on the disk.
-  The file was locked by another process.

**CV1042 library module not loaded**

The program being debugged uses load-on-demand dynamic-link libraries (DLLs). At least one of these libraries is needed but could not be found.

**CV1043 application output lost; screen exchange is off**

The program being debugged wrote to the display when the Flip (/F) or Swap (/S) option was turned off. The program output was lost.

When flipping is on, video page 1 is usually reserved for CodeView. Programs usually write to video page 0 by default. Programs that write to video page 1 must be debugged with swapping on.

Turn Flip or Swap on to be able to view program output.

**CV1046     invalid executable file: relink**

The executable file did not have a valid format.

One of the following may have occurred:

- The executable file was not created with the linker released with this version of CodeView. Relink the object code using the current version of LINK.EXE.
- The .EXE file may have been corrupted. Recompile and relink the program.

**CV1047     overlay not resident**

An attempt was made to access machine code from an overlay section of code that is not currently resident in memory.

Execute the program until the overlay is loaded.

**CV1048     floating-point support not loaded**

An attempt was made to access the math processor registers in a program that does not use floating-point arithmetic.

One of the following can cause this error:

- Math processor registers can only be accessed through the floating-point library code. If the program does not perform floating-point calculations, this error can occur because the floating-point library code will not be loaded and cannot be used to access math processor registers.
- If the program does not use floating-point instructions, this error can occur when you attempt to access the math processor before any floating-point instructions have been performed. The run-time library includes a floating-point instruction near the beginning so that the math processor registers are always accessible.
- If a floating-point instruction occurs in an assembly language routine before such an instruction occurs in the high-level language code that calls the routine, this error occurs.

**CV1050     expression not a memory address**

The expression does not evaluate to an address.

For example, **buffer[count]** is a valid address because it points to a specific memory location. The logical comparison **zed != 0** is not a valid address because it evaluates to TRUE or FALSE, not a memory address.

**CV1051     missing or corrupt emulator information**

Status information about the floating-point emulator is missing or corrupt.

The program probably wrote to this area of memory. Make sure the pointer points to its intended object.



- CV1053      **TOOLS.INI not found****  
The directory listed in the INIT environment variable did not contain a TOOLS.INI file.  
Check the INIT variable to be sure that it points to the correct directory.
- CV1054      **cannot read this version of CURRENT.STS****  
The state file (CURRENT.STS) has a version number that is not recognized by this version of CodeView.  
  
The old CURRENT.STS was ignored, and a new one will be created when CodeView exits.
- CV1056      **cannot understand entry in *filename*****  
At least one line in the given file (either the state file or the TOOLS.INI file) could not be interpreted.  
  
On startup, CodeView reads the state file (CURRENT.STS) and the TOOLS.INI file (if the latter is available).  
  
Examine the given file to find the problem.
- CV1057      **CURRENT.STS not found; creating****  
Since the state file (CURRENT.STS) could not be located at startup, CodeView created a state file.
- CV1058      **no source window open****  
A command was entered to manipulate the contents of a Source window, but a Source window was not open.
- CV1059      **no CodeView source information****  
CodeView symbol listing for the source file or module being debugged does not exist.  
  
Be sure the file was compiled with the /Zi option or the /Zd option. If linking in a separate step, be sure to use the /CO option.
- CV1060      **command not supported for current configuration****  
If you have specified the two monitors option (/2), you cannot specify the flip/swap option (/of- or /of+) from the CodeView Command Window.
- CV1061      **no second monitor connected to system****  
CodeView was invoked with the /2 option, but there was only one monitor for CodeView to use.
- CV1062      **invalid code-segment context change****  
An attempt was made to set the IP register to a line or address in a different segment.

**CV1063 cannot create CURRENT.STS**

CodeView could not find an existing state file (CURRENT.STS), and CodeView tried to create one but failed.

One of the following may have occurred:

- 🔍 There was not enough space either on the disk containing the program to be debugged or on the disk pointed to by the INIT environment variable.
- 🔍 There were not enough free file handles. Increase the number of file handles by changing the FILES setting in CONFIG.SYS to allow a larger number of open files.
- 🔍 The environment variable INIT pointed to a directory that does not exist.

**CV1064 window could not be opened**

CodeView tried to open a window, but failed to do so.

This error is probably caused by a lack of memory available to CodeView.

Exit CodeView and make more memory available, then restart CodeView.

**CV1065 cannot load expression evaluator *filename***

CodeView could not load the specified expression evaluator.

Make sure that *filename* is a valid expression evaluator DLL. If not, try reinstalling the CodeView DLLs from the distribution disks.

**CV1066 cannot load expression evaluator *filename*; limit is 10**

Up to 10 expression evaluators can be specified in the TOOLS.INI file.

Try removing expression evaluators you won't be using in your debugging session.

**CV1067 extension missing for Expression Evaluator: *filename* in TOOLS.INI**

The Eval entry in the TOOLS.INI file expected a list of filename extensions.

**CV1068 breakpoint specifier is out of range**

The breakpoint number specified was higher than the number of current breakpoints.

**CV1250      general expression-evaluator error**

An error occurred in a CodeView expression evaluator.

This error is probably caused by a lack of memory available to the expression evaluator. You can free memory by doing one or more of the following:

- 🔧 Close windows that are not needed. The Memory window should be closed if possible.
- 🔧 Delete breakpoints that are not needed.
- 🔧 Disable options that are not needed.

As a last resort, exit CodeView and start the debugging session again.

This error can also be caused by an expression that cannot be evaluated by the expression evaluator.

**CV1251      *message***

An error occurred within a CodeView expression evaluator.

No further explanation is available.

**CV1254      invalid address expression**

The expression entered does not evaluate to an address.

The expression must be in a form that can appear on the left side of an assignment and refer to a memory location.

For example, **buffer[ count ]** is a valid l-value because it points to a specific memory location. The logical comparison **zed != 0** is not a valid l-value because it evaluates to TRUE or FALSE, not a memory address.

**CV1255      no data members**

The class, structure, or union that was expanded did not have data members. A class must contain at least one data member to be expanded.

**CV2206      corrupt CodeView information in *filename*; discarding**

This error can be caused by using mismatched versions of development tools. Verify that the versions of all tools are current and synchronized.

Try recompiling the file with the /Zd switch (Prepare for Debugging option).

This option produces an object file containing only public symbols (global or external) and line numbers.

**CV2207      loaded symbols for *module***

CodeView automatically loaded the symbols for the given dynamic-link library (DLL). The DLL can now be debugged.

This message is for your information only and does not indicate an error.

- CV2209 cannot restart; current process is not the process being debugged**  
The debugging session was halted, and a different process was started.  
Return to the debugged program's process by setting a breakpoint in it and issue a Go command.
- CV2210 invalid tab setting; using 8**  
The value for tabs cannot be less than 0 or greater than 19. If you supply a value that is not in this range, the default tab value is 8.
- CV2211 cannot terminate; current process is not the process being debugged**  
The debugging session was halted, and a different process was current.  
Return to the debugged program's process by setting a breakpoint in it and issue a Go command.
- CV2401 missing argument for *option option***  
This error can be caused by splitting a response file line naming a program to be debugged and its command-line options. The program name and its command-line option must be on one line.
- CV2402 unknown option *option*; ignored**  
The specified option was not a valid option.  
Check that the option was typed correctly.
- CV2403 response files cannot be nested**  
A response file cannot refer to another response file.
- CV2404 cannot open response file: *filename***  
The specified response file could not be opened.  
Check that the name of the file is spelled correctly and that the response file is correct.
- CV2405 command line option, *option*, invalid for target operating system**  
The specified command line option was illegal in this context.
- CV2406 command-line is too big. arguments truncated**  
The command line argument in a response file was longer than the limit of 256 bytes.

**CV3608 out of memory**

CodeView needed additional memory, but insufficient memory was available.

Possible solutions include the following:

- 🔧 Remove some drivers or applications that have been loaded in high memory.
- 🔧 Recompile without symbolic information in some of the modules. CodeView requires memory to hold information about the program being debugged. Compile some modules with the /Zd option instead of /Zi, or don't use either option.
- 🔧 Remove other programs or drivers running in the system that could be consuming significant amounts of high memory.
- 🔧 Free some memory by removing terminate-and-stay-resident (TSR) software.
- 🔧 Remove unneeded watch expressions or breakpoints.

**CV3620 bad DLL format in *filename***

CodeView did not recognize the format of the specified CodeView dynamic-link library (DLL) file.

The DLL may be damaged or may be the wrong version.

This error is caused if the specified file is not a DLL.

**CV3621 cannot find DLL *filename***

CodeView could not find the specified dynamic-link library (DLL). This may be caused by a mistyped filename in the TOOLS.INI file.

**CV3622 cannot load DLL *filename***

CodeView was unable to load the specified dynamic-link library (DLL) file.

Reinstall the CodeView DLL from the distribution disks.

**CV3623 wrong DLL *filename***

CodeView expected one type of dynamic-link library (DLL) but read a different type. This error is probably caused by specifying an incorrect filename in the TOOLS.INI file. For example, you may have specified an execution model in the expression-evaluator entry.

**CV3624 cannot load execution model *filename* - limit is 1**

Too many execution models are specified in the TOOLS.INI file.

Only one execution model can be used at a time.

Remove those execution models you are not using in your debugging session.

- CV3625 no transport layer; exiting**  
CodeView needs a transport layer to make appropriate calls to the operating system in local debugging and to a remote computer in remote debugging.  
Check your TOOLS.INI file, and make sure there is a Transport entry in the [cv] or [cvw] CodeView section.
- CV3626 no execution model; exiting**  
CodeView needs an execution model in order to function.  
Check your TOOLS.INI file, and make sure there is a Native entry specified.
- CV3627 no nonnative execution models found**  
You must specify a nonnative execution model in order to debug a p-code program.  
Add the following line to your TOOLS.INI file:  
`model : nmd1pcd. dll`
- CV3628 too many transport layers: choose one**  
Only one transport layer can be selected at one time.
- CV3629 too many execution models: choose one**  
Only one execution model can be selected at a time.  
Additional execution models should be removed.
- CV3630 no symbol handler found; exiting**  
A symbol handler dynamic-link library (DLL) could not be found. The DLLs that CodeView uses must be in a location specified by the cvdllpath entry in the [cvw] or [cv] section of TOOLS.INI.
- CV3631 program being debugged contains p-code, but no *model*: specified in tools.ini**  
The entry `model =nmd1pcd. dll` must be specified in TOOLS.INI to debug a program that contains p-code.
- CV4000 assembler: not enough operands**  
Additional operands are required for this instruction.  
The instruction was rejected and the address was not advanced.
- CV4001 assembler: too many operands**  
Too many operands were specified for the most recently issued instruction.  
The instruction was rejected and the address was not advanced.
- CV4002 assembler: incorrect operand size**  
An instruction required an operand of a different size.  
The instruction was rejected and the address was not advanced.

- CV4003 assembler: illegal range**  
The size of a specified value exceeds the size expected by the instruction.  
The instruction was rejected and the address was not advanced.
- CV4004 assembler: overflow**  
Numeric overflow occurred while assembling the current instruction.  
The instruction was rejected and the address was not advanced.
- CV4005 assembler: syntax error**  
The syntax for the instruction is incorrect.  
The instruction was rejected and the address was not advanced.
- CV4006 assembler: unknown opcode**  
An instruction was not recognized.  
Check that the instruction was typed correctly.  
The instruction was rejected and the address was not advanced.
- CV4007 assembler: extra characters**  
The instruction contained extra characters that could not be recognized. The instruction may have been mistyped.  
The line was ignored.  
The instruction was rejected and the address was not advanced.
- CV4008 assembler: illegal operand**  
The wrong type of operand was used for this context.  
The instruction may have been mistyped.  
The instruction was rejected and the address was not advanced.
- CV4009 assembler: illegal segment**  
An invalid segment was used.  
The instruction was rejected and the address was not advanced.
- CV4010 assembler: illegal register**  
An illegal or nonexistent register was accessed.  
The register name may have been mistyped.  
This error can be caused by trying to access 80386- or 80486-specific registers when CodeView is running on an 8088- or 80286-based machine.  
The instruction was rejected and the address was not advanced.

- CV4011 assembler: divide by zero**  
CodeView encountered a divide-by-zero error while assembling the current instruction. The instruction was rejected and the address was not advanced.
- CV4012 cannot assemble code with current execution model**  
This error can be caused by trying to assemble p-code in CodeView.
- CV4500 bad fixed format length: using variable length**  
An invalid length was specified for the Memory window. CodeView will set the length based on the current window width.  
Try specifying a different length.
- CV4501 invalid window id**  
The window ID was invalid. It must be either 0 or 1.
- CV4502 unable to open the requested memory window**  
CodeView could not open a Memory window.  
The only valid window IDs are 0 and 1. You may need to close some windows.
- CV5001 cannot select**  
The cursor was not on the same line as an automatically selectable symbol.
- CV5004 cannot read file**  
CodeView could not read a file.  
Read the file again. If the second read fails, exit and restart CodeView. If the read process still fails, the file may be corrupt.
- CV5005 no file selected**  
A module must be selected before OK is chosen.  
To exit the dialog box without selecting a module, choose Cancel.
- CV5009 no watch expression to delete**  
An attempt was made to delete one or more watch variables (watch expressions), but watch expressions are not currently selected.
- CV5012 packed executable file**  
CodeView cannot step through the beginning of files that are linked with the /EXEPACK option. There are two solutions to this problem:
- Relink without this option to debug the file and then switch back to linking with /EXEPACK for the release version of your program.
  - Execute the program through startup code, and set breakpoints only after the program has entered main.



**CV5013 no expression evaluators found; exiting**

CodeView needs at least one expression evaluator in order to operate.



Check the [cv] or [cvw] section of your TOOLS.INI file and specify at least one Eval entry.

**CV5014 cannot execute function in watch expression**

A watch expression cannot specify a function to be executed.

## CVPACK Error Messages

Microsoft Debugging Information Compactor (CVPACK) generates the following error messages:

-  Fatal errors (CK1xxx) cause CVPACK to stop execution.
-  Warnings (CK4xxx) indicate possible problems in the packing process.

## CVPACK Fatal Error Messages

**CK1000 unknown error; contact Microsoft Product Support Services**

CVPACK detected an unknown error condition.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**CK1001 out of memory**

The executable file is too big for the available memory. This error can occur with MS-DOS when there is little extra memory. Even though CVPACK uses virtual memory, which involves swapping to disk, some information can be stored only in real memory.

One of the following may be a solution:

- 🔧 Assemble and link in separate steps (that is, use NMAKE).
- 🔧 Recompile one or more of the object files without debugging information. If the file was compiled using the /Zi option, use either /Zd or no option.
- 🔧 Add more memory to your computer.

**CK1002 out of virtual memory**

There was not enough virtual memory for CVPACK to pack the executable file. Virtual memory can be any of the following:

- 🔧 Conventional memory. Remove TSR (terminate-and-stay-resident) programs or run CVPACK outside of a shell or a makefile.
- 🔧 Extended or expanded memory. Run CVPACK under a DPMI server, or as an MS-DOS session within the Windows operating system (386 Enhanced Mode).
- 🔧 Disk space. Free some disk storage.

**CK1003 cannot open file**

CVPACK could not open the specified executable file.

One of the following may be a cause:

- 🔧 The specified file does not exist. Check the spelling of the filename and path.
- 🔧 The executable file was opened or deleted by another process.

**CK1004 file is read-only**

CVPACK cannot pack a read-only file. Change the read attribute on the executable file and run CVPACK again.

**CK1005 invalid executable file**

CVPACK could not process the executable file. One of the following may be a cause:

- 🔧 The debugging information in the executable file is corrupt.
- 🔧 The executable file is a zero-length file.

**CK1006 invalid module *module***

The given object file did not have a valid format.

Check the linker version.

**CK1007    invalid *table* table in module *module***

The given table in the given object file was not valid.

Check the compiler and linker versions.

**CK1008    cannot write packed information**

There was not enough space on disk for CVPACK to write the packed executable file. This leaves a corrupt file on disk.

Make more space available on disk and relink the program.

**CK1009    module *module* unknown type index *number*;  
contact Microsoft Product Support Services**

The debugging information in the executable file is corrupt. This is due to an internal error in either the compiler or CVPACK. Recompile the program. If the problem persists, note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**CK1010    symbol error in module *module*;  
contact Microsoft Product Support Services**

The debugging information in the executable file is corrupt. This is due to an internal error in either the compiler or CVPACK. Recompile the program. If the problem persists, note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**CK1011    error in type *number* for module *module*;  
contact Microsoft Product Support Services**

The debugging information in the executable file is corrupt. This is due to an internal error in either the compiler or CVPACK. Recompile the program. If the problem persists, note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**CK1012    no Symbol and Type Information**

The executable file does not contain debugging information.

Link the program using the /CO option to put at least minimal debugging information in the executable file. To include full debugging information in an object file, compile or assemble using the /Zi option. To include minimal information and line numbers, compile or assemble using the /Zd option.

**CK1013     debugging information missing or unknown format**

One of the following has occurred:

- 🔧 The program did not contain debugging information. Recompile using /Zi or /Zd, then link using /CO.
- 🔧 The executable file was linked using an obsolete or unsupported linker. Use Microsoft LINK version 5.3x or later.
- 🔧 The executable file was already packed using a previous version of CVPACK.

**CK1014     module *module* type *number* refers to skipped type index;  
contact Microsoft Product Support Services**

The debugging information in the executable file is corrupt. This is due to an internal error in the compiler. Recompile the program. If the problem persists, note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**CK1015     too many segments in module *module***

The **alloc\_text** pragma was used more than 20 times in an object file that was compiled with Microsoft C version 6.x or earlier.

One of the following may be a solution:

- 🔧 Recompile using Microsoft C/C++ version 7.0 or later.
- 🔧 Split the object file into multiple files.
- 🔧 Group the pragma statements according to segment.

**CK1016     unable to execute MPC for CVPACK /PCODE**

CVPACK could not find MPC.EXE on the path.

**CK1017     precompiled types file *filename* not found**

The program used a precompiled header, but the program was linked without the object file that was created when the header was precompiled.

**CK1018     precompiled types object file *filename* inconsistent with  
precompiled header used to compile object file *filename***

The program used a precompiled header, but the object file linked to the program was not the object file that was created when the header was precompiled. Either the user or the creator changed since the last compilation.

Recompile and relink. If a makefile is used, check the makefile dependencies.

**CK1020    packed type index exceeds 65535 in module *module***

The debugging information exceeded a CVPACK limit.

This error may occur when precompiled headers are used.

One of the following may be a solution:

- 🔧 Eliminate unused type strings.
- 🔧 Compile some object files without debugging information.

**CK1021    error in precompiled types signature in module *module***

The program was compiled with an out-of-date precompiled header.

Delete the object file and recompile.

**CK 1022    Symbol table for *file* is too large**

The corrective action is to compile *file* without CodeView information, reduce the number of symbols in the file, or split the file into two or more pieces.

## CVPACK Warning Messages

**CK4000    unknown warning; contact Microsoft Product Support Services**

CVPACK detected an unknown error condition.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**CK4001    file already packed**

CVPACK took no action because the executable file has already been processed by CVPACK 4.00.

**CK4002    duplicate public symbol *symbol* in module *module***

The given symbol was redefined in the given module. CVPACK deleted the second occurrence of the symbol.

Probably an earlier version of the linker was used. Use LINK 5.30 or later.

**CK4003    error in lexical scopes for module *module*, symbols deleted**

The scoping of symbols in the given object module was corrupted. CVPACK deleted the symbols in the module.

This is probably a compiler error. Recompile and relink the object file.

## EXEHDR Error Messages

This section includes error messages generated by the Microsoft EXE File Header Utility (EXEHDR). EXEHDR errors (U1100 through U1140) are always fatal.

## EXEHDR Fatal Error Messages

- U1110**     **malformed number** *number*  
A command-line option for EXEHDR required a value, but the given number was mistyped.
- U1111**     **option requires value**  
A command-line option for EXEHDR required a value, but no value was specified or the specified value was in an illegal format for the given option.
- U1112**     **value out of legal range** *lower – upper*  
A command-line option for EXEHDR required a value, but the specified number did not fall in the required decimal range.
- U1113**     **value out of legal range** *lower – upper*  
A command-line option for EXEHDR required a value, but the specified number did not fall in the required hexadecimal range.
- U1114**     **missing option value; option** *option* **ignored**  
The given command-line option for EXEHDR required a value, but nothing was specified. EXEHDR ignored the option.
- U1115**     **option** *option* **ignored**  
The given command-line option for EXEHDR was ignored. This error usually occurs with error U1116, unrecognized option.
- U1116**     **unrecognized option:** *option*  
A command-line option for EXEHDR was not recognized. This error usually occurs with either U1115, option ignored, or U1111, option requires value.
- U1120**     **input file missing**  
No input file was specified on the EXEHDR command line.
- U1121**     **command line too long:** *commandline*  
The given EXEHDR command line exceeded the limit of 512 characters.
- U1130**     **cannot read** *filename*  
EXEHDR could not read the input file. Either the file is missing or the file attribute is set to prevent reading.
- U1131**     **invalid .EXE file**  
The input file specified on the EXEHDR command line was not recognized as an executable file.
- U1132**     **unexpected end-of-file**  
EXEHDR found an unexpected end-of-file condition while reading the executable file. The file is probably corrupt.

**U1140 out of memory**

There was not enough memory for EXEHDR to decode the header of the executable file.

## Math Coprocessor Error Messages

The error messages listed below correspond to exceptions generated by the math coprocessor hardware. Refer to the manufacturer's documentation for your processor for a detailed discussion of hardware exceptions. These errors may also be detected by the floating-point emulator or alternate math library.

**M6101 invalid**

An invalid operation occurred. This error usually occurs when the operand is NAN (not a number) or infinity.

This error terminates the program with exit code 129.

**M6102 denormal**

A very small floating-point number was generated, which may no longer be valid because of a loss of significance. Denormal floating-point exceptions are usually masked, causing them to be trapped and operated upon.

This error terminates the program with exit code 130.

**M6103 divide by 0**

A floating-point operation attempted to divide by zero.

This error terminates the program with exit code 131.

**M6104 overflow**

An overflow occurred in a floating-point operation.

This error terminates the program with exit code 132.

**M6105 underflow**

An underflow occurred in a floating-point operation. Underflow floating-point exceptions are usually masked, causing the underflowing value to be replaced by 0.0.

This error terminates the program with exit code 133.

**M6106 inexact**

Loss of precision occurred in a floating-point operation. This exception is usually masked. Many floating-point operations cause a loss of precision.

This error terminates the program with exit code 134.

**M6107 unemulated**

An attempt was made to execute a coprocessor instruction that is invalid or is not supported by the emulator.

This error terminates the program with exit code 135.

**M6108 square root**

The operand in a square-root operation was negative.

This error terminates the program with exit code 136.

The **sqrt** function in the C run-time library and the FORTRAN intrinsic function **SQRT** do not generate this error. The C **sqrt** function checks the argument before performing the operation and returns an error value if the operand is negative. The FORTRAN **SQRT** function generates the DOMAIN error M6201 instead of this error.

**M6110 stack overflow**

A floating-point expression caused a stack overflow on the 8087/80287/80387 coprocessor or the emulator.

Stack-overflow floating-point exceptions are trapped up to a limit of seven levels in addition to the eight levels usually supported by the 8087/80287/80387 coprocessor.

This error terminates the program with exit code 138.

**M6111 stack underflow**

A floating-point operation resulted in a stack underflow on the 8087/80287/80387 coprocessor or the emulator.

This error terminates the program with exit code 139.

**M6201 *function* : \_DOMAIN error**

An argument to the given function was outside the domain of legal input values for that function.

**M6202 *function* : \_SING error**

An argument to the given function was a singularity value for this function. The function is not defined for that argument.

For example, in FORTRAN the following statement generates this error:

```
result = LOG10(0.0)
```

This error calls the **\_matherr** function with the function name, its arguments, and the error type. You can rewrite the **\_matherr** function to customize the handling of certain run-time floating-point math errors.



- M6203**     *function* : **\_OVERFLOW error**  
The given function result was too large to be represented.  
This error calls the **\_matherr** function with the function name, its arguments, and the error type. You can rewrite the **\_matherr** function to customize the handling of certain run-time floating-point math errors.
- M6205**     *function* : **\_TLOSS error**  
A total loss of significance (precision) occurred.  
This error may be caused by giving a very large number as the operand of **sin**, **cos**, or **tan** because the operand must be reduced to a number between 0 and  $2\pi$ .

## H2INC Error Messages

### H2INC Fatal Errors

- HI1003**     **error count exceeds n; stopping compilation**  
Errors in the program were too numerous or too severe to allow recovery, and the compiler must terminate.
- HI1004**     **unexpected end-of-file found**  
The default disk drive did not contain sufficient space for the compiler to create temporary files. The space required is approximately two times the size of the source file.  
This message also appears when the **#if** directive occurs without a corresponding closing **#endif** directive while the **#if** test directs the compiler to skip the section.
- HI1007**     **unrecognized flag *string* in *option***  
The *string* in the command-line *option* was not a valid option.
- HI1008**     **no input file specified**  
The compiler was not given a file to compile.
- HI1009**     **compiler limit : macros nested too deeply**  
Too many macros were being expanded at the same time.  
This error occurs when a macro definition contains macros to be expanded and those macros contain other macros.  
Try to split the nested macros into simpler macros.
- HI1011**     **compiler limit : *identifier* : macro definition too big**  
The macro definition was longer than allowed.  
Split the definition into shorter definitions.

- HI1012**      **unmatched parenthesis nesting - missing *character***  
The parentheses in a preprocessor directive were not matched. The missing character is either a left, (, or right, ), parenthesis.
- HI1016**      **#if[n]def expected an identifier**  
An identifier must be specified with the **#ifdef** and **#ifndef** directives.
- HI1017**      **invalid integer constant expression**  
The expression in an **#if** directive either did not exist or did not evaluate to a constant.
- HI1018**      **unexpected '#elif'**  
The **#elif** directive is legal only when it appears within an **#if**, **#ifdef**, or **#ifndef** construct.
- HI1019**      **unexpected '#else'**  
The **#else** directive is legal only when it appears within an **#if**, **#ifdef**, or **#ifndef** construct.
- HI1020**      **unexpected '#endif'**  
An **#endif** directive appeared without a matching **#if**, **#ifdef**, or **#ifndef** directive.
- HI1021**      **invalid preprocessor command *string***  
The characters following the number sign (#) did not form a valid preprocessor directive.
- HI1022**      **expected '#endif'**  
An **#if**, **#ifdef**, or **#ifndef** directive was not terminated with an **#endif** directive.
- HI1023**      **cannot open source file *filename***  
The given file either did not exist, could not be opened, or was not found.  
Make sure the environment settings are valid and that the correct path name for the file is specified.  
If this error appears without an error message, the compiler has run out of file handles. If in MS-DOS, increase the number of file handles by changing the FILES setting in CONFIG.SYS to allow a larger number of open files. FILES=20 is the recommended setting.
- HI1024**      **cannot open include file *filename***  
The specified file in an **#include** preprocessor directive could not be found.  
Make sure settings for the INCLUDE and TMP environment variables are valid and that the correct path name for the file is specified.  
If this error appears without an error message, the compiler has run out of file handles. If in MS-DOS, increase the number of file handles by changing the FILES setting in CONFIG.SYS to allow a larger number of open files. FILES=20 is the recommended setting.

**HI1026      parser stack overflow, please simplify your program**

The program cannot be processed because the space required to parse the program causes a stack overflow in the compiler.

Simplify the program by decreasing the complexity of expressions. Decrease the level of nesting in for and switch statements by putting some of the more deeply nested statements in separate functions. Break up very long expressions involving ',' operators or function calls.

**HI1033      cannot open assembly language output file *filename***

There are several possible causes for this error:

- 🔒 The given name is not valid
- 🔒 The file cannot be opened for lack of space.
- 🔒 A read-only file with the given name already exists.

**HI1036      cannot open source listing file *filename***

There are several possible causes for this error:

- 🔒 The given name is not valid.
- 🔒 The file cannot be opened for lack of space.
- 🔒 A read-only file with the given name already exists.

**HI1039      unrecoverable heap overflow in Pass 3**

The post-optimizer compiler pass overflowed the heap and could not continue.

One of the following may be a solution:

- 🔒 Break up the function containing the line that caused the error.
- 🔒 Recompile with the /Od option, removing optimization.
- 🔒 In MS-DOS, remove other programs or drivers running in the system which could be consuming significant amounts of memory.
- 🔒 In MS-DOS, if using NMAKE, compile without using NMAKE.

**HI1040      unexpected end-of-file in source file *filename***

The compiler detected an unexpected end-of-file condition while creating a source listing or mingled source/object listing.

**HI1047      limit of option exceeded at *string***

The given option was specified too many times. The given string is the argument to the option that caused the error.

If the CL or H2INC environment variables have been set, options in these variables are read before options specified on the command line. The CL environment variable is read before the H2INC environment variable.

- HI1048** This error existed in previous versions of H2INC as “unknown option *character* in *option*.” This condition now generates warning HI4799.
- HI1049** This error existed in previous versions of H2INC as “invalid numerical argument *string*.” This condition now generates warning HI4052.
- HI1050** ***segment : code segment too large***  
A code segment grew to within 36 bytes of 64K during compilation.  
A 36-byte pad is used because of a bug in some 80286 chips that can cause programs to exhibit strange behavior when, among other conditions, the size of a code segment is within 36 bytes of 64K.
- HI1052** **compiler limit : #if/#ifdef nested too deeply**  
The program exceeded the maximum of 32 nesting levels for **#if** and **#ifdef** directives.
- HI1053** **compiler limit : struct/union nested too deeply**  
A structure or union definition was nested to more than 15 levels.  
Break the structure or union into two parts by defining one or more of the nested structures using **typedef**.
- HI1090** ***segment data allocation exceeds 64K***  
The size of the named segment exceeds 64K.  
This error occurs with **\_based** allocation.
- HI1800** This error existed in previous versions of H2INC as “*option*: unrecognized option.” This condition now generates warning HI4799.
- HI1801** **incomplete model specification**  
Only part of a custom memory-model specification was specified on the command line.  
When you specify a custom memory model with the /A command-line option, you must specify code pointer distance, data pointer distance, and DS register setup. This error is equivalent to the D2013 error message for CL.

## H2INC Nonfatal Errors

- HI2000** **UNKNOWN ERROR Contact Microsoft Product Support Services**  
The compiler detected an unknown error condition.  
Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**HI2001      newline in constant**

A string constant was continued onto a second line without either a backslash or closing and opening quotes.

To break a string constant onto two lines in the source file, do one of the following:

- 1. End the first line with the line-continuation character, a backslash, \.
- 2. Close the string on the first line with a double quotation mark, and open the string on the next line with another quotation mark.

It is not sufficient to end the first line with \n, the escape sequence for embedding a newline character in a string constant.

The following two examples demonstrate causes of this error:

```
printf("Hello,  
world");
```

OR

```
printf("Hello, \n  
world");
```

The following two examples show ways to correct this error:

```
printf("Hello, \n  
world");
```

OR

```
printf("Hello, "  
" world");
```

Note that any spaces at the beginning of the next line after a line-continuation character are included in the string constant. Note, also, that neither solution actually places a newline character into the string constant. To embed this character:

```
printf("Hello, \n\  
world");
```

OR

```
printf("Hello, \nworld");
```

or

```
printf("Hello, \n" "world");
```

or

```
printf("Hello, "\nworld");
```

**HI2003**     **expected *defined id***

An identifier was expected after the preprocessing keyword *defined*.

**HI2004**     **expected *defined(id)***

An identifier was expected after the left parenthesis, (, following the preprocessing keyword *defined*.

**HI2005**     **#line expected a line number, found *token***

A **#line** directive lacked the required line-number specification.

**HI2006**     **#include expected a file name, found *token***

An **#include** directive lacked the required file-name specification.

**HI2007**     **#define syntax**

An identifier was expected following **#define** in a preprocessing directive.

**HI2008**     ***character* : unexpected in macro definition**

The given character was found immediately following the name of the macro.

**HI2009**     **reuse of macro formal *identifier***

The given identifier was used more than once in the formal-parameter list of a macro definition.

**HI2010**     ***character* : unexpected in macro formal-parameter list**

The given character was used incorrectly in the formal-parameter list of a macro definition.

**HI2012**     **missing name following '<'**

An **#include** directive lacked the required filename specification.

**HI2013**     **missing '>'**

The closing angle bracket (>) was missing from an **#include** directive.

- HI2014**     **preprocessor command must start as first non-white-space**  
Non-white-space characters appeared before the number sign (#) of a preprocessor directive on the same line.
- HI2015**     **too many characters in constant**  
A character constant contained more than one character.  
Note that an escape sequence (for example, \t for tab) is converted to a single character.
- HI2016**     **no closing single quotation mark**  
A newline character was found before the closing single quotation mark of a character constant.
- HI2017**     **illegal escape sequence**  
An escape sequence appeared where one was not expected.  
An escape sequence (a backslash, \, followed by a number or letter) may occur only in a character or string constant.
- HI2018**     **unknown character** *hexnumber*  
The ASCII character corresponding to the given hexadecimal number appeared in the source file but is an illegal character.  
One possible cause of this error is corruption of the source file. Edit the file and look at the line on which the error occurred.
- HI2019**     **expected preprocessor directive, found** *character*  
The given character followed a number sign (#), but it was not the first letter of a preprocessor directive.
- HI2021**     **expected exponent value, not** *character*  
The given character was used as the exponent of a floating-point constant but was not a valid number.
- HI2022**     **number : too big for character**  
The octal number following a backslash (\) in a character or string constant was too large to be represented as a character.
- HI2025**     **identifier : enum/struct/union type redefinition**  
The given identifier had already been used for an enumeration, structure, or union tag.
- HI2026**     **identifier : member of enum redefinition**  
The given identifier has already been used for an enumeration constant, either within the same enumeration type or within another visible enumeration type.
- HI2027**     **use of undefined enum/struct/union** *identifier*  
The given identifier referred to a structure or union type that was not defined.

**HI2028     struct/union member needs to be inside a struct/union**

Structure and union members must be declared within the structure or union.

This error may be caused by an enumeration declaration containing a declaration of a structure member, as in the following example:

```
enum a {  
  j anuary,  
  february,  
  int march; /* Illegal structure declaration */  
};
```

**HI2030     *identifier* : struct/union member redefinition**

The identifier was used for more than one member of the same structure or union.

**HI2031     *identifier* : function cannot be struct/union member**

The given function was declared to be a member of a structure or union.

To correct this error, use a pointer to the function instead.

**HI2033     *identifier* : bit field cannot have indirection**

The given bit field was declared as a pointer (\*), which is not allowed.

**HI2034     *identifier* : type of bit field too small for number of bits**

The number of bits specified in the bit-field declaration exceeded the number of bits in the given base type.

**HI2035     struct/union *identifier* : unknown size**

The given structure or union had an undefined size.

Usually this occurs when referencing a declared but not defined structure or union tag.

For example, the following causes this error:

```
struct s_tag *ps;  
ps = &my_var;  
*ps = 17; /* This line causes the error */
```

**HI2037     left of operator specifies undefined struct/union *identifier***

The expression before the member-selection operator ( -> or .) identified a structure or union type that was not defined.

**HI2038     *identifier* : not struct/union member**

The given identifier was used in a context that required a structure or union member.

**HI2041     illegal digit character for base number**

The given character was not a legal digit for the base used.



**HI2042 signed/unsigned keywords mutually exclusive**

The keywords *signed* and *unsigned* were both used in a single declaration, as in the following example:

```
unsigned signed int i;
```

**HI2056 illegal expression**

An expression was illegal because of a previous error, which may not have produced an error message.

**HI2057 expected constant expression**

The context requires a constant expression.

**HI2058 constant expression is not integral**

The context requires an integral constant expression.

**HI2059 syntax error : *token***

The token caused a syntax error.

**HI2060 syntax error : end-of-file found**

The compiler expected at least one more token.

Some causes of this error include:

❗ Omitting a semicolon (;), as in

```
int *p
```

❗ Omitting a closing brace (}) from the last function, as in

```
main()  
{
```

**HI2061 syntax error : identifier *identifier***

The identifier caused a syntax error.

**HI2062 type *type* unexpected**

The compiler did not expect the given type to appear here, possibly because it already had a required type.

**HI2063 identifier : not a function**

The given identifier was not declared as a function, but an attempt was made to use it as a function.

**HI2064 term does not evaluate to a function**

An attempt was made to call a function through an expression that did not evaluate to a function pointer.

- HI2065**     *identifier* : **undefined**  
An attempt was made to use an identifier that was not defined.
- HI2066**     **cast to function type is illegal**  
An object was cast to a function type, which is illegal.  
However, it is legal to cast an object to a function pointer.
- HI2067**     **cast to array type is illegal**  
An object was cast to an array type.
- HI2068**     **illegal cast**  
A type used in a cast operation was not legal for this expression.
- HI2069**     **cast of void term to nonvoid**  
The void type was cast to a different type.
- HI2070**     **illegal sizeof operand**  
The operand of a **sizeof** expression was not an identifier or a type name.
- HI2071**     *identifier* : **illegal storage class**  
The given storage class cannot be used in this context.
- HI2072**     *identifier* : **initialization of a function**  
An attempt was made to initialize a function.
- HI2043**     **illegal break**  
A break statement is legal only within a do, for, while, or switch statement.
- HI2044**     **illegal continue**  
A continue statement is legal only within a do, for, or while statement.
- HI2045**     *identifier* : **label redefined**  
The label appeared before more than one statement in the same function.
- HI2046**     **illegal case**  
The keyword case may appear only within a switch statement.
- HI2047**     **illegal default**  
The keyword default may appear only within a switch statement.
- HI2048**     **more than one default**  
A switch statement contained more than one default label.
- HI2049**     **case value *value* already used**  
The case value was already used in this switch statement.

- HI2050 nonintegral switch expression**  
A switch expression did not evaluate to an integral value.
- HI2051 case expression not constant**  
Case expressions must be integral constants.
- HI2052 case expression not integral**  
Case expressions must be integral constants.
- HI2054 expected '(' to follow *identifier***  
The context requires parentheses after the function identifier.  
One cause of this error is forgetting an equal sign (=) on a complex initialization, as in
- ```
int array1[] /* Missing = */
{
    1, 2, 3
};
```
- HI2055 expected formal-parameter list, not a type list**  
An argument-type list appeared in a function definition instead of a formal-parameter list.
- HI2075 *identifier* : array initialization needs curly braces**  
There were no curly braces, {}, around the given array initializer.
- HI2076 *identifier* : struct/union initialization needs curly braces**  
There were no curly braces, {}, around the given structure or union initializer.
- HI2077 nonscalar field initializer *identifier***  
An attempt was made to initialize a bit-field member of a structure with a nonscalar value.
- HI2078 too many initializers**  
The number of initializers exceeded the number of objects to be initialized.
- HI2079 *identifier* uses undefined struct/union *name***  
The *identifier* was declared as structure or union type name, but the *name* had not been defined. This error may also occur if an attempt is made to initialize an anonymous union.
- HI2080 illegal far \_fastcall function**  
A far \_fastcall function may not be compiled with the /Gw option, or with the /Gq option if stack checking is enabled.
- HI2082 redefinition of formal parameter *identifier***  
A formal parameter to a function was redeclared within the function body.

**HI2084      function *function* already has a body**

The function has already been defined.

**HI2086      *identifier* : redefinition**

The given identifier was defined more than once, or a subsequent declaration differed from a previous one.

The following are ways to cause this error:

```
int a;
char a;
main()
{
}
main()
{
int a;
int a;
}
```

However, the following does not cause this error:

```
int a;
int a;
main()
{
}
```

**HI2087      *identifier* : missing subscript**

The definition of an array with multiple subscripts was missing a subscript value for a dimension other than the first dimension.

The following is an example of an illegal definition:

```
int func(a)
char a[10][ ];
{ }
```

The following is an example of a legal definition:

```
int func(a)
char a[][5];
{ }
```

**HI2090      function returns array**

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

**HI2091      function returns function**

A function cannot return a function. It can return a pointer to a function.

- HI2092     array element type cannot be function**  
Arrays of functions are not allowed. Arrays of pointers to functions are allowed.
- HI2095     *function* : actual has type void : parameter *number***  
An attempt was made to pass a void argument to a function. The given number indicates which argument was in error.  
  
Formal parameters and arguments to functions cannot have type void. They can, however, have type void \* (pointer to void).
- HI2100     illegal indirection**  
The indirection operator (\*) was applied to a nonpointer value.
- HI2101     '&' on constant**  
The address-of operator (&) did not have an **lvalue** as its operand.
- HI2102     '&' requires lvalue**  
The address-of operator (&) must be applied to an **lvalue** expression.
- HI2103     '&' on register variable**  
An attempt was made to take the address of a register variable.
- HI2104     '&' on bit field ignored**  
An attempt was made to take the address of a bit field.
- HI2105     *operator* needs lvalue**  
The given operator did not have an **lvalue** operand.
- HI2106     *operator* : left operand must be lvalue**  
The left operand of the given operator was not an lvalue.
- HI2107     illegal index, indirection not allowed**  
A subscript was applied to an expression that did not evaluate to a pointer.
- HI2108     nonintegral index**  
A nonintegral expression was used in an array subscript.
- HI2109     subscript on nonarray**  
A subscript was used on a variable that was not an array.
- HI2110     pointer + pointer**  
An attempt was made to add one pointer to another using the plus (+) operator.
- HI2111     pointer + nonintegral value**  
An attempt was made to add a nonintegral value to a pointer.
- HI2112     illegal pointer subtraction**  
An attempt was made to subtract pointers that did not point to the same type.

**HI2113      pointer subtracted from nonpointer**

The right operand in a subtraction operation using the minus (-) operator was a pointer, but the left operand was not.

**HI2114      operator : pointer on left; needs integral right**

The left operand of the given operator was a pointer; so the right operand must be an integral value.

**HI2115      identifier : incompatible types**

An expression contained incompatible types.

**HI2117      operator : illegal for struct/union**

Structure and union type values are not allowed with the given operator.

**HI2118      negative subscript**

A value defining an array size was negative.

**HI2120      void illegal with all types**

The void type was used in a declaration with another type.

**HI2121      operator : bad left/right operand**

The left or right operand of the given operator was illegal for that operator.

**HI2124      divide or mod by zero**

A constant expression was evaluated and found to have a zero denominator.

**HI2128      identifier : huge array cannot be aligned to segment boundary**

The given huge array was large enough to cross two segment boundaries, but could not be aligned to both boundaries to prevent an individual array element from crossing a boundary.

If the size of a huge array causes it to cross two boundaries, the size of each array element must be a power of two, so that a whole number of elements will fit between two segment boundaries.

**HI2129      static function *function* not found**

A forward reference was made to a static function that was never defined.

**HI2130      #line expected a string containing the file name, found *token***

The optional token following the line number on a #line directive was not a string.

**HI2131      more than one memory attribute**

More than one of the keywords **\_near**, **\_far**, **\_huge**, or **\_based** were applied to an item, as in the following example:

```
typedef int _near nint;  
nint _far a; /* Illegal */
```

- HI2132     syntax error : unexpected identifier**  
An identifier appeared in a syntactically illegal context.
- HI2133     *identifier* : unknown size**  
An attempt was made to declare an unsized array as a local variable.
- HI2134     *identifier* : struct/union too large**  
The size of a structure or union exceeded the 64K compiler limit.
- HI2136     *function* : prototype must have parameter types**  
A function prototype declarator had formal-parameter names, but no types were provided for the parameters.  
  
A formal parameter in a function prototype must either have a type or be represented by an ellipsis (...) to indicate a variable number of arguments and no type checking.  
  
One cause of this error is a misspelling of a type name in a prototype that does not provide the names of formal parameters.
- HI2137     empty character constant**  
The illegal empty-character constant (") was used.
- HI2139     type following *identifier* is illegal**  
Two types were used in the same declaration.  
  
For example:  
  
`int double a;`
- HI2141     value out of range for enum constant**  
An enumeration constant had a value outside the range of values allowed for type int.
- HI2143     syntax error : missing *token1* before *token2***  
The compiler expected *token1* to appear before *token2*.  
  
This message may appear if a required closing brace (}), right parenthesis ()), or semicolon (;) is missing.
- HI2144     syntax error : missing *token* before type *type***  
The compiler expected the given token to appear before the given type name.  
  
This message may appear if a required closing brace (}), right parenthesis ()), or semicolon (;) is missing.
- HI2145     syntax error : missing *token* before identifier**  
The compiler expected the given token to appear before an identifier.  
  
This message may appear if a semicolon (;) does not appear after the last declaration of a block.

- HI2146**     **syntax error : missing *token* before identifier *identifier***  
The compiler expected the given token to appear before the given identifier.
- HI2147**     **unknown size**  
An attempt was made to increment an index or pointer to an array whose base type has not yet been declared.
- HI2148**     **array too large**  
An array exceeded the maximum legal size of 64K.  
Either reduce the size of the array, or declare it with **`_huge`**.
- HI2149**     ***identifier* : named bit field cannot have 0 width**  
The given named bit field had zero width. Only unnamed bit fields are allowed to have zero width.
- HI2150**     ***identifier* : bit field must have type int, signed int, or unsigned int**  
The ANSI C standard requires bit fields to have types of int, signed int, or unsigned int. This message appears only when compiling with the `/Za` option.
- HI2151**     **more than one language attribute**  
More than one keyword specifying a calling convention for a function was given.
- HI2152**     ***identifier* : pointers to functions with different attributes**  
An attempt was made to assign a pointer to a function declared with one calling convention (**`_cdecl`**, **`_fortran`**, **`_pascal`**, or **`_fastcall`**) to a pointer to a function declared with a different calling convention.
- HI2153**     **hex constants must have at least 1 hex digit**  
The hexadecimal constants `0x`, `0X` and `\x` are illegal. At least one hexadecimal digit must follow the `x` or `X`.
- HI2154**     ***segment* : does not refer to a segment name**  
A **`_based`**-allocated variable must be allocated in a segment unless it is extern and uninitialized.
- HI2156**     **pragma must be outside function**  
A pragma that must be specified at a global level, outside a function body, occurred within a function.  
  
For example, the following causes this error:
- ```
main()
{
#pragma optimize("l", on)
}
```



- HI2157**     *function* : **must be declared before use in pragma list**  
The function name in the list of functions for an **alloc\_text** pragma has not been declared prior to being referenced in the list.
- HI2158**     *identifier* : **is a function**  
The given identifier was specified in the list of variables in a **same\_seg** pragma but was previously declared as a function.
- HI2159**     **more than one storage class specified**  
A declaration contained more than one storage class, as in  
`extern static int i;`
- HI2160**     **## cannot occur at the beginning of a macro definition**  
A macro definition began with a token-pasting operator (##), as in  
`#define mac(a, b) ##a`
- HI2161**     **## cannot occur at the end of a macro definition**  
A macro definition ended with a token-pasting operator (##), as in  
`#define mac(a, b) a##`
- HI2162**     **expected macro formal parameter**  
The token following a stringizing operator (#) was not a formal-parameter name.  
For example:  
`#define print(a) printf(#b)`
- HI2165**     *keyword* : **cannot modify pointers to data**  
The **\_fortran**, **\_pascal**, **\_cdecl**, or **\_fastcall** keyword was used illegally to modify a pointer to data, as in the following example:  
`char _pascal *p;`
- HI2166**     **lvalue specifies const object**  
An attempt was made to modify an item declared with **const** type.
- HI2167**     *function* : **too many actual parameters for intrinsic function**  
A reference to the intrinsic function name contained too many actual parameters.
- HI2168**     *function* : **too few actual parameters for intrinsic function**  
A reference to the intrinsic function name contained too few actual parameters.

**HI2171**     *operator : illegal operand*

The given unary operator was used with an illegal operand type, as in the following example:

```
int (*fp)();  
double d, d1;  
fp++;  
d = ~d1;
```

**HI2172**     *function : actual is not a pointer : parameter number*

An attempt was made to pass an argument that was not a pointer to a function that expected a pointer. The given number indicates which argument was in error.

**HI2173**     *function : actual is not a pointer : parameter number1, parameter list number2*

An attempt was made to pass a nonpointer argument to a function that expected a pointer.

This error occurs in calls that return a pointer to a function. The first number indicates which argument was in error; the second number indicates which argument list contained the invalid argument.

**HI2174**     *function : actual has type void : parameter number1, parameter list number2*

An attempt was made to pass a void argument to a function. Formal parameters and arguments to functions cannot have type void. They can, however, have type void \* (pointer to void).

This error occurs in calls that return a pointer to a function. The first number indicates which argument was in error; the second number indicates which argument list contained the invalid argument.

**HI2177**     **constant too big**

Information was lost because a constant value was too large to be represented in the type to which it was assigned.

**HI2178**     *identifier : storage class for same\_seg variables must be extern*

The given variable was specified in a **same\_seg** pragma, but it was not declared with extern storage class.

**HI2179**     *identifier : was used in same\_seg, but storage class is no longer extern*

The given variable was specified in a **same\_seg** pragma, but it was redeclared with a storage class other than extern.

**HI2185**     *identifier* : **illegal \_based allocation**

A **\_based**-allocated variable that explicitly has extern storage class and is uninitialized may not have a base of any of the following:

```
(_segment) & var  
_segname("_STACK")  
(_segment)_self  
void
```

If the variable does not explicitly have extern storage class or it is uninitialized, then its base must use **\_segname**("string") where *string* is any segment name or reserved segment name except "\_STACK".

**HI2187**     **cast of near function pointer to far function pointer**

An attempt was made to cast a near function pointer as a far function pointer.

**HI2189**     **#error** : *string*

An **#error** directive was encountered. The *string* is the descriptive text supplied in the directive.

**HI2193**     *identifier* : **already in a segment**

A variable in the **same\_seg** pragma has already been allocated in a segment, using **\_based**.

**HI2194**     *segment* : **is a text segment**

The given text segment was used where a data, const, or bss segment was expected.

**HI2195**     *segment* : **is a data segment**

The given data segment was used where a text segment was expected.

**HI2200**     *function* : **function has already been defined**

A function name passed as an argument in an **alloc\_text** pragma has already been defined.

**HI2201**     *function* : **storage class must be extern**

A function declaration appears within a block, but the function is not declared extern. This causes an error if the /Za option is in effect.

For example, the following causes this error, when compiled with /Za:

```
main()  
{  
  static int func1();  
}
```

**HI2205**     *identifier* : **cannot initialize extern block-scoped variables**

A variable with extern storage class may not be initialized in a function.

- HI2208      no members defined using this type**  
An **enum**, **struct**, or **union** was defined without any members. This is an error only when compiling with /Za; otherwise, it is a warning.
- HI2209      type cast in \_based construct must be (\_segment)**  
The only type allowed within a cast in a **\_based** declarator is (**\_segment**).
- HI2210      identifier : must be near/far data pointer**  
The base in a **\_based** declarator must not be an array, a function, or a **\_based** pointer.
- HI2211      (\_segment) applied to function identifier *function***  
The item cast in a **\_based** declarator must not be a function.
- HI2212      identifier : \_based not available for functions/pointers to functions**  
Functions cannot be **\_based**-allocated. Use the **alloc\_text** pragma.
- HI2213      identifier : illegal argument to \_based**  
A symbol used as a base must have type **\_segment** or be a near or far pointer.
- HI2214      pointers based on void require the use of :>**  
A **\_based** pointer based on void cannot be dereferenced. Use the **:>** operator to create an address that can be dereferenced.
- HI2215      :> operator only for objects based on void**  
The right operand of the **:>** operator must be a pointer based on void, as in  
`char _based(void) *cbvpi`
- HI2216      attribute1 may not be used with attribute2**  
The given function attributes are incompatible.  
Some combinations of attributes that cause this error are
- 🔒 **\_saveregs** and **\_interrupt**
  - 🔒 **\_fastcall** and **\_saveregs**
  - 🔒 **\_fastcall** and **\_interrupt**
  - 🔒 **\_fastcall** and **\_export**

**HI2217**     *attribute1 must be used with attribute2*

The first function attribute requires the second attribute to be used.

Some causes for this error include

- An interrupt function explicitly declared as **near**. Interrupt functions must be **far**.
- An interrupt function or a function with a variable number of arguments, when that function is declared with the **\_fortran**, **\_pascal**, or **\_fastcall** attribute. Functions declared with the **\_interrupt** attribute or with a variable number of arguments must use the C calling conventions. Remove the **\_fortran**, **\_pascal**, or **\_fastcall** attribute from the function declaration.

**HI2218**     **type in \_based construct must be void**

The only type allowed within a **\_based** construct is **void**.

**HI2219**     **syntax error : type qualifier must be after '\*'**

Either **const** or **volatile** appeared where a type or qualifier is not allowed, as in

```
int (const *p);
```

**HI2220**     **warning treated as error - no object file generated**

When the compiler option **/WX** is used, the first warning generated by the compiler causes this error message to be displayed.

Either correct the condition that caused the warning, or compile at a lower warning level or without **/WX**.

**HI2221**     **'.' : left operand points to struct/union, use '->'**

The left operand of the **'.'** operator must be a **struct/union** type. It cannot be a pointer to a **struct/union** type.

This error usually means that a **->** operator must be used.

**HI2222**     **-> : left operand has struct/union type, use '.'**

The left operand of the **->** operator must be a pointer to a **struct/union** type. It cannot be a **struct/union** type.

This error usually means that a **'.'** operator must be used.

**HI2223**     **left of ->member must point to struct/union**

The left operand of the **->** operator is not a pointer to a **struct/union** type.

This error can occur when the left operand is an undefined variable. Undefined variables have type **int**.

**HI2224      left of *.member* must have struct/union type**

The left operand of the '.' operator is not a **struct/union** type.

This error can occur when the left operand is an undefined variable. Undefined variables have type **int**.

**HI2225      *tagname* : first member of struct is unnamed**

The **struct** with the given tag started with an unnamed member (an alignment member). **Struct** definitions must start with a named member.

## H2INC Warnings

**HI4000      UNKNOWN WARNING Contact Microsoft Product Support Services**

**(level 1)**      The compiler detected an unknown error condition.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the "Microsoft Support Services" section of the introduction to this book.

**HI4001      nonstandard extension used - *extension***

**(level 1, 4)**      The given nonstandard language extension was used when the /Ze option was specified.

This is a level 4 warning, except in the case of a function pointer cast to data when the Quick Compile option, /qc, is in use, which produces a level 1 warning.

If the /Za option has been specified, this condition generates a syntax error.

**HI4002      too many actual parameters for macro *identifier***

**(level 1)**      The number of actual arguments specified with the given identifier was greater than the number of formal parameters given in the macro definition of the identifier.

The additional actual parameters are collected but ignored during expansion of the macro.

**HI4003      not enough actual parameters for macro *identifier***

**(level 1)**      The number of actual arguments specified with the given identifier was less than the number of formal parameters given in the macro definition of the identifier.

When a formal parameter is referenced in the definition and the corresponding actual parameter has not been provided, empty text is substituted in the macro expansion.

**HI4004**      **missing ')' after *defined*****(level 1)**

The closing parenthesis was missing from an **#if** defined phrase.

The compiler assumes a right parenthesis, ), after the first identifier it finds. It then attempts to compile the remainder of the line, which may result in another warning or error.

The following example causes this warning and a fatal error:

```
#if defined( ID1 ) || ( ID2 )
```

The compiler assumed a right parenthesis after ID1, then found a mismatched parenthesis in the remainder of the line. The following avoids this problem:

```
#if defined( ID1 ) || defined( ID2 )
```

**HI4005**      ***identifier* : macro redefinition****(level 1)**

The given identifier was defined twice. The compiler assumed the new macro definition.

To eliminate the warning, either remove one of the definitions or use an **#undef** directive before the second definition.

This warning is caused in situations where a macro is defined both on the command line and in the code with a **#define** directive.

**HI4006**      ***#undef* expected an identifier****(level 1)**

The name of the identifier whose definition was to be removed was not given with the **#undef** directive. The **#undef** was ignored.

**HI4007**      ***identifier* : must be attribute****(level 2)**

The attribute of the given function was not explicitly stated. The compiler forced the attribute.

For example, the function main must have the **\_cdecl** attribute.

**HI4008**      ***identifier* : *\_fastcall* attribute on data ignored****(level 2)**

The **\_fastcall** attribute on the given data identifier was ignored.

**HI4009**      **string too big, trailing characters truncated****(level 1)**

A string exceeded the compiler limit of 2047 on string size. The excess characters at the end of the string were truncated.

To correct this problem, break the string into two or more strings.

**HI4010      identifier is a MASM keyword****(level 1)**

This warning is issued if the .h include file tries to redefine a MASM keyword.

H2INC will give a warning whenever such conflicts take place. This includes **#define**, **typedef**, structures, and other variables. If you want to redefine a MASM keyword, use **#define** instead. A **#define** in the .INC file will not try to redefine the MASM keyword unless the /Ht option is set.

This warning will also be issued anytime converting a **typedef** statement will result in a type with the same name as the type. The translation is not done in this case. For more information on warning HI4010, see "Miscellaneous Utilities."

**HI4011      identifier truncated to identifier****(level 1)**

Only the first 31 characters of an identifier are significant. The characters after the limit were truncated.

This may mean that two identifiers that are different before truncation may have the same identifier name after truncation.

**HI4015      identifier : bit-field type must be integral****(level 1)**

The given bit field was not declared as an integral type. The compiler assumed the base type of the bit field to be unsigned.

Bit fields must be declared as unsigned integral types.

**HI4016      function : no function return type, using int as default****(level 3)**

The given function had not yet been declared or defined, so the return type was unknown. A default return type of **int** was assumed.

**HI4017      cast of int expression to far pointer****(level 1)**

A far pointer represents a full segmented address. On an 8086/8088 processor, casting an **int** value to a far pointer may produce an address with a meaningless segment value.

The compiler extended the **int** expression to a 4-byte value.

**HI4020      function : too many actual parameters****(level 1)**

The number of arguments specified in a function call was greater than the number of parameters specified in the function prototype or function definition.

The extra parameters were passed according to the calling convention used on the function.

**HI4021      function : too few actual parameters****(level 1)**

The number of arguments specified in a function call was less than the number of parameters specified in the function prototype or function definition.

Only the provided actual parameters are passed. If the called function references a variable that was not passed, the results are undefined and may be unexpected.



- HI4022** *function : pointer mismatch : parameter number*  
(level 1) The pointer type of the given parameter was different from the pointer type specified in the argument-type list or function definition.  
The parameter will be passed without change. Its value will be interpreted as a pointer within the called function.
- HI4023** *function : \_based pointer passed to unprototyped function : parameter number*  
(level 1) When in a near data model, only the offset portion of a **\_based** pointer is passed to an unprototyped function. If the function expects a far pointer, the resulting code will be wrong. In any data model, if the function is defined to take a **\_based** pointer with a different base, the resulting code may be unpredictable.  
If a prototype is used before the call, the call will be generated correctly.
- HI4024** *function : different types : parameter number*  
(level 1) The type of the given parameter in a function call did not agree with the type given in the argument-type list or function definition.  
The parameter will be passed without change. The function will interpret the parameter's type as the type expected by the function.
- HI4028** *parameter number declaration different*  
(level 1) The type of the given parameter did not agree with the corresponding type in the argument-type list or with the corresponding formal parameter.  
The original declaration was used.
- HI4030** *first parameter list longer than the second*  
(level 1) A function was declared more than once with different parameter lists.  
The first declaration was used.
- HI4031** *second parameter list is longer than the first*  
(level 1) A function was declared more than once with different parameter lists.  
The first declaration was used.
- HI4034** *sizeof returns 0*  
(level 1) The **sizeof** operator was applied to an operand that yielded a size of zero.  
This warning is informational.
- HI4040** *memory attribute on identifier ignored*  
(level 1) The **\_near**, **\_far**, **\_huge**, or **\_based** keyword has no effect in the declaration of the given identifier and is ignored.  
One cause of this warning is a huge array that is not declared globally. Declare huge arrays outside of main.

**HI4042**     *identifier* : has bad storage class**(level 1)**

The storage class specified for *identifier* cannot be used in this context.

The default storage class for this context was used in place of the illegal class:

- ☞ If *identifier* was a function, the compiler assumed extern class.
- ☞ If *identifier* was a formal parameter or local variable, the compiler assumed auto class.
- ☞ If *identifier* was a global variable, the compiler assumed the variable was declared with no storage class.

**HI4044**     **\_huge on *identifier* ignored, must be an array****(level 1)**

The compiler ignored the **\_huge** memory attribute on the given identifier. Only arrays may be declared with the **\_huge** memory attribute. On pointers, **\_huge** must be used as a modifier, not as a memory attribute.

**HI4047**     *operator* : different levels of indirection**(level 1)**

An expression involving the specified operator had inconsistent levels of indirection.

If both operands are of arithmetic type, or if both are not (such as array or pointer), then they are used without change, though the compiler may DS-extend one of the operands if one is far and one is near. If one is arithmetic and one is not, the arithmetic operator is converted to the type of the other operator.

For example, the following code causes this warning but is compiled without change:

```
char **p;
char *q;
p = q;    /* Warning */
```

**HI4048**     **array's declared subscripts different****(level 1)**

An expression involved pointers to arrays of different size.

The pointers were used without conversion.



**HI4049**     *operator* : indirection to different types**(level 1)**

The pointer expressions used with the given operator had different base types.

The expressions were used without conversion.

For example, the following code causes this warning:

```
struct ts1 *s1;
struct ts2 *s2;
s2 = s1;    /* Warning */
```

- HI4050**      *operator* : **different code attributes**  
(level 4)      The function-pointer expressions used with *operator* had different code attributes. The attribute involved is either **\_export** or **\_loadadds**.  
This is a warning and not an error, because **\_export** and **\_loadadds** affect only entry sequences and not calling conventions.
- HI4051**      **type conversion, possible loss of data**  
(level 2)      Two data items in an expression had different base types, causing the type of one item to be converted. During the conversion, a data item was truncated.
- HI4052**      **invalid numerical argument** *string*  
A numerical argument was expected instead of the given string.
- HI4053**      **at least one void operand**  
(level 1)      An expression with type void was used as an operand.  
The expression was evaluated using an undefined value for the void operand.
- HI4063**      *function* : **function too large for post-optimizer**  
(level 2)      Not enough space was available to optimize the given function.  
One of the following may be a solution:  
     Reccompile with fewer optimizations.  
     Divide the function into two or more smaller functions.
- HI4066**      **local symbol-table overflow - some local symbols may be missing in listings**  
(level 2)      The listing generator ran out of heap space for local variables, so the source listing may not contain symbol-table information for all local variables.
- HI4067**      **unexpected characters following** *directive* **directive - newline expected**  
(level 1)      Extra characters followed a preprocessor directive and were ignored. This warning appears only when compiling with the */Za* option.  
For example, the following code causes this warning:  

```
#endif      NO_EXT_KEYS
```

  
To remove the warning, compile with */Ze* or use comment delimiters:  

```
#endif      /* NO_EXT_KEYS */
```
- HI4071**      *function* : **no function prototype given**  
(level 2)      The given function was called before the compiler found the corresponding function prototype.  
The function will be called using the default rules for calling a function without a prototype.

**HI4072**     *function* : **no function prototype on \_fastcall function**

**(level 1)**     A **\_fastcall** function was called without first being prototyped.

Functions that are **\_fastcall** should be prototyped to guarantee that the registers assigned at each point of call are the same as the registers assumed when the function is defined. A function defined in the new ANSI style is a prototype.

A prototype must be added when this warning appears, unless the function takes no arguments or takes only arguments that cannot be passed in the general-purpose registers.

**HI4073**     **scoping too deep, deepest scoping merged when debugging**

**(level 1)**     Declarations appeared at a static nesting level greater than 13. As a result, all declarations beyond this level will seem to appear at the same level.

**HI4076**     *type* : **may be used on integral types only**

**(level 1)**     The signed or unsigned type modifier was used with a nonintegral type.

The given qualifier was ignored.

The following example causes this warning:

```
unsigned double x;
```

**HI4079**     **unexpected token** *token*

**(level 1)**     An unexpected separator token was found in the argument list of a pragma.

The remainder of the pragma was ignored.

**HI4082**     **expected an identifier, found** *token*

**(level 1)**     An identifier was missing from the argument list.

The remainder of the pragma was ignored.

**HI4083**     **expected '(', found** *token*

**(level 1)**     A left parenthesis, (, was missing from a pragma's argument list.

The pragma was ignored.

The following example causes this warning:

```
#pragma check_pointer on)
```

**HI4084**     **expected a pragma keyword, found** *token*

**(level 1)**     The *token* following **#pragma** was not recognized as a directive.

The pragma was ignored.

The following example causes this warning:

```
#pragma (on)
```

- HI4085**  
**(level 1)**      **expected [on | off]**  
The pragma expected an on or off parameter, but the specified parameter was unrecognized or missing.  
The pragma was ignored.
- HI4086**  
**(level 1)**      **expected [1 | 2 | 4]**  
The pragma expected a parameter of either 1, 2, or 4, but the specified parameter was unrecognized or missing.
- HI4087**  
**(level 1)**      ***function* : declared with void parameter list**  
The given function was declared as taking no parameters, but a call to the function specified actual parameters.  
The extra parameters were passed according to the calling convention used on the function.  
The following example causes this warning:  

```
int f1(void);  
f1(10);
```
- HI4088**  
**(level 1)**      ***function* : pointer mismatch : parameter *number*, parameter list *number***  
The argument passed to the given function had a different level of indirection from the given parameter in the function definition.  
The parameter will be passed without change. Its value will be interpreted as a pointer within the called function.
- HI4089**  
**(level 1)**      ***function* : different types : parameter *number*, parameter list *number***  
The argument passed to the given function did not have the same type as the given parameter in the function definition.  
The parameter will be passed without change. The function will interpret the parameter's type as the type expected by the function.
- HI4090**  
**(level 1)**      **different const/volatile qualifiers**  
A pointer to an item declared as const was assigned to a pointer that was not declared as const. As a result, the const item pointed to could be modified without being detected.  
The expression was compiled without modification.  
The following example causes this warning:  

```
const char *p = "abcde";  
int str(char *s);  
str(p);
```

**HI4091 no symbols were declared**

(level 2) The compiler detected an empty declaration, as in the following example:

```
int ;
```

The declaration was ignored.

**HI4092 untagged enum/struct/union declared no symbols**

(level 2) The compiler detected an empty declaration using an untagged structure, union, or enumerated variable.

The declaration was ignored.

For example, the following code causes this warning:

```
struct { . . . };
```

**HI4093 unescaped newline in character constant in inactive code**

(level 3) The constant expression of an **#if**, **#elif**, **#ifdef**, or **#ifndef** preprocessor directive evaluated to 0, making the code that follows inactive. Within that inactive code, a newline character appeared within a set of single or double quotation marks.

All text until the next double quotation mark was considered to be within a character constant.

**HI4095 expected ')', found *token***

(level 1) More than one argument was given for a pragma that can take only one argument. The compiler assumed the expected parenthesis and ignored the remainder of the line.

**HI4096 *attribute1* must be used with *attribute2***

(level 2) The use of *attribute2* requires the use of *attribute1*.

For example, using a variable number of arguments (...) requires that **\_cdecl** be used. Also, **\_interrupt** functions must be **\_far** and **\_cdecl**.

The compiler assumed *attribute1* for the function.

**HI4098 void function returning a value**

(level 1) A function declared with a void return type also returned a value.

A function was declared with a void return type but was defined as a value.

The compiler assumed the function returns a value of type int.

**HI4104 *identifier* : near data in same\_seg pragma, ignored**

(level 1) The given near variable was specified in a **same\_seg** pragma.

The *identifier* was ignored.

- HI4105**     *identifier* : **code modifiers only on function or pointer to function**  
(level 1)     The given identifier was declared with a code modifier that can be used only with a function or function pointer.  
The code modifier was ignored.
- HI4109**     **unexpected identifier** *identifier*  
(level 1)     The pragma contained an unexpected token.  
The pragma was ignored.
- HI4110**     **unexpected token** *int constant*  
(level 1)     The pragma contained an unexpected integer constant.  
The pragma was ignored.
- HI4111**     **unexpected token** *string*  
(level 1)     The pragma contained an unexpected string.  
The pragma was ignored.
- HI4112**     **macro name *name* is reserved, command ignored**  
(level 1)     The given command attempted to define or undefine the predefined macro *name* or the preprocessor operator *defined*. The given command is displayed as either **#define** or **#undef**, even if the attempt was made using command-line options.  
The command was ignored.
- HI4113**     **function parameter lists differed**  
(level 1)     A function pointer was assigned to a function pointer, but the parameter lists of the functions do not agree.  
The expression was compiled without modification.
- HI4114**     **same type qualifier used more than once**  
(level 1)     A type qualifier (const, volatile, signed, or unsigned) was used more than once in the same type.  
The second occurrence of the qualifier was ignored.
- HI4115**     **tag : type definition in formal parameter list**  
(level 1)     The given tag was used to define a **struct**, **union**, or **enum** in the formal parameter list of a function.  
The compiler assumed the definition was at the global level.
- HI4116**     **(no tag) : type definition in formal parameter list**  
(level 1)     A **struct**, **union**, or **enum** type with no tag was defined in the formal parameter list of a function.  
The compiler assumed the definition was at the global level.

**HI4119      different bases *name1* and *name2* specified**

(level 1)      The **\_based** pointers in the expression have different symbolic bases. There may be truncation or loss in the code generated.

**HI4120      \_based/unbased mismatch**

(level 1)      The expression contains a conversion between a **\_based** pointer and another pointer that is unbased. Some information may have been truncated.

This warning commonly occurs when a **\_based** pointer is passed to a function that accepts a near or far pointer.

**HI4123      different base expressions specified**

(level 1)      The expression contains a conversion between **\_based** pointers, but the base expressions of the **\_based** pointers are different. Some of the **\_based** conversions may be unexpected.

**HI4125      decimal digit terminates octal escape sequence**

(level 4)      An octal escape sequence in a character or string constant was terminated with a decimal digit.

The compiler evaluated the octal number without the decimal digit, and assumed the decimal digit was a character.

The following example causes this warning:

```
char array1[] = "\709";
```

If the digit 9 was intended as a character and was not a typing error, correct the example as follows:

```
char array[] = "\0709"; /* String containing "89" */
```

**HI4126      *flag* : unknown memory model flag**

(level 1)      The flag used with the /A option was not recognized and was ignored.

**HI4128      storage-class specifier after type**

(level 4)      A storage-class specifier (auto, extern, register, static) appears after a type in a declaration. The compiler assumed the storage class specifier occurred before the type.

New-style code places the storage-class specifier first.

**HI4129      *character* : unrecognized character escape sequence**

(level 4)      The *character* following a backslash in a character or string constant was not recognized as a valid escape sequence.

As a result, the backslash is ignored and not printed, and the character following the backslash is printed.

To print a single backslash (\), specify a double backslash (\\).



**HI4130** *operator : logical operation on address of string constant***(level 4)**

The operator was used with the address of a string literal. Unexpected code was generated.

For example, the following code causes this warning:

```
char *pc;
pc = "Hello";
if (pc == "Hello") ...
```

The if statement compares the value stored in the pointer pc to the address of the string "Hello" which is separately allocated each time it occurs in the code. It does not compare the string pointed to by pc with the string "Hello."

To compare strings, use the strcmp function.

**HI4131** *function : uses old-style declarator***(level 4)**

The function declaration or definition is not a prototype.

New-style function declarations are in prototype form.

 old style

```
int addrec( name, id )
char *name;
int id;
{ }
```

 new style

```
int addrec( char *name, int id )
{ }
```

**HI4132** *object : const object should be initialized***(level 4)**

The value of a const object cannot be changed, so the only way to give the const object a value is to initialize it.

It will not be possible to assign a value to *object*.

**HI4135** *conversion between different integral types***(level 3)**

Information was lost between two integral types.

For example, the following code causes this warning:

```
int intvar;
long longvar;
intvar = longvar;
```

If the information is merely interpreted differently, this warning is not given, as in the following example:

```
unsigned uintvar = intvar;
```

**HI4136      conversion between different floating types****(level 4)**

Information was lost or truncated between two floating types.

For example, the following code causes this warning:

```
double doublevar;  
float floatvar;  
floatvar = doublevar;
```

Note that unsuffixed floating-point constants have type double, so the following code causes this warning:

```
floatvar = 1.0;
```

If the floating-point constant should be treated as float type, use the F (or f) suffix on the constant to prevent the following warning:

```
floatvar = 1.0F;
```

**HI4138      \*/ found outside of comment****(level 1)**

The compiler found a closing comment delimiter (\*/) without a preceding opening delimiter. It assumed a space between the asterisk (\*) and the forward slash (/).

The following example causes this warning:

```
int /*comment*/ptr;
```

In this example, the compiler assumed a space before the first comment delimiter (/), and issued the warning but compiled the line normally. To remove the warning, insert the assumed space.

Usually, the cause of this warning is an attempt to nest comments.



To comment out sections of code that may contain comments, enclose the code in an **#if/#endif** block and set the controlling expression to zero, as in:

```
#if 0  
int my_variable;    /* Declaration currently not needed */  
#endif
```

- HI4139 (level 1)** *hexnumber* : **hex escape sequence is out of range**  
A hex escape sequence appearing in a character or string constant was too large to be converted to a character.  
If in a string constant, the compiler cast the low byte of the hexadecimal number to a char. If in a char constant, the compiler made the cast and then sign extended the result. If in a char constant and compiled with /J, the compiler cast the value to an unsigned char.  
For example, '\x1fff' is out of range for a character. Note that the following code causes this warning:  

```
printf("\x7Bell\n");
```

  
The number 7be is a legal hex number, but is too large for a character. To correct this example, use three hex digits:  

```
printf("\x007Bell\n");
```
- HI4186 (level 1)** **string too long - truncated to 40 characters**  
The string argument for a title or subtitle pragma exceeded the maximum allowable length and was truncated.
- HI4200 (level 1)** **local variable *identifier* used without having been initialized**  
A reference was made to a local variable that had not been assigned a value. As a result, the value of the variable is unpredictable.  
This warning is given only when compiling with global register allocation on (/Oe).
- HI4201 (level 3)** **local variable *identifier* may be used without having been initialized**  
A reference was made to a local variable that might not have been assigned a value. As a result, the value of the variable may be unpredictable.  
This warning is given only when compiling with the global register allocation on (/Oe).
- HI4202 (level 4)** **unreachable code**  
The flow of control can never reach the indicated line.  
This warning is given only when compiling with one of the global optimizations (/Oe, /Og, or /Ol).
- HI4203 (level 1)** *function* : **function too large for global optimizations**  
The named function was too large to fit in memory and be compiled with the selected optimization. The compiler did not perform any global optimizations (/Oe, /Og, or /Ol). Other /O optimizations, such as /Oa and /Oi, are still performed.  
One of the following may remove this warning:  
 Reccompile with fewer optimizations.  
 Divide the function into two or more smaller functions.

**HI4204      *function* : in-line assembler precludes global optimizations****(level 3)**

The use of in-line assembler in the named function prevented the specified global optimizations (/Oe, /Og, or /Ol) from being performed.

**HI4205      statement has no effect****(level 4)**

The indicated statement will have no effect on the program execution.

Some examples of statements with no effect:

```
1;  
a + 1;  
b == c;
```

**HI4209      comma operator within array index expression****(level 4)**

The value used as an index into an array was the last one of multiple expressions separated by the comma operator.

An array index legally may be the value of the last expression in a series of expressions separated by the comma operator. However, the intent may have been to use the expressions to specify multiple indexes into a multidimensional array.

For example, the following line, which causes this warning, is legal in C, and specifies the index c into array a:

```
a[b, c]
```

However, the following line uses both b and c as indexes into a two-dimensional array:

```
a[b][c]
```

**HI4300      insufficient memory to process debugging information****(level 2)**

The program was compiled with the /Zi option, but not enough memory was available to create the required debugging information.

One of the following may be a solution:

- 🔧 Split the current file into two or more files and compile them separately.
- 🔧 Remove other programs or drivers running in the system which could be consuming significant amounts of memory.

**HI4301      loss of debugging information caused by optimization****(level 2)**

Some optimizations, such as code motion, cause references to nested variables to be moved. The information about the level at which the variables are declared may be lost. As a result, all declarations will seem to be at nesting level 1.

- HI4323 (level 3) potential divide by 0**  
The second operand in a divide operation evaluated to zero at compile time, giving undefined results.  
The 0 operand may have been generated by the compiler, as in the following example:  

```
func1() { int i,j,k; i /= j && k; }
```
- HI4324 (level 3) potential mod by 0**  
The second operand in a remainder operation evaluated to zero at compile time, giving undefined results.
- HI4799 (level 1) unknown option *character* in *option***  
A command line option was specified that was not understood by H2INC, or the given character was not a valid letter for the option.  
For example, the following line:  

```
#pragma optimize("q", on)
```

  
causes the following warning:  

```
unknown option 'q' in '#pragma optimize'
```
- HI4800 (level 1) more than one memory model specified**  
There was more than one memory model given at the command line. The /AT, /AS, /AM, /AC, /AL, and /AH options specify the memory model.  
This error is caused by conflicting options specified at the command line and in the CL and H2INC environment variables.
- HI4801 (level 1) more than one target processor specified**  
There was more than one processor type given at the command line. The /G0, /G1, and /G2 options specify the processor type.  
This error is caused by conflicting options specified at the command line and in the CL and H2INC environment variables.
- HI4802 (level 1) ignoring invalid /Zp value *value***  
The alignment value specified to the /Zp option was not 1, 2, or 4. The default of 1 was assumed.
- HI4810 (level 2) untranslatable basic type size**  
H2INC could not translate the item to a MASM type.  
The C void type cannot be translated to a similar MASM type.
- HI4811 (level 1) static function prototype not translated**  
H2INC does not translate static items, as they are not visible outside the C source file.

- HI4812**      **static variable declaration not accepted with /Mn switch**  
(level 1)      H2INC does not translate static items, as they are not visible outside the C source file.
- HI4815**      ***string* : EQU string truncated to 254 characters**  
(level 1)      A **#define** statement exceeded 254 characters, the maximum length of a MASM **EQU** statement. The string was truncated.
- HI4816**      **ignoring \_fastcall function definition**  
(level 1)      H2INC does not translate function declarations or prototypes with the **\_fastcall** attribute. The **\_fastcall** calling convention cannot be used directly with MASM. See the documentation with your C compiler for details on **\_fastcall**.
- HI4820**      **ignoring function definition : *function()***  
(level 1)      H2INC translates header information only; it cannot convert program code. H2INC does not translate function bodies.

## HELPMAKE Error Messages

Microsoft Help File Maintenance Utility (HELPMAKE) generates the following error messages:

- ⚠ Fatal Errors (H1xxx) cause HELPMAKE to stop execution. No output file is produced.
- ⚠ Errors (H2xxx) do not prevent an output file from being produced, but parts of the conversion are not completed.
- ⚠ Warnings (H4xxx) do not prevent an output file from being produced, but problems may exist in the output.

## HELPMAKE Fatal Error Messages

- H1000**      **/A requires character**  
The /A option requires an application-specific control character.  
The correct form is:  
    /Ac  
where *c* is the control character.

**H1001     /E compression level must be numeric**

The /E option requires either no argument or a numeric value in the range 0–15.

The correct form is:

`/En`

where *n* specifies the amount of compression requested.

**H1002     multiple /O parameters specified**

Only one output file can be specified with the /O option.

**H1003     invalid /S file-type identifier**

The /S option was given an argument other than 1, 2, or 3.

The /S option requires specification of the type of input file. An invalid file-type identifier was specified.

The correct form is:

`/Sn`

where *n* specifies the format of the input file. Valid values are 1 (RTF), 2 (QuickHelp format), and 3 (minimally formatted ASCII).

**H1004     /S requires file-type identifier**

The /S option requires specification of the type of input file. There was no file-type identifier specified.

The correct form is:

`/Sn`

where *n* specifies the format of the input file. Valid values are 1 (RTF), 2 (QuickHelp format), and 3 (minimally formatted ASCII).

**H1005     /W fixed width invalid**

An invalid width was specified with the /W option. The valid range is 11–255.

**H1006     multiple /K parameters specified**

The option for specifying a keyword separator file, /K, was used more than once on the HELPMAKE command line.

Only one file containing separator characters can be specified.

**H1050     option invalid with /DS**

The /C, /L, and /O options for encoding are invalid with the /DS option for decoding.

**H1051     improper arguments for /D**

The /D option permits either no argument or an S or U argument. In addition, /D is invalid with the /C or /L option.

**H1052 encode requires /O option**

Database encoding was requested without a specified output-file name for the operation.

**H1053 compression level exceeds 15**

A value greater than 15 was specified with the /E option.

The /E option requires either no argument or a numeric value in the range 0–15.

The correct form is:

/En

where *n* specifies the amount of compression requested.

**H1097 no operation specified**

The HELPMAKE command line did not contain an option for encoding, decoding, or Help.

HELPMAKE requires the /E, /D, /H, or /? option.

**H1098 unrecognized option**

An unrecognized name followed the option indicator.

An option is specified by a forward slash (/) or a dash (–) and an option name.

**H1099 syntax error on command line**

HELPMAKE cannot interpret the command line.

**H1100 cannot open file**

One of the files specified on the HELPMAKE command line could not be found or created.

**H1101 error writing file**

The output file could not be written, probably because the disk is full.

**H1102 no input file specified**

In an encoding operation, no input Help text file was specified.

**H1103 no context strings found**

No context strings were found in the input stream during encoding.

Either the file is empty or the specified /S value does not correspond to the Help text formatting.

**H1104 no topic text found**

No topic text was found in the Help text file.

Either the file is empty or the specified /S value does not correspond to the Help text formatting.



**H1107 cannot overwrite input file**

The /DS option for splitting a concatenated Help file was specified, but the Help file contained a database with the same name as the Help file. It may be that the Help file is not a concatenated file and contains only one database, and the database has the same name as its physical Help file.

One of the following may be a solution:

- 🔧 Rename the Help file so that the filename does not match any of the database names.
- 🔧 Run HELPMAKE from a directory other than the one that holds the physical Help file. Since HELPMAKE creates the split files in the current directory, no filename conflict occurs.

**H1200 insufficient memory to allocate context buffer**

There was insufficient memory to run HELPMAKE.

HELPMAKE requires 256K free memory.

**H1201 insufficient memory to allocate utility buffer**

There was insufficient memory to run HELPMAKE.

HELPMAKE requires 256K free memory.

**H1250 not a valid compressed Help file**

The input file specified for a decompression operation is not a valid Help database file.

**H1251 cannot decompress locked Help file**

An attempt was made to decompress a Help database file that is locked.

A file is locked if the /L option is specified when the Help file is created.

**H1300 word too long in RTF processing**

A single word was longer than the specified format width (set by the /W option) or was found to be longer than 128 characters when HELPMAKE was reformatting a paragraph.

**H1302 attribute stack overflow processing RTF**

RTF attribute groups are nested too deeply. HELPMAKE supports a maximum of 50 levels of attribute-group nesting in RTF format.

**H1303      unknown RTF attribute**

An unknown RTF formatting command was found.

One of the following may have occurred:

- 🔍 A new RTF attribute was used. HELPMAKE recognizes a set of attributes that were current at the time this version of HELPMAKE was created. It interprets some of the attributes and knows to ignore the others. Any RTF attribute defined after HELPMAKE was created is not known by HELPMAKE and will cause this error.
- 🔍 The RTF file is corrupted.

**H1304      topic too large**

A topic exceeded the limit for the size of topics.

A single topic cannot exceed 64K.

**H1305      topic text without context string**

The source file contained topic text that was not preceded by a .context definition.

**H1900      internal virtual memory error**

This message indicates an internal HELPMAKE error.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**H1901      out of local memory**

This message indicates an internal HELPMAKE error.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**H1902      out of disk space for swap file**

The current drive or directory is full.

HELPMAKE uses a temporary swap file, written to the current drive and directory. The temporary file can grow to 1.5 times the size of the input files (for large Help files) and is not removed until the final Help file is completed.

**H1903      cannot open swap file**

HELPMAKE was unable to create its temporary swap file on the current drive and directory for one of the following reasons:

- 🔍 The current drive or directory is full.
- 🔍 The device cannot be written to.

**H1990 internal compression error**

This message indicates an internal HELPMAKE error.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

## HELMAKE Error Messages

**H2000 line too long, truncated**

A line exceeded the fixed width specified by the /W option or the default of 76 characters. HELPMAKE truncated the extra characters.

**H2001 duplicate context string**

A context string preceded more than one topic in a Help database. A context string can be associated with only one block of topic text.

**H2002 zero length hot spot**

A cross-reference was specified, but the word or anchored text associated with it was of zero length.

With no visible text to associate with the cross-reference, the hot spot will be inoperative. This error is issued as a warning and does not prevent the building of a Help file. However, some applications may not be able to use the resulting Help file correctly.

The following example will cause this error:

```
\a\vcross_reference\v
```

**H2003 unrecognized dot command**

A line in the source file contained a dot (.) in column 1, but it was not followed by a command recognized by HELPMAKE.

## HELMAKE Warning Messages

**H4000 keyword compression analysis table size exceeded  
no further new words will be analyzed**

The maximum number (16,000) of unique keywords has been encountered during keyword compression. This happens only in very large Help files. No further keywords will be included in the analysis. HELPMAKE continues to analyze how frequently words occur that it has already encountered.

**H4002 reference to undefined local context**

A string specifying a local context was used in a cross-reference but was not defined in a .context statement.

A local context begins with an at sign (@). Each local context that is used must be defined in a .context statement in one of the input files to HELPMAKE.

**H4003      negative left indent**

Topic text in an RTF file was formatted with a left indent to a position to the left of column 1. HELPMAKE deleted all text preceding column 1.

## IMPLIB Error Messages

Microsoft Import Library Manager (IMPLIB) generates the following error messages:

- ⓘ Fatal errors (IM16xx) cause IMPLIB to stop execution.
- ⓘ Errors (IM26xx) prevent IMPLIB from creating an import library.
- ⓘ Warnings (IM46xx) indicate possible problems in the output file being created.

## IMPLIB Fatal Error Messages

**IM1600      error writing to output file—*message***

IMPLIB could not create the import library for the given reason.

Probably the drive or directory where the import library is being created is full.

**IM1601      out of memory, near/far heap exhausted**

There was not enough room in memory for the heap needed by IMPLIB.

Increase the available memory. Some ways to do this include:

- ⓘ Remove TSR (terminate-and-stay-resident) programs.
- ⓘ Run IMPLIB outside of an NMAKE session.
- ⓘ Run IMPLIB outside of a shell.

**IM1602      syntax error in module-definition file**

IMPLIB could not understand the contents of a .DEF input file.

**IM1603      *filename* : cannot create file—*message***

IMPLIB could not create the given file for the given reason.

One of the following may be a cause:

- ⓘ The file already exists with a read-only attribute.
- ⓘ There is insufficient disk space to create the file.
- ⓘ The drive cannot be written to.

- IM1604** *filename : cannot open file—message*  
IMPLIB could not find the specified module-definition (.DEF) file or dynamic-link library (DLL) for the given reason.
- IM1605** **too many nested include files in module-definition file**  
A module-definition (.DEF) file contained an **INCLUDE** statement specifying a nested set of include files that exceeded the limit for nesting. The limit is 10 levels.
- IM1606** **missing or invalid include file name**  
A syntax error occurred in an **INCLUDE** statement in a module-definition (.DEF) file. One of the following may have occurred:
- A filename was not specified.
  - More than one filename was specified.
  - A long filename was specified without being enclosed in quotation marks or was enclosed in one single and one double quotation mark.
- IM1607** *extension : invalid extension for target library*  
The given extension was specified for the import library.  
An import library cannot be given a .DEF or .DLL extension.
- IM1608** **no .DLL or .DEF source file specified**  
No input file was specified on the IMPLIB command line.

## IMPLIB Error Messages

- IM2601** *symbol multiply defined*  
The given symbol was defined more than once in the input files.
- IM2602** **unexpected end of name table in DLL**  
A dynamic-link library (DLL) specified to IMPLIB was corrupted.
- IM2603** *filename : invalid .DLL file*  
IMPLIB did not recognize the given input file as a dynamic-link library (DLL).

## IMPLIB Warning Messages

- IM4600** **line *number* too long; truncated to 512 characters**  
The given line in the module-definition (.DEF) file exceeded the limit on line length. IMPLIB ignored text after the first 512 characters.
- IM4601** **unrecognized option *option*; option ignored**  
The given option was not a valid IMPLIB option. IMPLIB used the rest of the command line to try to build an import library.

## LIB Error Messages

This section lists error messages generated by the LIB utility.

Microsoft Library Manager (LIB) generates the following error messages:

- 🔊 Fatal errors (U1150 through U1203) cause LIB to stop execution.
- 🔊 Errors (U2152 through U2159) do not stop execution but prevent LIB from creating a library.
- 🔊 Warnings (U4150 through U4158) indicate possible problems in the library being created.

## LIB Fatal Error Messages

### **U1150      page size too small; use option /PAGE:n to increase it**

The page size of an input library was too small, indicating an invalid input .LIB file.

### **U1151      syntax error : illegal file specification**

A command operator was not followed by a module name or filename.

One possible cause of this error is an option specified with a dash (–) instead of a forward slash (/).

### **U1152      syntax error : option name missing**

A forward slash (/) appeared on the command line without an option name after it.

### **U1153      syntax error : option value missing**

The /PAGE option was given without a value following it.

### **U1154      unrecognized option**

An unrecognized name followed the option indicator (/).

An option is specified by a forward slash (/) and a name. The name can be specified by a legal abbreviation of the full name.

### **U1155      syntax error : illegal input**

A specified command did not follow correct LIB syntax.

### **U1156      syntax error**

A specified command did not follow correct LIB syntax.

**U1157 comma or newline missing**

A comma or newline character was expected in the command line but did not appear. One cause of this error is an incorrectly placed comma, as in the following command line:

```
LIB math.lib, -mod1 +mod2;
```

The line must be entered as follows:

```
LIB math.lib -mod1 +mod2;
```

**U1158 terminator missing**

The last line of the response file supplied to LIB did not end with a newline character.

**U1161 cannot rename old library**

LIB could not rename the old library with a .BAK extension because the .BAK version already existed with read-only protection.

Change the protection attribute on the .BAK file.

**U1162 cannot reopen library**

The old library could not be reopened after it was renamed with a .BAK extension.

One of the following may have occurred:

- Another process deleted the file or changed it to read-only.
- The floppy disk containing the file was removed.
- A hard-disk error occurred.

**U1163 error writing to cross-reference file**

The disk or root directory was full.

Delete or move files to make space.

**U1164 name length exceeds 255 characters**

A filename specified on the command line exceeded the LIB limit of 255 characters. Reduce the number of characters in the name.

**U1170 too many symbols**

The number of symbols in all object files and libraries exceeded the capacity of the dictionary created by LIB.

Create two or more smaller libraries.

**U1171 insufficient memory**

LIB did not have enough memory to run.

Remove any shells or resident programs, or add more memory.

**U1172      no more virtual memory**

The LIB session required more memory than the 1-megabyte limit imposed by LIB.

Try using the /NOE option or reducing the number of object modules.

**U1173      internal failure**

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**U1174      mark : not allocated**

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**U1175      free : not allocated**

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**U1180      write to extract file failed**

The disk or root directory was full.

Delete or move files to make space.

**U1181      write to library file failed**

The disk or root directory was full.

Delete or move files to make space.

**U1182      *filename* : cannot create extract file**

The disk or root directory was full, or the given extract file already existed with read-only protection.

Make space on the disk or change the protection of the extract file.

**U1183      cannot open response file**

The response file was not found.

**U1184      unexpected end-of-file on command input**

An end-of-file character was received prematurely in response to a prompt.

**U1185      cannot create new library**

The disk or root directory was full, or the library file already existed with read-only protection.

Make space on the disk or change the protection of the library file.

**U1186      error writing to new library**



The disk or root directory was full.

Delete or move files to make space.



- U1187 cannot open temporary file VM.TMP**  
The disk or root directory was full.  
Delete or move files to make space.
- U1188 insufficient disk space for temporary file**  
The library manager cannot write to the virtual memory.  
Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.
- U1189 cannot read from temporary file**  
The library manager cannot read the virtual memory.  
Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.
- U1190 interrupted by user**  
LIB was interrupted with either CTRL+C or CTRL+BREAK.
- U1191 *libraryname* : cannot write to read-only file**  
Operations cannot be performed on the given library because it is marked as a read-only file.  
Change the protection attribute on the library.
- U1200 *filename* : invalid library header**  
The input library file had an invalid format.  
Either it was not a library file or it was corrupted.
- U1203 *filename* : invalid object file near *location***  
The given file was not a valid object file or was corrupted at the given location.

## LIB Error Messages

- U2152 *filename* : cannot create listing**  
One of the following may have occurred:
-  The directory or disk was full.
  -  The cross-reference-listing file already existed with read-only protection.

**U2155**     *module* : **module not in library; ignored**

The specified module was not found in the input library.

One cause of this error is a filename or directory containing a hyphen or dash (–). LIB interprets the dash as the operator for the delete command. This error occurs if you install a Microsoft language product in a directory that has a dash in its path, such as C:\MS-C. The SETUP program for a language calls LIB to create combined libraries, but the dash in the command line passed to LIB causes the library-building session to fail.

Another possible cause of this error is an option specified with a dash (–) instead of a forward slash (/).

**U2157**     *filename* : **cannot access file**

LIB was unable to open the specified file, probably because the file did not exist.

Check the path and filename.

**U2158**     *library* : **invalid library header; file ignored**

The given library had an incorrect format and was not combined.

**U2159**     *filename* : **invalid format (number); file ignored**

The given file was not recognized as a XENIX archive and was not combined.

## LIB Warning Messages

**U4150**     *module* : **module redefinition ignored**

A module was specified with the add operator (+) to be added to a library, but a module having that name was already in the library.

One cause of this error is an incorrect specification of the replace operator (– +).

**U4151**     *symbol* : **symbol defined in module *module*; redefinition ignored**

The given symbol was defined in more than one module.

**U4153**     *option* : *value* : **page size invalid; ignored**

The argument specified with the /PAGE option was not valid for that option. The value must be an integer power of 2 between 16 and 32,768. LIB assumed an existing page size from a library that is being combined.

**U4155**     *modulename* : **module not in library**

The given module specified with a command operator does not exist in the library.

If the replacement command (– +) was specified, LIB added the file anyway. If the delete (–), copy (\*), or move (– \*) command was specified, LIB ignored the command.

**U4156**     *library* : output-library specification ignored

A new library was created because the filename specified in the *oldlibrary* field did not exist. However, a filename was also specified in the *newlibrary* field. LIB ignored the *newlibrary* specification.

For example, both of the following command lines cause this error if PROJECT.LIB does not already exist:

```
LIB project.lib +one.obj, new.lst, project.lib
LIB project.lib +one.obj, new.lst, new.lib
```

**U4157**     insufficient memory, extended dictionary not created

Insufficient memory prevented LIB from creating an extended dictionary.

The library is still valid, but the linker cannot take advantage of the extended dictionary to speed linking.

**U4158**     internal error, extended dictionary not created

An internal error prevented LIB from creating an extended dictionary.

The library is still valid, but the linker cannot take advantage of the extended dictionary to speed linking.

## LINK Error Messages

Microsoft Segmented-Executable Linker (LINK) generates the following error messages:

- ⚠ Fatal errors (L1xxx) cause LINK to stop execution.
- ⚠ Errors (L2xxx) do not stop execution but might prevent LINK from creating the main output file.
- ⚠ Warnings (L4xxx) indicate possible problems in the output file being created.

## LINK Fatal Error Messages

**L1001**     *option* : option name ambiguous

A unique option name did not appear after the option indicator.

An option is specified by a forward slash (/) and a name. The name can be specified by an abbreviation of the full name, but the abbreviation must be unambiguous.

For example, since many options begin with the letter **N**, the following command causes this error:

```
LINK /N main;
```

This error can also occur if the wrong version of the linker is being used. Check the directories in the PATH environment variable for other versions of LINK.EXE.

**L1003     /Q and /EXEPACK incompatible**

LINK cannot be given both the /Q option and the /EXEPACK option.

**L1004     value : invalid numeric value**

An incorrect value was specified with a LINK option.

For example, this error occurs if a nonnumeric string is specified with an option that requires a number.

**L1005     option : packing limit exceeds 64K**

The value specified with the /PACKC or /PACKD option exceeded the limit of 65,536 bytes.

**L1006     number : stack size exceeds 64K-2**

One of the following may have occurred:

- 🗑 The given value specified with the /STACK option exceeded the limit of 65,534 bytes.
- 🗑 A space appeared before or after the colon (:) between /STACK and the argument specified with it.

**L1007     /OVERLAYINTERRUPT : interrupt number exceeds 255**

An overlay interrupt number greater than 255 was specified with the /OV option value.

Check the *Microsoft MS-DOS Programmer's Reference* or other MS-DOS technical manual for information about interrupts.

**L1008     /SEGMENTS : segment limit set too high**

The value specified with the /SEG option exceeded 16,375.

**L1009     value : /CPARM : illegal value**

The value specified with the /CPARM option was not in the range 1–65,535.

- 
- L1020 no object files specified**  
The object-files field was empty.  
LINK requires the name of at least one object file.
- L1021 cannot nest response files**  
A response file was specified in a response file.
- L1022 response line too long**  
A line in a response file was longer than 255 characters.  
To extend a field to another line, put a plus sign (+) at the end of the current line.
- L1023 terminated by user**  
The LINK session was halted by CTRL+C or CTRL+BREAK.
- L1024 nested right parentheses**  
The parentheses for assigning overlays were specified incorrectly.
- L1025 nested left parentheses**  
The parentheses for assigning overlays were specified incorrectly.
- L1026 unmatched right parenthesis**  
The parentheses for assigning overlays were specified incorrectly.
- L1027 unmatched left parenthesis**  
The parentheses for assigning overlays were specified incorrectly.
- L1030 missing internal name**  
An **IMPORTS** statement specified an ordinal value but not an internal name for the routine or data item being imported.  
An item imported by ordinal must be given an internal name.
- L1031 module description redefined**  
The module-definition (.DEF) file contained more than one **DESCRIPTION** statement.
- L1032 module name redefined**  
The module-definition (.DEF) file contained more than one **NAME** or **LIBRARY** statement.
- L1033 input line too long; *number* characters allowed**  
The LINK command line cannot exceed the given number of characters.

**L1034      name truncated to *string***

A name specified either on the LINK command line or in a module-definition (.DEF) file exceeded 255 characters. The name was truncated to the given string.

This is a warning, not a fatal error. However, it indicates a serious problem. This message may be followed by another error as LINK tries to use the specified name. For example, if the string is a filename, LINK issues an error when it cannot open the file.

**L1035      syntax error in module-definition file**

A statement in the module-definition (.DEF) file was incorrect.

**L1040      too many exported entries**

The program exceeded the limit of 65,535 exported names.

**L1041      resident names table overflow**

The size of the resident names table exceeded 65,535 bytes.

An entry in the resident names table is made for each exported routine designated **RESIDENTNAME** and consists of the name plus three bytes of information. The first entry is the module name.

Reduce the number of exported routines or change some to nonresident status.

**L1042      nonresident names table overflow**

The size of the nonresident names table exceeded 65,535 bytes.

An entry in the nonresident names table is made for each exported routine not designated **RESIDENTNAME** and consists of the name plus three bytes of information. The first entry is the **DESCRIPTION** statement.

Reduce the number of exported routines or change some to resident status.

**L1043      relocation table overflow**

More than 32,768 long calls, long jumps, or other long pointers appeared in the program.

Replace long references with short references wherever possible.

**L1044      imported names table overflow**

The size of the imported names table exceeds 65,535 bytes.

An entry in the imported names table is made for each new name given in the **IMPORTS** section, including the module names, and consists of the name plus one byte.

Reduce the number of imports.

**L1045      too many TYPDEF records**

An object file contained more than 255 TYPDEF records.

TYPDEF records describe communal variables. (TYPDEF is an MS-DOS term. It is explained in the *Microsoft MS-DOS Programmer's Reference* and in other reference books on MS-DOS.)

This error appears only with programs created by the Microsoft FORTRAN Compiler or other compilers that support communal variables.

**L1046      too many external symbols in one module**

An object file specified more than 1023 external symbols.

Break the object file into smaller files.

**L1047      too many group, segment, and class names in one module**

An object file contained too many group, segment, and class names.

Reduce the number of groups, segments, or classes in the object file, or break the object file into smaller files.

**L1048      too many segments in one module**

An object file had more than 255 segments.

Either create fewer segments or break the object file into smaller files.

**L1049      too many segments**

The program contained more than the maximum number of segments.

The maximum number of segments is set with the /SEG option (in the range 1–16,384). If /SEG is not specified, the default is 128.

If this error occurs when linking a p-code program, recompile and use CL's /NQ option to combine the temporary p-code segments.

**L1050      too many groups in one module**

An object file contained more than 21 group definitions (GRPDEF).

Reduce the number of group definitions or split the module.

(Group definitions are explained in the *Microsoft MS-DOS Programmer's Reference* and in other reference books on MS-DOS.)

**L1051      too many groups**

The program defined more than 20 groups, not counting DGROUP.

Reduce the number of groups.

**L1052      too many libraries**

An attempt was made to link with more than 32 libraries.

Combine libraries, or use modules that require fewer libraries.

**L1053 out of memory for symbol table**

The program had more symbolic information than could fit in available memory. Symbolic information includes public, external, segment, group, class, and file names.

One of the following may be a solution:

- 🔧 Eliminate as many public symbols as possible.
- 🔧 Combine object files or segments.
- 🔧 Link from the command line instead of from a makefile or PWB.
- 🔧 Remove terminate-and-stay-resident programs or otherwise free some memory.

**L1054 requested segment limit too high**

LINK did not have enough memory to allocate tables describing the requested number of segments. The number of segments is the value specified with the /SEG option or the default of 128.

One of the following may be a solution:

- 🔧 Assemble with /c and link in a separate step.
- 🔧 Link again using the /SEG option to set fewer segments.
- 🔧 Remove terminate-and-stay-resident programs or otherwise free some memory.

**L1056 too many overlays**

The program defined more than 127 overlays.

**L1057 data record too large**

An LEDATA record in an object module contained more than 1024 bytes of data. This is a translator error. (LEDATA is an MS-DOS term explained in the *Microsoft MS-DOS Programmer's Reference* and in other MS-DOS reference books.)

Note which translator (compiler or assembler) produced the incorrect object module. Please report the circumstances of the error to Microsoft Corporation by following the instructions in the "Microsoft Support Services" section of the introduction to this book.

**L1063 out of memory for debugging information**

LINK ran out of memory for processing debugging information.

Reduce the amount of debugging information by compiling some object files with /Zd instead of /Zi or with neither option.



**L1064 out of memory—near/far heap exhausted**

LINK was not able to allocate enough memory for the given heap.

One of the following may be a solution:

- Reduce the size of code, data, and symbols in the program.
- If the program is a segmented executable file, put some code into a dynamic-link library.

**L1065 too many interoverlay calls  
use /DYNAMIC:nnn; current limit is number**

The program had more than the given limit of interoverlay calls.

The maximum number of interoverlay calls is set with the /DYNAMIC option (in the range 1–10,922). If /DYNAMIC is not specified, the default is 256.

To determine the setting needed by the program, run LINK with the /INFO option. The output gives the number of interoverlay calls that are generated and the current limit.

**L1066 size of overlaynumber overlay exceeds 64K**

The overlay represented by the given number exceeded the MOVE size limit of 65,535 bytes.

**L1067 memory allocation error**

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**L1070 segment : segment size exceeds 64K**

A single segment contained more than 65,536 bytes of code or data.

Try changing the memory model to use far code or data as appropriate. If the program is in C, use CL's /NT option or the **\_\_based** keyword (or its predecessor, the **alloc\_text** pragma) to build smaller segments.

**L1071 segment \_TEXT exceeds 64K–16**

The segment named \_TEXT grew larger than 65,520 bytes. This error is likely to occur only in small-model C programs, but it can occur when any program with a segment named \_TEXT is linked using the LINK /DOSSEG option.

Small-model C programs must reserve code addresses 0 and 1; this range is increased to 16 for alignment purposes.

Try compiling and linking using the medium or large model. If the program is in C, use CL's /NT option or the **\_\_based** keyword (or its predecessor, the **alloc\_text** pragma) to build smaller segments.

**L1072      common area exceeds 64K**

The program had more than 65,536 bytes of communal variables. This error occurs only with programs produced by the Microsoft FORTRAN optimizing compiler or other compilers that support communal variables.

**L1073      file-segment limit exceeded**

The number of physical or file segments exceeded the limit of 255 imposed by the Windows operating system for each application or dynamic-link library.

A file segment is created for each group definition, nonpacked logical segment, and set of packed segments.

Reduce the number of segments, or put more information into each segment. Use the /PACKC option or the /PACKD option or both.

**L1074      group : group exceeds 64K**

The given group exceeds the limit of 65,536 bytes.

Reduce the size of the group, or remove any unneeded segments from the group. Refer to the map file for a listing of segments.

**L1075      entry table exceeds 64K-1**

The entry table exceeded the limit of 65,535 bytes.

The table contains an entry for each exported routine and for each address that is the target of a far relocation, when **PROTMODE** is not enabled and the target segment is designated **MOVABLE**.

Declare **PROTMODE** if applicable, reduce the number of exported routines, or make some segments **FIXED** if possible.

**L1078      file-segment alignment too small**

The segment-alignment size specified with the /ALIGN option was too small.

**L1080      cannot open list file**

The disk or the root directory was full.

Delete or move files to make space.

**L1081      out of space for run file**

The disk or the root directory was full.

Delete or move files to make space.

**L1082      filename : stub file not found**

LINK could not open the file given in the **STUB** statement in the module-definition (.DEF) file.

The file must be in the current directory or in a directory specified by the PATH environment variable.

**L1083 cannot open run file**

One of the following may have occurred:

- The disk or the root directory was full.
- Another process opened or deleted the file.
- A read-only file existed with the same name.
- The floppy disk containing the file was removed.
- A hard-disk error occurred.

**L1084 cannot create temporary file**

One of the following may have occurred:

- The disk or the root directory was full.
- The directory specified in the TMP environment variable did not exist.

**L1085 cannot open temporary file—*message***

LINK could not open a temporary file for the given reason.

One of the following may have occurred:

- The disk or the root directory was full.
- The directory specified in the TMP environment variable did not exist.

**L1086 temporary file missing**

An internal error has occurred.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**L1087 unexpected end-of-file on temporary file**

A problem occurred with the temporary linker-output file.

One of the following may have occurred:

- The disk that holds the temporary file was removed.
- The disk or directory specified in the TMP environment variable was full.

**L1088 out of space for list file**

The disk or the root directory was full.

Delete or move files to make space.

- L1089**     *filename* : **cannot open response file**  
LINK could not find the given response file.  
One of the following may have occurred:
- 🔒 The response file does not exist.
  - 🔒 The name of the response file was incorrectly specified.
  - 🔒 An old version of LINK was used. Check your path. To see the version number of LINK, run LINK with the */?* option.
- L1090**     **cannot reopen list file**  
The original floppy disk was not replaced at the prompt.  
Restart the LINK session.
- L1091**     **unexpected end-of-file on library**  
The floppy disk containing the library was probably removed.  
Replace the disk containing the library and run LINK again.
- L1092**     **cannot open module-definition file**  
LINK could not find the specified module-definition (.DEF) file.  
Check that the name of the .DEF file is spelled correctly.
- L1093**     *filename* : **object file not found**  
LINK could not find the given object file.  
Check that the name of the object file is spelled correctly.
- L1094**     *filename* : **cannot open file for writing**  
LINK was unable to open the given file with write permission.  
Check the attributes for the file.
- L1095**     *filename* : **out of space for file**  
LINK ran out of disk space for the specified output file.  
Delete or move files to make space.
- L1096**     **unexpected end-of-file in response file**  
LINK encountered a problem while reading the response file.  
One of the following may be a cause:
- 🔒 The response file is corrupt.
  - 🔒 The file was deleted between reads.
- L1097**     **I/O error—message**  
LINK encountered the given input or output error.

- L1098 cannot open include file *filename*—message**  
LINK could not open the given include file for the given reason.  
An include file is specified in an **INCLUDE** statement in the module-definition (.DEF) file.
- L1100 stub .EXE file invalid**  
The file specified in the **STUB** statement in the module-definition (.DEF) file is not a valid MS-DOS executable file.
- L1101 invalid object module**  
LINK could not link one of the object files.  
Check that the correct version of LINK is being used.  
If the error persists after recompiling, note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.
- L1102 unexpected end-of-file**  
The given library or object file had an invalid format.
- L1103 attempt to access data outside segment bounds**  
A data record in an object file specified data extending beyond the end of a segment. This is a translator error.  
Note which translator (compiler or assembler) produced the incorrect object module and the circumstances in which it was produced. Please report the error to Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.
- L1104 *filename* : invalid library**  
The given file had an invalid format for a library.
- L1105 invalid object due to interrupted incremental compile**  
Delete the object file, recompile the program, and relink.
- L1106 unknown COMDAT allocation type for *symbol*; record ignored**  
This is a translator error. The given symbol is either a routine or a data item.  
Note which translator (compiler or assembler) produced the incorrect object module and the circumstances in which it was produced. Please report this error to Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**L1107      unknown COMDAT selection type for *symbol*; record ignored**

This is a translator error. The given symbol is either a routine or a data item.

Note which translator (compiler or assembler) produced the incorrect object module and the circumstances in which it was produced. Please report this error to Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**L1108      invalid format of debugging information**

This is a translator error.

Note which translator (compiler or assembler) produced the incorrect object module and the circumstances in which it was produced. Please report this error to Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**L1113      unresolved COMDEF; internal error**

This is a translator error.

Note which translator (compiler or assembler) produced the incorrect object module and the circumstances in which it was produced. Please report this error to Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**L1114      unresolved COMDAT *symbol*; internal error**

This is a translator error. The given symbol is either a routine or a data item.

Note which translator (compiler or assembler) produced the incorrect object module and the circumstances in which it was produced. Please report this error to Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**L1115      *option* : option incompatible with overlays**

The given option cannot be used when linking an overlaid program.

**L1117      unallocated COMDAT *symbol*; internal error**

This is a translator error. The given symbol is either a routine or a data item.

Note which translator (compiler or assembler) produced the incorrect object module and the circumstances in which it was produced. Please report this error to Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**L1123      *segment* : segment defined both 16-bit and 32-bit**

Define the segment as either 16-bit or 32-bit.

**L1127 far segment references not allowed with /TINY**

The /TINY option for producing a .COM file was used in a program that has a far segment reference.

Far segment references are not compatible with the .COM file format. High-level-language programs cause this error unless the language supports the tiny memory model. An assembly-language program that references a segment address also causes this error.

For example, the following causes this error:

```
mov     ax, seg mydata
```

**L1128 too many nested include files in module-definition file**

Nesting of **INCLUDE** statements in a module-definition (.DEF) file is limited to 10 levels.

**L1129 missing or invalid include file name**

The file specification in an **INCLUDE** statement in the module-definition (.DEF) file was missing or was not a valid filename.

## LINK Error Messages

**L2000 imported starting address**

The program starting address as specified in the **END** statement in an assembly-language file is an imported routine. This is not supported by the Windows operating system.

**L2002 fixup overflow at *number* in segment *segment***

This error message is followed by one of these strings:

🔧 target external *symbol*

🔧 frm seg *name1*, tgt seg *name2*, tgt offset *number*

A fixup overflow is an attempted reference to code or data that is impossible because the source location (where the reference is made “from”) and the target address (where the reference is made “to”) are too far apart. Usually the problem is corrected by examining the source location.

For information about frame and target segments, see the *Microsoft MS-DOS Programmer's Reference*.

**L2003 near reference to far target at *offset* in segment *segment*  
pos: *offset* target external *name***

The program issued a near call or jump to a label in a different segment.

This error occurs most often when specifically declaring an external procedure as near that should be declared as far.

This error can be caused by compiling a small-model C program with CL's /NT option.

**L2005      *fixup type unsupported at *number* in segment *segment****

A fixup type occurred that is not supported by LINK. This is probably a translator error. Note which translator (compiler or assembler) produced the incorrect object module and the circumstances in which it was produced. Please report this error to Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**L2010      *too many fixups in LIDATA record***

The number of far relocations (pointer- or base-type) in an LIDATA record exceeds the limit imposed by LINK.

The cause is usually a **DUP** statement in an assembly-language program. The limit is dynamic: a 1,024-byte buffer is shared by relocations and the contents of the LIDATA record. There are 8 bytes per relocation.

Reduce the number of far relocations in the **DUP** statement.

**L2011      *identifier : NEAR/HUGE conflict***

Conflicting **NEAR** and **HUGE** attributes were given for a communal variable. This error can occur only with programs produced by the Microsoft FORTRAN optimizing compiler or other compilers that support communal variables.

**L2012      *arrayname : array-element size mismatch***

A far communal array was declared with two or more different array-element sizes (for instance, an array was declared once as an array of characters and once as an array of real numbers). This error occurs only with the Microsoft FORTRAN optimizing compiler and any other compiler that supports far communal arrays.

**L2013      *LIDATA record too large***

An LIDATA record contained more than 512 bytes. This is probably a translator error.

Note which translator (compiler or assembler) produced the incorrect object module and the circumstances in which it was produced. Please report this error to Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**L2022      *entry (alias *internalname*) : export undefined***

The internal name of the given exported routine or data item is undefined.

**L2023      *entry (alias *internalname*) : export imported***

The internal name of the given exported routine or data item conflicts with the internal name of a previously imported routine or data item.



**L2024**     *symbol* : **special symbol already defined**

The program defined a symbol name already used by LINK for one of its own low-level symbols. For example, LINK generates special symbols used in overlay support and other operations.

Choose another name for the symbol to avoid conflict.

**L2025**     *symbol* : **symbol defined more than once**

The same symbol has been found in two different object files.

**L2026**     **entry ordinal** *number*, **name** *name* : **multiple definitions for same ordinal**

The given exported name with the given ordinal number conflicted with a different exported name previously assigned to the same ordinal. Only one name can be associated with a particular ordinal.

**L2027**     *name* : **ordinal too large for export**

The given exported name was assigned an ordinal that exceeded the limit of 65,535 (64K-1).

**L2028**     **automatic data segment plus heap exceed 64K**

The size of the sum of the following exceeds 64K:

- 🗄 Data declared in DGROUP
- 🗄 The size of the heap specified in the **HEAPSIZE** statement in the module-definition (.DEF) file
- 🗄 The size of the stack specified in either the /STACK option or the **STACKSIZE** statement in the .DEF file

Reduce near-data allocation, HEAPSIZE, or stack.

**L2029**     *symbol* : **unresolved external**

A symbol was declared to be external in one or more modules, but it was not publicly defined in any module or library.

The name of the unresolved external symbol is given, followed by a list of object modules that contain references to this symbol. This message and the list of object modules are written to the map file, if one exists.

One cause of this error is using the /NOI option for files that use case inconsistently in identifiers.

This error can also occur when a program compiled with C/C++ version 7.0 (or later) is linked using /NOD. The /NOD option tells LINK to ignore all default libraries named in object files. C/C++ 7.0 embeds in an object file both the name of the default run-time library and OLDNAMES.LIB. To avoid this error, either specify OLDNAMES.LIB in the *libraries* field or specify /NOD:*library* where *library* is the name of the default run-time library to be excluded from the search.

**L2030      starting address not code (use class 'CODE')**

The program starting address, as specified in the **END** statement of an .ASM file, should be in a code segment. Code segments are recognized if their class name ends in "CODE". This is an error in a segmented-executable file.

The error message can be disabled by including the **REALMODE** statement in the module-definition (.DEF) file.

**L2041      stack plus data exceed 64K**

If the total of near data and requested stack size exceeds 64K, the program will not run correctly. LINK checks for this condition only when /DOSSEG is enabled, which is the case in the library startup module for Microsoft language libraries.

For object modules compiled with the Microsoft C or FORTRAN optimizing compilers, recompile with the /Gt command-line option to set the data-size threshold to a smaller number.

This is a fatal LINK error.

**L2043      Quick library support module missing**

The required file QUICKLIB.OBJ was missing. QUICKLIB.OBJ must be linked in when creating a Quick library.

**L2044      *symbol* : symbol multiply defined, use /NOE**

LINK found what it interprets as a public-symbol redefinition, probably because a symbol defined in a library was redefined.

Relink with the /NOE option. If error L2025 results for the same symbol, then this is a genuine symbol-redefinition error.

**L2046      share attribute conflict—segment *segment* in group *group***

The given segment has a different sharing attribute than other segments that are assigned to the given group.

All segments assigned to a group must have the same attribute, either **SHARED** or **NONSHARED**. The attributes cannot be mixed.

**L2047      IOPL attribute conflict—segment *segment* in group *group***

The specified segment is a member of the specified group but has an **IOPL** attribute that is different from other segments in the group.

**L2048 Microsoft Overlay Manager module not found**

Overlays were designated, but an overlay manager was missing.

By default, the overlay manager is the Microsoft Overlay Virtual Environment (MOVE). This is provided in MOVE.LIB, which is a component library of the default combined libraries provided with Microsoft C/C++ version 7.0. The error occurs when LINK cannot find the **\_moveinit** routine.

If the /OLDOVERLAY option is specified, the overlay manager is the Microsoft Static Overlay Manager, which is also provided in the default combined libraries.

**L2050 USE16/USE32 attribute conflict—segment *segment* in group *group***

You cannot group 16-bit segments with 32-bit segments.

**L2052 *symbol* : unresolved external; possible calling convention mismatch**

A symbol was declared to be external in one or more modules, but LINK could not find it publicly defined in any module or library.

The name of the unresolved external symbol is given, followed by a list of object modules that contain references to this symbol. The error message and the list of object modules are written to the map file, if one exists.

This error occurs in a C-language program when a prototype for an externally defined function is omitted and the program is compiled with CL's /Gr option. The calling convention for **\_\_fastcall** does not match the assumptions that are made when a prototype is not included for an external function.

Either include a prototype for the function, or compile without the /Gr option.

**L2057 duplicate of *function* with different size found; record ignored**

An inconsistent class definition was found.

Check the include files and recompile.

**L2058 different duplicate of *function* found; record ignored**

An inconsistent class definition was found.

Check the include files and recompile.

**L2060 size of data block associated with *symbol* (16-bit segment) exceeds 64K**

A class had too many virtual functions. The given symbol is the v-table for the class, in the form of a decorated name.

**L2061 no space for data block associated with *function* inside *segment* *segment***

The given function was allocated to the given segment, but the segment was full.

**L2062** continuation of COMDAT *function* has conflicting attributes; record ignored

This is a translator error.

Note which translator (compiler or assembler) produced the incorrect object module and the circumstances in which it was produced. Please report this error to Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

**L2063** *function* is allocated in undefined segment

The given function was allocated to a nonexistent segment.

**L2064** starting address not in the root overlay

The segment or object file that contains the starting address for the program was placed into an overlay.

The starting address in a C-language program is provided by the **main** function.

## LINK Warning Messages

**L4000** segment displacement included near *offset in segment segment*

This is the warning generated by the /W option.

**L4001** frame-relative fixup, frame ignored near *offset in segment segment*

A reference was made relative to a segment or group that is different from the target segment of the reference.

For example, if **\_i d1** is defined in segment **\_TEXT**, the instruction call **DGROUP: \_i d1** produces this warning. The frame **DGROUP** is ignored, so LINK treats the call as if it were call **\_TEXT: \_i d1**.

**L4002** frame-relative absolute fixup near *offset in segment segment*

A reference was made relative to a segment or group that was different from the target segment of the reference, and both segments are absolute (defined with AT).

LINK assumed that the executable file will be run only with MS-DOS.

**L4004** possible fixup overflow at *offset in segment segment*

A near call or jump was made to another segment that was not a member of the same group as the segment from which the call or jump was made.

This can cause an incorrect real-mode address calculation when the distance between the paragraph address (frame number) of the segment group and the target segment is greater than 64K, even though the distance between the segment where the call or jump was actually made and the target segment is less than 64K.

**L4010** invalid alignment specification

The number specified in the /ALIGN option must be a power of 2 in the range 2–32,768.

- L4011     /PACKC value exceeding 64K–36 unreliable**  
The packing limit specified with the /PACKC option was in the range 65,501–65,536 bytes. Code segments with a size in this range are unreliable on some versions of the 80286 processor.
- L4012     /HIGH disables /EXEPACK**  
The /HIGH and /EXEPACK options cannot be used at the same time.
- L4013     *option* : option ignored for segmented executable file**  
The given option is not allowed for segmented-executable programs.
- L4014     *option* : option ignored for DOS executable file**  
The given option is not allowed for MS-DOS programs.
- L4015     /CO disables /DSALLOC**  
The /CO and /DSALLOC options cannot be used at the same time.
- L4016     /CO disables /EXEPACK**  
The /CO and /EXEPACK options cannot be used at the same time.
- L4017     *option* : unrecognized option name; option ignored**  
The given option was not a valid LINK option. LINK ignored the option specification.  
One of the following may be a cause:
- An obsolete option was specified to the current version of LINK. For example, the /INCR option is obsolete in LINK version 5.30. The current options are described in the manual and in online Help. To see a list of options, run LINK with the /? option.
  - An old version of LINK was used. Check your path. To see the version number of LINK, run LINK with the /? option.
  - The name was incorrectly specified. For example, the option specification /NODEFAULTLIBSEARCH is an invalid abbreviation of the /NODEFAULTLIBRARYSEARCH option. Option names can be shortened by removing letters only from the end of the name.
- L4018     missing or unrecognized application type; option *option* ignored**  
The /PM option accepts only the keywords **PM**, **VIO**, and **NOVIO**.
- L4020     *segment* : code-segment size exceeds 64K–36**  
Code segments that are 65,501 through 65,536 bytes in length may be unreliable on some versions of the 80286 processor.

**L4021      no stack segment**

The program did not contain a stack segment defined with the STACK combine type.

Normally, every program should have a stack segment with the combine type specified as STACK. You can ignore this message if you have a specific reason for not defining a stack or for defining one without the STACK combine type. Linking with versions of LINK earlier than version 2.40 might cause this message since these linkers search libraries only once.

**L4022      *group1, group2* : groups overlap**

The given groups overlap. Since a group is assigned to a physical segment, groups cannot overlap in segmented-executable files.

Reorganize segments and group definitions so the groups do not overlap. Refer to the map file.

**L4023      *entry(internalname)* : export internal name conflict**

The internal name of the given exported function or data item conflicted with the internal name of a previous import definition or export definition.

**L4024      *name* : multiple definitions for export name**

The given name was exported more than once, an action that is not allowed.

**L4025      *module.name.entry(internalname)* : import internal name conflict**

The internal name of the given imported function or data item conflicted with the internal name of a previous export or import. (The given entry is either a name or an ordinal number.)

**L4026      *module.name.entry(internalname)* : self-imported**

The given function or data item was imported from the module being linked. This error can occur if a module tries to import a function or data item from itself or from another source (such as a DLL) that has the same name.

**L4027      *name* : multiple definitions for import internal name**

The given internal name was imported more than once. Previous import definitions are ignored.

**L4028      *segment* : segment already defined**

The given segment was defined more than once in a SEGMENTS statement of the module-definition (.DEF) file.

**L4029      *segment* : DGROUP segment converted to type DATA**

The given logical segment in the group DGROUP was defined as a code segment.

DGROUP cannot contain code segments because LINK always considers DGROUP to be a data segment. The name DGROUP is predefined as the automatic (or default) data segment.

LINK converted the named segment to type DATA.

**L4030      *segment* : segment attributes changed to conform with automatic data segment**

The given logical segment in the group DGROUP was given sharing attributes (**SHARED/NONSHARED**) that differed from the automatic data attributes as declared by the DATA instance specification (**SINGLE/MULTIPLE**). The attributes are converted to conform to those of DGROUP.

The name DGROUP is predefined as the automatic (or default) data segment. DGROUP cannot contain code segments because LINK always considers DGROUP to be a data segment.

**L4031      *segment* : segment declared in more than one group**

A segment was declared to be a member of two different groups.

**L4032      *segment* : code-group size exceeds 64K-36**

The given code group has a size in the range 65,501–65,536 bytes, a size that is unreliable on some versions of the 80286 processor.

**L4033      *first segment in mixed group group is a USE32 segment***

A 16-bit segment must be first in a group created with both USE16 and USE32 segments.

LINK continued to build the executable file, but the resulting file may not run correctly.

**L4034      *more than 1024 overlay segments; extra put in root***

The limit on the number of segments that can go into overlays is 1024. Segments starting with the 1025th segment are assigned to the permanently resident portion of the program (the root).

**L4036      *no automatic data segment***

The application did not define a group named DGROUP.

DGROUP has special meaning to LINK, which uses it to identify the automatic (or default) data segment used by the operating system. Most segmented-executable applications require DGROUP.

This warning will not be issued if **DATA NONE** is declared or if the executable file is a dynamic-link library.

**L4037      *group* : both USE16 and USE32 segments in group; assuming USE32**

The given group was allocated contributions from both 16-bit segments and 32-bit segments.

**L4038      program has no starting address**

The segmented-executable application had no starting address. A missing starting address will usually cause the program to fail.

High-level languages automatically specify a starting address. In a C-language program, this is provided by the **main** function.

If you are writing an assembly-language program, specify a starting address with the **END** statement.

MS-DOS programs and dynamic-link libraries should never receive this message, regardless of whether they have starting addresses.

**L4040      stack size ignored for /TINY**

LINK ignores stack size if the /TINY option is used and if the stack segment has been defined in front of the code segment.

**L4042      cannot open old version**

The file specified in the **OLD** statement in the module-definition (.DEF) file could not be opened.

**L4043      old version not segmented executable format**

The file specified in the **OLD** statement in the module-definition (.DEF) file was not a valid segmented-executable file.

**L4045      name of output file is *filename***

LINK used the given filename for the output file.

If the output filename is specified without an extension, LINK assumes the default extension .EXE. Creating a Quick library, DLL, or .COM file forces LINK to use a different extension. In the following cases, if either .EXE or no extension is specified, LINK assumes the appropriate extension:

/TINY option: .COM

/Q option: .QLB

**LIBRARY** statement: .DLL

**L4050      file not suitable for /EXEPACK; relink without**

The size of the packed load image plus packing overhead was larger than it would be for the unpacked load image. There is no advantage to packing this program.

Remove /EXEPACK from the LINK command line. In PWB, clear the Pack EXE File check box in the Additional Debug/Release Options dialog box under Link Options.

This warning also occurs if the name specified in the **LIBRARY** statement in the module-definition (.DEF) file does not match the name specified in the *exefile* field.



**L4051**     *filename* : **cannot find library**

LINK could not find the given library file.

One of the following may be a cause:

- The specified file does not exist. Enter the name or full path specification of a library file.
- The LIB environment variable is not set correctly. Check for incorrect directory specifications, mistyping, or a space, semicolon, or hidden character at the end of the line.
- An earlier version of LINK is being run. Check the path environment variable and delete or rename earlier linkers.

**L4053**     **VM.TMP : illegal filename; ignored**

VM.TMP appeared as an object-file name.

Rename the file and rerun LINK.

**L4054**     *filename* : **cannot find file**

LINK could not find the specified file.

Enter a new filename, a new path specification, or both.

**L4055**     **start address not equal to 0x100 for /TINY**

The starting address for a .COM file must be 100 hexadecimal.

Put the following line of assembly source code in front of the code segment:

ORG 100h

**L4056**     **/EXEPACK valid only for OS/2 and real-mode DOS; ignored**

The /EXEPACK option is incompatible with Windows-based programs.

**L4057**     **stack specified for DLL; ignored**

A stack was specified for a dynamic-link library (DLL). Either the /STACK option was used on the command line or the **STACKSIZE** statement was used in the module-definition (.DEF) file. LINK ignored the specification and did not create a stack.

A DLL does not have a stack.

**L4058**     **ignoring alias for already defined symbol** *symbol*

The specified symbol was redefined in the program. However, it is an identifier from a C run-time library that has an alias to a new name in OLDNAMES.LIB. LINK ignored the alias for the symbol.

This warning appears only when the /INFO option is specified.

- L4067**     **changing default resolution for weak external *symbol* from *oldresolution* to *newresolution***  
LINK found conflicting default resolutions for a weak external. It ignored the first resolution and used the second.
- L4068**     **ignoring stack size greater than 64K**  
A stack was defined with an invalid size. LINK assumed 64K.
- L4069**     **filename truncated to *filename***  
A filename specification exceeded the length allowed. LINK assumed the given filename.
- L4070**     **too many public symbols for sorting**  
LINK uses the stack and all available memory in the near heap to sort public symbols for the /MAP option. This warning is issued if the number of public symbols exceeds the space available for them. In addition, the symbols are not sorted in the map file but are listed in an arbitrary order.
- L4076**     **no segments defined**  
There was no code in the program.  
This warning can occur if the file contains only resources.
- L4077**     **symbol *function* not defined; ordered allocation ignored**  
The given function was specified in a **FUNCTIONS** statement in the module-definition (.DEF) file, but the function was not defined.
- L4079**     **symbol *function* already defined for ordered allocation; duplicate ignored**  
The given function was specified twice in **FUNCTIONS** statements in the module-definition (.DEF) file.
- L4080**     **changing substitute name for alias *symbol* from *oldalias* to *newalias***  
LINK found conflicting alias names. It ignored the first alias and used the second.
- L4081**     **cannot execute *program arguments*—*message***  
LINK could not run the given program (with the given arguments) for the given reason.
- L4082**     **changing overlay assignment for segment *segment* from *oldnumber* to *newnumber***  
The given segment was assigned to two overlays, represented by *oldnumber* and *newnumber*. LINK assumed the *newnumber* overlay.  
  
Probably a command-line overlay specification with parentheses conflicted with an overlay specification in the module-definition (.DEF) file.

- L4083**     **changing overlay assignment for symbol *symbol* from *oldnumber* to *newnumber***  
The given symbol was assigned to two overlays, represented by *oldnumber* and *newnumber*. LINK assumed the *newnumber* overlay.  
Probably a command-line overlay specification with parentheses conflicted with an overlay specification in the module-definition (.DEF) file.
- L4084**     ***option* : argument missing; option ignored**  
The given option requires an argument, but none was specified.  
For example, the following option specification causes this error:  
      /ONERROR
- L4085**     ***option* : argument invalid; assuming *argument***  
The given option was specified with a numeric argument that was out of range for the option. LINK assumed the given argument.  
For example, the option specification /DYNAMIC:11000 causes the following error:  
      /DYNAMIC:11000 : argument invalid; assuming 10922
- L4086**     **/r not first on command line; ignored**  
This message appears if the /r option is not specified before other LINK options. /r must be the first option specified or it will be ignored.

## ML Error Messages

### ML Fatal Errors

**A1000 cannot open file: *filename***

The assembler was unable to open a source, include, or output file.

One of the following may be a cause:

- The file does not exist.
- The file is in use by another process.
- The filename is not valid.
- A read-only file with the output filename already exists.
- Not enough file handles exist. In MS-DOS, increase the number of file handles by changing the FILES setting in CONFIG.SYS to allow a larger number of open files. FILES=50 is the recommended setting.
- The current drive is full.
- The current directory is the root and is full.
- The device cannot be written to.
- The drive is not ready.

**A1001 I/O error closing file**

The operating system returned an error when the assembler attempted to close a file.

This error can be caused by having a corrupt file system or by removing a disk before the file could be closed.

**A1002 I/O error writing file**

The assembler was unable to write to an output file.

One of the following may be a cause:

- The current drive is full.
- The current directory is the root and is full.
- The device cannot be written to.
- The drive is not ready.

**A1003 I/O error reading file**

The assembler encountered an error when trying to read a file.

One of the following may be a cause:

- The disk has a bad sector.
- The file-access attribute is set to prevent reading.
- The drive is not ready.

**A1005 assembler limit : macro parameter name table full**

Too many parameters, locals, or macro labels were defined for a macro. There was no more room in the macro name table.

Define shorter or fewer names, or remove unnecessary macros.

**A1006 invalid command-line option: *option***

ML did not recognize the given parameter as an option.

This error is generally caused when there is a syntax error on the command line.

**A1007 nesting level too deep**

The assembler reached its nesting limit. The limit is 20 levels except where noted otherwise.

One of the following was nested too deeply:

- 🔒 A high-level directive such as **.IF**, **.REPEAT**, or **.WHILE**
- 🔒 A structure definition
- 🔒 A conditional-assembly directive
- 🔒 A procedure definition
- 🔒 A **PUSHCONTEXT** directive (the limit is 10).
- 🔒 A segment definition
- 🔒 An include file
- 🔒 A macro

**A1008 unmatched macro nesting**

Either a macro was not terminated before the end of the file, or the terminating directive **ENDM** was found outside of a macro block.

One cause of this error is omission of the dot before **.REPEAT** or **.WHILE**.

**A1009 line too long**

A line in a source file exceeded the limit of 512 characters.

If multiple physical lines are concatenated with the line-continuation character ( \ ), the resulting logical line is still limited to 512 characters.

**A1010 unmatched block nesting :**

A block beginning did not have a matching end, or a block end did not have a matching beginning. One of the following may be involved:

- 🔒 A high-level directive such as **.IF**, **.REPEAT**, or **.WHILE**
- 🔒 A conditional-assembly directive such as **IF**, **REPEAT**, or **WHILE**
- 🔒 A structure or union definition
- 🔒 A procedure definition
- 🔒 A segment definition
- 🔒 A **POPCONTEXT** directive
- 🔒 A conditional-assembly directive, such as an **ELSE**, **ELSEIF**, or **ENDIF** without a matching **IF**

**A1011 directive must be in control block**

The assembler found a high-level directive where one was not expected. One of the following directives was found:

- ▼ **.ELSE** without **.IF**
- ▼ **.ENDIF** without **.IF**
- 🔒 **.ENDW** without **.WHILE**
- 🔒 **.UNTIL**[[CXZ]] without **.REPEAT**
- 🔒 **.CONTINUE** without **.WHILE** or **.REPEAT**
- 🔒 **.BREAK** without **.WHILE** or **.REPEAT**
- 🔒 **.ELSE** following **.ELSE**

**A1012 error count exceeds 100; stopping assembly**

The number of nonfatal errors exceeded the assembler limit of 100.

Nonfatal errors are in the range A2xxx. When warnings are treated as errors they are included in the count. Warnings are considered errors if you use the /Wx command-line option, or if you set the Warnings Treated as Errors option in the Macro Assembler Global Options dialog box of PWB.

**A1013 invalid numerical command-line argument : *number***

The argument specified with an option was not a number or was an invalid number.

**A1014 too many arguments**

There was insufficient memory to hold all of the command-line arguments.

This error usually occurs while expanding input filename wildcards (\* and ?). To eliminate this error, assemble multiple source files separately.

**A1015 statement too complex**

The assembler ran out of stack space while trying to parse the specified statement.

One or more of the following changes may eliminate this error:

- Break the statement into several shorter statements.
- Reorganize the statement to reduce the amount of parenthetical nesting.
- If the statement is part of a macro, break the macro into several shorter macros.

**A1017 missing source filename**

ML could not find a file to assemble or pass to the linker.

This error is generated when you give ML command-line options without specifying a filename to act upon. To assemble files that do not have a .ASM extension, use the /Ta command-line option.

This error can also be generated by invoking ML with no parameters if the ML environment variable contains command-line options.

**A1901 Internal Assembler Error  
Contact Microsoft Product Support Services**

The MASM driver called ML.EXE, which generated a system error.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

## ML Nonfatal Errors

**A2000 memory operand not allowed in context**

A memory operand was given to an instruction that cannot take a memory operand.

**A2001 immediate operand not allowed**

A constant or memory offset was given to an instruction that cannot take an immediate operand.

**A2002 cannot have more than one ELSE clause per IF block**

The assembler found an **ELSE** directive after an existing **ELSE** directive in a conditional-assembly block (**IF** block).

Only one **ELSE** can be used in an **IF** block. An **IF** block begins with an **IF**, **IFE**, **IFB**, **IFNB**, **IFDEF**, **IFNDEF**, **IFDIF**, or **IFIDN** directive. There can be several **ELSEIF** statements in an **IF** block.

One cause of this error is omission of an **ENDIF** statement from a nested **IF** block.

**A2003 extra characters after statement**

A directive was followed by unexpected characters.

**A2004      symbol type conflict : *identifier***

The **EXTERNDEF** or **LABEL** directive was used on a variable, symbol, data structure, or label that was defined in the same module but with a different type.

**A2005      symbol redefinition : *identifier***

The given nonredefinable symbol was defined in two places.

**A2006      undefined symbol : *identifier***

An attempt was made to use a symbol that was not defined.

One of the following may have occurred:

- 🔍 A symbol was not defined.
- 🔍 A field was not a member of the specified structure.
- 🔍 A symbol was defined in an include file that was not included.
- 🔍 An external symbol was used without an **EXTERN** or **EXTERNDEF** directive.
- 🔍 A symbol name was misspelled.
- 🔍 A local code label was referenced outside of its scope.

**A2007      non-benign record redefinition**

A RECORD definition conflicted with a previous definition.

One of the following occurred:

- 🔍 There were different numbers of fields.
- 🔍 There were different numbers of bits in a field.
- 🔍 There was a different label.
- 🔍 There were different initializers.

**A2008      syntax error :**

A token at the current location caused a syntax error.

One of the following may have occurred:

- 🔍 A dot prefix was added to or omitted from a directive.
- 🔍 A reserved word (such as **C** or **SIZE**) was used as an identifier.
- 🔍 An instruction was used that was not available with the current processor or coprocessor selection.
- 🔍 A comparison run-time operator (such as **==**) was used in a conditional assembly statement instead of a relational operator (such as **EQ**).
- 🔍 An instruction or directive was given too few operands.
- 🔍 An obsolete directive was used.



**A2009 syntax error in expression**

An expression on the current line contained a syntax error. This error message may also be a side-effect of a preceding program error.

**A2010 invalid type expression**

The operand to **THIS** or **PTR** was not a valid type expression.

**A2011 distance invalid for word size of current segment**

A procedure definition or a code label defined with **LABEL** specified an address size that was incompatible with the current segment size.

One of the following occurred:

- A **NEAR16** or **FAR16** procedure was defined in a 32-bit segment.
- A **NEAR32** or **FAR32** procedure was defined in a 16-bit segment.
- A code label defined with **LABEL** specified **FAR16** or **NEAR16** in a 32-bit segment.
- A code label defined with **LABEL** specified **FAR32** or **NEAR32** in a 16-bit segment.

**A2012 PROC, MACRO, or macro repeat directive must precede LOCAL**

A **LOCAL** directive must be immediately preceded by a **MACRO**, **PROC**, macro repeat directive (such as **REPEAT**, **WHILE**, or **FOR**), or another **LOCAL** directive.

**A2013 .MODEL must precede this directive**

A simplified segment directive or a **.STARTUP** or **.EXIT** directive was not preceded by a **.MODEL** directive.

A **.MODEL** directive must specify the model defaults before a simplified segment directive, or a **.STARTUP** or **.EXIT** directive may be used.

**A2014 cannot define as public or external : identifier**

Only labels, procedures, and numeric equates can be made public or external using **PUBLIC**, **EXTERN**, or **EXTERNDEF**. Local code labels cannot be made public.

**A2015 segment attributes cannot change : attribute**

A segment was reopened with different attributes than it was opened with originally.

When a **SEGMENT** directive opens a previously defined segment, the newly opened segment inherits the attributes the segment was defined with.

**A2016 expression expected**

The assembler expected an expression at the current location but found one of the following:

- A unary operator without an operand
- A binary operator without two operands
- An empty pair of parentheses, ( ), or brackets, [ ]

**A2017 operator expected**

An expression operator was expected at the current location.

One possible cause of this error is a missing comma between expressions in an expression list.

**A2018 invalid use of external symbol : *identifier***

An attempt was made to compare the given external symbol using a relational operator.

The comparison cannot be made because the value or address of an external symbol is not known at assembly time.

**A2019 operand must be RECORD type or field**

The operand following the **WIDTH** or **MASK** operator was not valid.

The **WIDTH** operator takes an operand that is the name of a field or a record. The **MASK** operator takes an operand that is the name of a field or a record type.

**A2020 identifier not a record : *identifier***

A record type was expected at the current location.

**A2021 record constants cannot span line breaks**

A record constant must be defined on one physical line. A line ended in the middle of the definition of a record constant.

**A2022 instruction operands must be the same size**

The operands to an instruction did not have the same size.

**A2023 instruction operand must have size**

At least one of the operands to an instruction must have a known size.

**A2024 invalid operand size for instruction**

The size of an operand was not valid.

**A2025 operands must be in same segment**

Relocatable operands used with a relational or minus operator were not located in the same segment.

**A2026 constant expected**

The assembler expected a constant expression at the current location. A constant expression is a numeric expression that can be resolved at assembly time.

**A2027 operand must be a memory expression**

The right operand of a **PTR** expression was not a memory expression.

When the left operand of the **PTR** operator is a structure or union type, the right operand must be a memory expression.

**A2028      expression must be a code address**

An expression evaluating to a code address was expected.

One of the following occurred:

- ▼ **SHORT** was not followed by a code address.
- ▼ **NEAR PTR** or **FAR PTR** was applied to something that was not a code address.

**A2029      multiple base registers not allowed**

An attempt was made to combine two base registers in a memory expression.

For example, the following expressions cause this error:

```
[bx+bp]
[bx][bp]
```

In another example, given the following definition:

```
id1 proc arg1: byte
```

either of the following lines causes this error:

```
mov al, [bx].arg1
lea ax, arg1[bx]
```

**A2030      multiple index registers not allowed**

An attempt was made to combine two index registers in a memory expression.

For example, the following expressions cause this error:

```
[si+di]
[di][si]
```

**A2031      must be index or base register**

An attempt was made to use a register that was not a base or index register in a memory expression.

For example, the following expressions cause this error:

```
[ax]
[bl]
```

**A2032    invalid use of register**

An attempt was made to use a register that was not valid for the intended use.

One of the following occurred:

- ▼ **OFFSET** was applied to a register. (**OFFSET** can be applied to a register under the **M510** option.)
- A special 386 register was used in an invalid context.
- A register was cast with **PTR** to a type of invalid size.
- A register was specified as the right operand of a segment override operator (:).
- A register was specified as the right operand of a binary minus operator (-).
- An attempt was made to multiply registers using the \* operator.
- Brackets ([ ]) were missing around a register that was added to something.

**A2033    invalid INVOKE argument : argument *number***

The **INVOKE** directive was passed a special 386 register, or a register pair containing a byte register or special 386 register. These registers are illegal with **INVOKE**.

**A2034    must be in segment block**

One of the following was found outside of a segment block:

- An instruction
- A label definition
- A **THIS** operator
- A \$ operator
- A procedure definition
- An **ALIGN** directive
- An **ORG** directive

**A2035    DUP too complex**

A declaration using the **DUP** operator resulted in a data structure with an internal representation that was too large.

**A2036    too many initial values for structure: *structure***

The given structure was defined with more initializers than the number of fields in the type declaration of the structure.

**A2037    statement not allowed inside structure definition**

A structure definition contained an invalid statement.

A structure cannot contain instructions, labels, procedures, control-flow directives, **.STARTUP**, or **.EXIT**.

- A2038 missing operand for macro operator**  
The assembler found the end of a macro's parameter list immediately after the ! or % operator.
- A2039 line too long**  
A source-file line exceeded the limit of 512 characters.  
If multiple physical lines are concatenated with the line-continuation character ( \ ), the resulting logical line is still limited to 512 characters.
- A2040 segment register not allowed in context**  
A segment register was specified for an instruction that cannot take a segment register.
- A2041 string or text literal too long**  
A string or text literal, or a macro function return value, exceeded the limit of 255 characters.
- A2042 statement too complex**  
A statement was too complex for the assembler to parse.  
Reduce either the number of tokens or the number of forward-referenced identifiers.
- A2043 identifier too long**  
An identifier exceeded the limit of 247 characters.
- A2044 invalid character in file**  
The source file contained a character outside a comment, string, or literal that was not recognized as an operator or other legal character.
- A2045 missing angle bracket or brace in literal**  
An unmatched angle bracket (either < or >) or brace (either { or }) was found in a literal constant or an initializer.  
One of the following occurred:
- 🔊 A pair of angle brackets or braces was not complete.
  - 🔊 An angle bracket was intended to be literal, but it was not preceded by an exclamation point (!) to indicate a literal character.

**A2046 missing single or double quotation mark in string**

An unmatched quotation mark (either ' or ") was found in a string.

One of the following may have occurred:

- A pair of quotation marks around a string was not complete.
- A pair of quotation marks around a string was formed of one single and one double quotation mark.
- A single or double quotation mark was intended to be literal, but the surrounding quotation marks were the same kind as the literal one.

**A2047 empty (null) string**

A string consisted of a delimiting pair of quotation marks and no characters within.

For a string to be valid, it must contain 1–255 characters.

**A2048 nondigit in number**

A number contained a character that was not in the set of characters used by the current radix (base).

This error can occur if a B or D radix specifier is used when the default radix is one that includes that letter as a valid digit.

**A2049 syntax error in floating-point constant**

A floating-point constant contained an invalid character.

**A2050 real or BCD number not allowed**

A floating-point (real) number or binary coded decimal (BCD) constant was used other than as a data initializer.

One of the following occurred:

- A real number or a BCD was used in an expression.
- A real number was used to initialize a directive other than **DWORD**, **QWORD**, or **TBYTE**.
- A BCD was used to initialize a directive other than **TBYTE**.

**A2051 text item required**

A literal constant or text macro was expected.

One of the following was expected:

- A literal constant, which is text enclosed in < >
- A text macro name
- A macro function call
- A % followed by a constant expression

- A2052      forced error**  
The conditional-error directive **.ERR** or **.ERR1** was used to generate this error.
- A2053      forced error : value equal to 0**  
The conditional-error directive **.ERRE** was used to generate this error.
- A2054      forced error : value not equal to 0**  
The conditional-error directive **.ERRNZ** was used to generate this error.
- A2055      forced error : symbol not defined**  
The conditional-error directive **.ERRNDEF** was used to generate this error.
- A2056      forced error : symbol defined**  
The conditional-error directive **.ERRDEF** was used to generate this error.
- A2057      forced error : string blank**  
The conditional-error directive **.ERRB** was used to generate this error.
- A2058      forced error : string not blank**  
The conditional-error directive **.ERRNB** was used to generate this error.
- A2059      forced error : strings equal**  
The conditional-error directive **.ERRIDN** or **.ERRIDNI** was used to generate this error.
- A2060      forced error : strings not equal**  
The conditional-error directive **.ERRDIF** or **.ERRDIFI** was used to generate this error.
- A2061      `[[ELSE]]IF2/.ERR2` not allowed : single-pass assembler**  
A directive for a two-pass assembler was found.  
  
The Microsoft Macro Assembler (MASM) is a one-pass assembler. MASM does not accept the **IF2**, **ELSEIF2**, and **.ERR2** directives.  
  
This error also occurs if an **ELSE** directive follows an **IF1** directive.
- A2062      expression too complex for .UNTILCXZ**  
An expression used in the condition that follows **.UNTILCXZ** was too complex.  
  
The **.UNTILCXZ** directive can take only one expression, which can contain only **==** or **!=**. It cannot take other comparison operators or more complex expressions using operators like **||**.
- A2063      can ALIGN only to power of 2 : *expression***  
The expression specified with the **ALIGN** directive was invalid.  
  
The **ALIGN** expression must be a power of 2 between 2 and 256, and must be less than or equal to the alignment of the current segment, structure, or union.

- A2064 structure alignment must be 1, 2, or 4**  
The alignment specified in a structure definition was invalid.
- A2065 expected : *token***  
The assembler expected the given token.
- A2066 incompatible CPU mode and segment size**  
An attempt was made to open a segment with a **USE16**, **USE32**, or **FLAT** attribute that was not compatible with the specified CPU, or to change to a 16-bit CPU while in a 32-bit segment.  
  
The **USE32** and **FLAT** attributes must be preceded by one of the following processor directives: **.386**, **.386C**, **.386P**, **.486**, or **.486P**.
- A2067 LOCK must be followed by a memory operation**  
The **LOCK** prefix preceded an invalid instruction. No instruction can take the **LOCK** prefix unless one of its operands is a memory expression.
- A2068 instruction prefix not allowed**  
One of the prefixes **REP**, **REPE**, **REPNE**, or **LOCK** preceded an instruction for which it was not valid.
- A2069 no operands allowed for this instruction**  
One or more operands were specified with an instruction that takes no operands.
- A2070 invalid instruction operands**  
One or more operands were not valid for the instruction they were specified with.
- A2071 initializer too large for specified size**  
An initializer value was too large for the data area it was initializing.
- A2072 cannot access symbol in given segment or group: *identifier***  
The given identifier cannot be addressed from the segment or group specified.
- A2073 operands have different frames**  
Two operands in an expression were in different frames.  
  
Subtraction of pointers requires the pointers to be in the same frame. Subtraction of two expressions that have different effective frames is not allowed. An effective frame is calculated from the segment, group, or segment register.
- A2074 cannot access label through segment registers**  
An attempt was made to access a label through a segment register that was not assumed to its segment or group.



**A2075      jump destination too far [: by 'n' bytes]**

The destination specified with a jump instruction was too far from the instruction.

One of the following may be a solution:

- 🔧 Enable the **LJMP** option.
- 🔧 Remove the **SHORT** operator. If **SHORT** has forced a jump that is too far, *n* is the number of bytes out of range.
- 🔧 Rearrange code so that the jump is no longer out of range.

**A2076      jump destination must specify a label**

A direct jump's destination must be relative to a code label.

**A2077      instruction does not allow NEAR indirect addressing**

A conditional jump or loop cannot take a memory operand. It must be given a relative address or label.

**A2078      instruction does not allow FAR indirect addressing**

A conditional jump or loop cannot take a memory operand. It must be given a relative address or label.

**A2079      instruction does not allow FAR direct addressing**

A conditional jump or loop cannot be to a different segment or group.

**A2080      jump distance not possible in current CPU mode**

A distance was specified with a jump instruction that was incompatible with the current processor mode.

For example, 48-bit jumps require **.386** or above.

**A2081      missing operand after unary operator**

An operator required an operand, but no operand followed.

**A2082      cannot mix 16- and 32-bit registers**

An address expression contained both 16- and 32-bit registers.

For example, the following expression causes this error:

[bx+edi ]

**A2083      invalid scale value**

A register scale was specified that was not 1, 2, 4, or 8.

**A2084      constant value too large**

A constant was specified that was too big for the context in which it was used.

**A2085 instruction or register not accepted in current CPU mode**

An attempt was made to use an instruction, register, or keyword that was not valid for the current processor mode.

For example, 32-bit registers require **.386** or above. Control registers such as CR0 require privileged mode **.386P** or above. This error will also be generated for the **NEAR32**, **FAR32**, and **FLAT** keywords, which require **.386** or above.

**A2086 reserved word expected**

One or more items in the list specified with a **NOKEYWORD** option were not recognized as reserved words.

**A2087 instruction form requires 80386/486**

An instruction was used that was not compatible with the current processor mode.

One of the following processor directives must precede the instruction: **.386**, **.386C**, **.386P**, **.486**, or **.486P**.

**A2088 END directive required at end of file**

The assembler reached the end of the main source file and did not find an **.END** directive.

**A2089 too many bits in RECORD : *identifier***

One of the following occurred:

- Too many bits were defined for the given record field.
- Too many total bits were defined for the given record.

The size limit for a record or a field in a record is 16 bits when doing 16-bit arithmetic or 32 bits when doing 32-bit arithmetic.

**A2090 positive value expected**

A positive value was not found in one of the following situations:

- The starting position specified for **SUBSTR** or **@SubStr**
- The number of data objects specified for **COMM**
- The element size specified for **COMM**

**A2091 index value past end of string**

An index value exceeded the length of the string it referred to when used with **INSTR**, **SUBSTR**, **@InStr**, or **@SubStr**.

**A2092 count must be positive or zero**

The operand specified to the **SUBSTR** directive, **@SubStr** macro function, **SHL** operator, **SHR** operator, or **DUP** operator was negative.

**A2093 count value too large**

The length argument specified for **SUBSTR** or **@SubStr** exceeded the length of the specified string.

**A2094 operand must be relocatable**

An operand was not relative to a label.

One of the following occurred:

- An operand specified with the **END** directive was not relative to a label.
- An operand to the **SEG** operator was not relative to a label.
- The right operand to the minus operator was relative to a label, but the left operand was not.
- The operands to a relational operator were either not both integer constants or not both memory operands. Relational operators can take operands that are both addresses or both non-addresses but not one of each.

**A2095 constant or relocatable label expected**

The operand specified must be a constant expression or a memory offset.

**A2096 segment, group, or segment register expected**

A segment or group was expected but was not found.

One of the following occurred:

- The left operand specified with the segment override operator (:) was not a segment register (CS, DS, SS, ES, FS, or GS), group name, segment name, or segment expression.
- The **ASSUME** directive was given a segment register without a valid segment address, segment register, group, or the special **FLAT** group.

**A2097 segment expected : *identifier***

The **GROUP** directive was given an identifier that was not a defined segment.

**A2098 invalid operand for OFFSET**

The expression following the **OFFSET** operator must be a memory expression or an immediate expression.

**A2099 invalid use of external absolute**

An attempt was made to subtract a constant defined in another module from an expression.

You can avoid this error by placing constants in include files rather than making them external.

**A2100 segment or group not allowed**

An attempt was made to use a segment or group in a way that was not valid. Segments or groups cannot be added.

**A2101 cannot add two relocatable labels**

An attempt was made to add two expressions that were both relative to a label.

**A2102 cannot add memory expression and code label**

An attempt was made to add a code label to a memory expression.

**A2103 segment exceeds 64K limit**

A 16-bit segment exceeded the size limit of 64K.

**A2104 invalid type for data declaration : *type***

The given type was not valid for a data declaration.

**A2105 HIGH and LOW require immediate operands**

The operand specified with either the **HIGH** or the **LOW** operator was not an immediate expression.

**A2107 cannot have implicit far jump or call to near label**

An attempt was made to make an implicit far jump or call to a near label in another segment.

**A2108 use of register assumed to ERROR**

An attempt was made to use a register that had been assumed to ERROR with the **ASSUME** directive.

**A2109 only white space or comment can follow backslash**

A character other than a semicolon (;) or a white-space character (spaces or TAB characters) was found after a line-continuation character (\).

**A2110 COMMENT delimiter expected**

A delimiter character was not specified for a **COMMENT** directive.

The delimiter character is specified by the first character that is not white space (spaces or TAB characters) after the **COMMENT** directive. The comment consists of all text following the delimiter until the end of the line containing the next appearance of the delimiter.

**A2111 conflicting parameter definition**

A procedure defined with the **PROC** directive did not match its prototype as defined with the **PROTO** directive.

**A2112 PROC and prototype calling conventions conflict**

A procedure was defined in a prototype (using the **PROTO**, **EXTERNDEF**, or **EXTERN** directive), but the calling convention did not match the corresponding **PROC** directive.

- A2113     invalid radix tag**  
The specified radix was not a number in the range 2–16.
- A2114     INVOKE argument type mismatch : argument *number***  
The type of the arguments passed using the **INVOKE** directive did not match the type of the parameters in the prototype of the procedure being invoked.
- A2115     invalid coprocessor register**  
The coprocessor index specified was negative or greater than 7.
- A2116     instructions and initialized data not allowed in AT segments**  
An instruction or initialized data was found in a segment defined with the **AT** attribute.  
Data in **AT** segments must be declared with the **?** initializer.
- A2117     /AT option requires TINY memory model**  
The **/AT** option was specified on the assembler command line, but the program being assembled did not specify the **TINY** memory model with the **.MODEL** directive.  
This error is only generated for modules that specify a start address or use the **.STARTUP** directive.
- A2118     cannot have segment address references with TINY model**  
An attempt was made to reference a segment in a **TINY** model program.  
All **TINY** model code and data must be accessed with **NEAR** addresses.
- A2119     language type must be specified**  
A procedure definition or prototype was not given a language type.  
A language type must be declared in each procedure definition or prototype if a default language type is not specified. A default language type is set using either the **.MODEL** directive, **OPTION LANG**, or the ML command-line options **/Gc** or **/Gd**.
- A2120     PROLOGUE must be macro function**  
The identifier specified with the **OPTION PROLOGUE** directive was not recognized as a defined macro function.  
The user-defined prologue must be a macro function that returns the number of bytes needed for local variables and any extra space needed for the macro function.
- A2121     EPILOGUE must be macro procedure**  
The identifier specified with the **OPTION EPILOGUE** directive was not recognized as a defined macro procedure.  
The user-defined epilogue macro cannot return a value.

- A2122     alternate identifier not allowed with EXTERNDEF**  
An attempt was made to specify an alternate identifier with an **EXTERNDEF** directive.  
You can specify an optional alternate identifier with the **EXTERN** directive but not with **EXTERNDEF**.
- A2123     text macro nesting level too deep**  
A text macro was nested too deeply. The nesting limit for text macros is 40.
- A2125     missing macro argument**  
A required argument to **@InStr**, **@SubStr**, or a user-defined macro was not specified.
- A2126     EXITM used inconsistently**  
The **EXITM** directive was used both with and without a return value in the same macro.  
A macro procedure returns a value; a macro function does not.
- A2127     macro function argument list too long**  
There were too many characters in a macro function's argument list. This error applies also to a prologue macro function called implicitly by the **PROC** directive.
- A2129     VARARG parameter must be last parameter**  
A parameter other than the last one was given the **VARARG** attribute.  
The **:VARARG** specification can be applied only to the last parameter in a parameter list for macro and procedure definitions and prototypes. You cannot use multiple **:VARARG** specifications in a macro.
- A2130     VARARG parameter not allowed with LOCAL**  
An attempt was made to specify **:VARARG** as the type in a procedure's **LOCAL** declaration.
- A2131     VARARG parameter requires C calling convention**  
A **VARARG** parameter was specified in a procedure definition or prototype, but the **C**, **SYSCALL**, or **STDCALL** calling convention was not specified.
- A2132     ORG needs a constant or local offset**  
The expression specified with the **ORG** directive was not valid.  
**ORG** requires an immediate expression with no reference to an external label or to a label outside the current segment.
- A2133     register value overwritten by INVOKE**  
A register was passed as an argument to a procedure, but the code generated by **INVOKE** to pass other arguments destroyed the contents of the register.  
The **AX**, **AL**, **AH**, **EAX**, **DX**, **DL**, **DH**, and **EDX** registers may be used by the assembler to perform data conversion.  
Use a different register.

- A2134 structure too large to pass with INVOKE : argument *number***  
An attempt was made with **INVOKE** to pass a structure that exceeded 255 bytes.  
Pass structures by reference if they are larger than 255 bytes.
- A2136 too many arguments to INVOKE**  
The number of arguments passed using the **INVOKE** directive exceeded the number of parameters in the prototype for the procedure being invoked.
- A2137 too few arguments to INVOKE**  
The number of arguments passed using the **INVOKE** directive was fewer than the number of required parameters specified in the prototype for the procedure being invoked.
- A2138 invalid data initializer**  
The initializer list for a data definition was invalid.  
This error can be caused by using the R radix override with too few digits.
- A2140 RET operand too large**  
The operand specified to **RET**, **RETN**, or **RETF** exceeded two bytes.
- A2141 too many operands to instruction**  
Too many operands were specified with a string control instruction.
- A2142 cannot have more than one .ELSE clause per .IF block**  
The assembler found more than one **.ELSE** clause within the current **.IF** block.  
Use **.ELSEIF** for all but the last block.
- A2143 expected data label**  
The **LENGTHOF**, **SIZEOF**, **LENGTH**, or **SIZE** operator was applied to a non-data label, or the **SIZEOF** or **SIZE** operator was applied to a type.
- A2144 cannot nest procedures**  
An attempt was made to nest a procedure containing a parameter, local variable, **USES** clause, or a statement that generated a new segment or group.
- A2145 EXPORT must be FAR : *procedure***  
The given procedure was given **EXPORT** visibility and **NEAR** distance.  
All **EXPORT** procedures must be **FAR**. The default visibility may have been set with the **OPTION PROC:EXPORT** statement or the **SMALL** or **COMPACT** memory models.
- A2146 procedure declared with two visibility attributes : *procedure***  
The given procedure was given conflicting visibilities.  
A procedure was declared with two different visibilities (**PUBLIC**, **PRIVATE**, or **EXPORT**). The **PROC** and **PROTO** statements for a procedure must have the same visibility.

- A2147      macro label not defined : *macrolabel***  
The given macro label was not found.  
A macro label is defined with *:macrolabel*.
- A2148      invalid symbol type in expression : *identifier***  
The given identifier was used in an expression in which it was not valid.  
For example, a macro procedure name is not allowed in an expression.
- A2149      byte register cannot be first operand**  
A byte register was specified to an instruction that cannot take it as the first operand.
- A2150      word register cannot be first operand**  
A word register was specified to an instruction that cannot take it as the first operand.
- A2151      special register cannot be first operand**  
A special register was specified to an instruction that cannot take it as the first operand.
- A2152      coprocessor register cannot be first operand**  
A coprocessor (stack) register was specified to an instruction that cannot take it as the first operand.
- A2153      cannot change size of expression computations**  
An attempt was made to set the expression word size when the size had been already set using the **EXPR16**, **EXPR32**, **SEGMENT:USE32**, or **SEGMENT:FLAT** option or the **.386** or higher processor selection directive.
- A2154      syntax error in control-flow directive**  
The condition for a control-flow directive (such as **.IF** or **.WHILE**) contained a syntax error.
- A2155      cannot use 16-bit register with a 32-bit address**  
An attempt was made to mix 16-bit and 32-bit offsets in an expression.  
Use a 32-bit register with a symbol defined in a 32-bit segment.  
For example, if **i d1** is defined in a 32-bit segment, the following causes this error:  
**i d1[bx]**
- A2156      constant value out of range**  
An invalid value was specified for the **PAGE** directive.  
The first parameter of the **PAGE** directive can be either 0 or a value in the range 10–255.  
The second parameter of the **PAGE** directive can be either 0 or a value in the range 60–255.



**A2157 missing right parenthesis**

A right parenthesis, ), was missing from a macro function call.

Be sure that parentheses are in pairs if nested.

**A2158 type is wrong size for register**

An attempt was made to assume a general-purpose register to a type with a different size than the register.

For example, the following pair of statements causes this error:

```
ASSUME bx:far ptr byte ; far pointer is 4 or 6 bytes
ASSUME al:word          ; al is a byte reg, cannot hold word
```

**A2159 structure cannot be instantiated**

An attempt was made to create an instance of a structure when there were no fields or data defined in the structure definition or when **ORG** was used in the structure definition.

**A2160 non-benign structure redefinition : label incorrect**

A label given in a structure redefinition either did not exist in the original definition or was out of order in the redefinition.

**A2161 non-benign structure redefinition : too few labels**

Not enough members were defined in a structure redefinition.

**A2162 OLDSTRUCT/NOOLDSTRUCT state cannot be changed**

Once the **OLDSTRUCTS** or **NOOLDSTRUCTS** option has been specified and a structure has been defined, the structure scoping cannot be altered or respecified in the same module.

**A2163 non-benign structure redefinition : incorrect initializers**

A **STRUCT** or **UNION** was redefined with a different initializer value.

When structures and unions are defined more than once, the definitions must be identical. This error can be caused by using a variable as an initializer and having the value of the variable change between definitions.

**A2164 non-benign structure redefinition : too few initializers**

A **STRUCT** or **UNION** was redefined with too few initializers.

When structures and unions are defined more than once, the definitions must be identical.

**A2165 non-benign structure redefinition : label has incorrect offset**

The offset of a label in a redefined **STRUCT** or **UNION** differs from the original definition.

When structures and unions are defined more than once, the definitions must be identical. This error can be caused by a missing member or by a member that has a different size than in its original definition.

**A2166 structure field expected**

The righthand side of a dot operator (.) is not a structure field.

This error may occur with some code acceptable to previous versions of the assembler. To enable the old behavior, use **OPTION OLDSTRUCTS**, which is automatically enabled by **OPTION M510** or the /Zm command-line option.

**A2167 unexpected literal found in expression**

A literal was found where an expression was expected.

One of the following may have occurred:

- 🔍 A literal was used as an initializer
- 🔍 A record tag was omitted from a record constant

**A2169 divide by zero in expression**

An expression contains a divisor whose value is equal to zero.

Check that the syntax of the expression is correct and that the divisor (whether constant or variable) is correctly initialized.

**A2170 directive must appear inside a macro**

A **GOTO** or **EXITM** directive was found outside the body of a macro.

**A2171 cannot expand macro function**

A syntax error prevented the assembler from expanding the macro function.

**A2172 too few bits in RECORD**

There was an attempt to define a record field of 0 bits.

**A2173 macro function cannot redefine itself**

There was an attempt to define a macro function inside the body of a macro function with the same name. This error can also occur when a member of a chain of macros attempts to redefine a previous member of the chain.

**A2175 invalid qualified type**

An identifier was encountered in a qualified type that was not a type, structure, record, union, or prototype.

**A2176 floating point initializer on an integer variable**

An attempt was made to use a floating-point initializer with **DWORD**, **QWORD**, or **TBYTE**. Only integer initializers are allowed.

**A2177 nested structure improperly initialized**

The nested structure initialization could not be resolved.

This error can be caused by using different beginning and ending delimiters in a nested structure initialization.

**A2178 invalid use of FLAT**

There was an ambiguous reference to **FLAT** as a group.

This error is generated when there is a reference to **FLAT** instead of a **FLAT** subgroup. For example,

```
mov    ax, FLAT           ; Generates A2178
mov    ax, SEG FLAT: _data ; Correct
```

**A2179 structure improperly initialized**

There was an error in a structure initializer.

One of the following occurred:

- The initializer is not a valid expression.
- The initializer is an invalid **DUP** statement.

**A2180 improper list initialization**

In a structure, there was an attempt to initialize a list of items with a value or list of values of the wrong size.

**A2181 initializer must be a string or single item**

There was an attempt to initialize a structure element with something other than a single item or string.

This error can be caused by omitting braces ( { } ) around an initializer.

**A2182 initializer must be a single item**

There was an attempt to initialize a structure element with something other than a single item.

This error can be caused by omitting braces ( { } ) around an initializer.

**A2183 initializer must be a single byte**

There was an attempt to initialize a structure element of byte size with something other than a single byte.

**A2184 improper use of list initializer**

The assembler did not expect an opening brace ( { ) at this point.

**A2185 improper literal initialization**

A literal structure initializer was not properly delimited.

This error can be caused by missing angle brackets ( < > ) or braces ( { } ) around an initializer or by extra characters after the end of an initializer.

**A2186      extra characters in literal initialization**

A literal structure initializer was not properly delimited.

One of the following may have occurred:

- There were missing or mismatched angle brackets (< >) or braces ({ }) around an initializer.
- There were extra characters after the end of an initializer.
- There was a syntax error in the structure initialization.

**A2187      must use floating point initializer**

A variable declared with the **REAL4**, **REAL8**, and **REAL10** directives must be initialized with a floating-point number or a question mark (?).

This error can be caused by giving an initializer in integer form (such as 18) instead of in floating-point form (18.0).

**A2188      cannot use .EXIT for OS\_OS2 with .8086**

The **INVOKE** generated by the **.EXIT** statement under **OS\_OS2** requires the **.186** (or higher) directive, since it must be able to use the **PUSH** instruction to push immediates directly.

**A2189      invalid combination with segment alignment**

The alignment specified by the **ALIGN** or **EVEN** directive was greater than the current segment alignment as specified by the **SEGMENT** directive.

**A2190      INVOKE requires prototype for procedure**

The **INVOKE** directive must be preceded by a **PROTO** statement for the procedure being called.

When using **INVOKE** with an address rather than an explicit procedure name, you must precede the address with a pointer to the prototype.

**A2191      cannot include structure in self**

You cannot reference a structure recursively (inside its own definition).

**A2192      symbol language attribute conflict**

Two declarations for the same symbol have conflicting language attributes (such as **C** and **PASCAL**). The attributes should be identical or compatible.

**A2193      non-benign COMM redefinition**

A variable was redefined with the **COMM** directive to a different language type, distance, size, or instance count.

Multiple **COMM** definitions of a variable must be identical.

**A2194      COMM variable exceeds 64K**

A variable declared with the **COMM** directive in a 16-bit segment was greater than 64K.

**A2195     parameter or local cannot have void type**

The assembler attempted to create an argument or create a local without a type.

This error can be caused by declaring or passing a symbol followed by a colon without specifying a type or by using a user-defined type defined as void.

**A2196     cannot use TINY model with OS\_OS2**

A **.MODEL** statement specified the **TINY** memory model and the **OS\_OS2** operating system. The tiny memory model is not allowed under OS/2.

**A2197     expression size must be 32-bits**

There was an attempt to use the 16-bit expression evaluator in a 32-bit segment. In a 32-bit segment (**USE32** or **FLAT**), you cannot use the default 16-bit expression evaluator (**OPTION EXPR16**).

**A2198     .EXIT does not work with 32-bit segments**

The **.EXIT** directive cannot be used in a 32-bit segment; it is valid only when generating 16-bit code.

**A2199     .STARTUP does not work with 32-bit segments**

The **.STARTUP** directive cannot be used in a 32-bit segment; it is valid only when generating 16-bit code.

**A2200     ORG directive not allowed in unions**

The **ORG** directive is not valid inside a **UNION** definition.

You can use the **ORG** directive inside **STRUCT** definitions, but it is meaningless inside a **UNION**.

**A2201     scope state cannot be changed**

Both **OPTION SCOPED** and **OPTION NOSCOPED** statements occurred in a module. You cannot switch scoping behavior in a module.

This error may be caused by an **OPTION SCOPED** or **OPTION NOSCOPED** statement in an include file.

**A2202     illegal use of segment register**

You cannot use segment overrides for the FS or GS segment registers when generating floating-point emulation instructions with the **/FPi** command-line option or **OPTION EMULATOR**.

**A2203     cannot declare scoped code label as PUBLIC**

A code label defined with the "label:" syntax was declared **PUBLIC**. Use the "label::" syntax, the **LABEL** directive, or **OPTION NOSCOPED** to eliminate this error.

**A2204     .MSFLOAT directive is obsolete : ignored**

The Microsoft Binary Format is no longer supported. You should convert your code to the IEEE numeric standard, which is used in the 80x87-series coprocessors.

**A2205      ESC instruction is obsolete : ignored**

The **ESC** (Escape) instruction is no longer supported. All numeric coprocessor instructions are now supported directly by the assembler.

**A2206      missing operator in expression**

An expression cannot be evaluated because it is missing an operator. This error message may also be a side-effect of a preceding program error.

The following line will generate this error:

```
value1 = ( 1 + 2 ) 3
```

**A2207      missing right parenthesis in expression**

An expression cannot be evaluated because it is missing a right (closing) parenthesis. This error message may also be a side-effect of a preceding program error.

The following line will generate this error:

```
value1 = ( ( 1 + 2 ) * 3
```

**A2208      missing left parenthesis in expression**

An expression cannot be evaluated because it is missing a left (opening) parenthesis. This error message may also be a side-effect of a preceding program error.

The following line will generate this error:

```
value1 = ( ( 1 + 2 ) * 3 ) )
```

**A2209      reference to forward macro redefinition**

A macro cannot be accessed because it has not been yet defined.

Move the macro definition ahead of all references to the macro.

**A2901      cannot run ML.EXE**

The MASM driver could not spawn ML.EXE.

One of the following may have occurred:

- 🔍 ML.EXE was not in the path.
- 🔍 The READ attribute was not set on ML.EXE.
- 🔍 There was not enough memory.

## ML Warnings

**A4000      cannot modify READONLY segment**

An attempt was made to modify an operand in a segment marked with the **READONLY** attribute.

**A4002 non-unique STRUCT/UNION field used without qualification**

A **STRUCT** or **UNION** field can be referenced without qualification only if it has a unique identifier.

This conflict can be resolved either by renaming one of the structure fields to make it unique or by fully specifying both field references.

The **NONUNIQUE** keyword requires that all references to the elements of a **STRUCT** or **UNION** be fully specified.

**A4003 start address on END directive ignored with .STARTUP**

Both **.STARTUP** and a program load address (optional with the **END** directive) were specified. The address specification with the **END** directive was ignored.

**A4004 cannot ASSUME CS**

An attempt was made to assume a value for the CS register. CS is always set to the current segment or group.

**A4005 unknown default prologue argument**

An unknown argument was passed to the default prologue.

The default prologue understands only the **FORCEFRAME** and **LOADDS** arguments.

**A4006 too many arguments in macro call**

There were more arguments given in the macro call than there were parameters in the macro definition.

**A4007 option untranslated, directive required : *option***

There is no ML command-line equivalent for the given MASM option. The desired behavior can be obtained by using a directive in the source file.

Option	Directive
/A	.ALPHA
/P	OPTION READONLY
/S	.SEQ

**A4008 invalid command-line option value, default is used : *option***

The value specified with the given option was not valid. The option was ignored, and the default was assumed.

**A4009 insufficient memory for /EP : /EP ignored**

There is not enough memory to generate a first-pass listing.

**A4010 expected '>' on text literal**

A macro was called with a text literal argument that was missing a closing angle bracket.

**A4011 multiple .MODEL directives found : .MODEL ignored**

More than one **.MODEL** directive was found in the current module. Only the first **.MODEL** statement is used.

**A4012 line number information for segment without class 'CODE'**

There were instructions in a segment that did not have a class name that ends with "CODE." The assembler did not generate CodeView information for these instructions.

CodeView cannot process modules with code in segments with class names that do not end with "CODE."

**A4013 instructions and initialized data not supported in AT segments**

An instruction or initialized data was found in a segment defined with the AT attribute. The code or data will not be loaded at run time.

Data in AT segments must be declared with the ? initializer.

**A4910 cannot open file: *filename***

The given filename could not be in the current path.

Make sure that *filename* was copied from the distribution disks and is in the current path.

**A5000 @@: label defined but not referenced**

A jump target was defined with the @@: label, but the target was not used by a jump instruction.

One common cause of this error is insertion of an extra @@: label between the jump and the @@: label that the jump originally referred to.

**A5001 expression expected, assume value 0**

There was an **IF**, **ELSEIF**, **IFE**, **IFNE**, **ELSEIFE**, or **ELSEIFNE** directive without an expression to evaluate. The assembler assumes a 0 for the comparison expression.

**A5002 externdef previously assumed to be external**

The **OPATTR** or **.TYPE** operator was applied to a symbol after the symbol was used in an **EXTERNDEF** statement but before it was declared. These operators were used on a line where the assembler assumed that the symbol was external.

**A5003 length of symbol previously assumed to be different**

The **LENGTHOF**, **LENGTH**, **SIZEOF**, or **SIZE** operator was applied to a symbol after the symbol was used in an **EXTERNDEF** statement but before it was declared. These operators were used on a line where the assembler assumed that the symbol had a different length and size.

**A5004 symbol previously assumed to not be in a group**

A symbol was used in an **EXTERNDEF** statement outside of a segment and then was declared inside a segment.



**A5005      types are different**

The type given by an **INVOKE** statement differed from that given in the procedure prototype. The assembler performed the appropriate type conversion.

**A6001      no return from procedure**

A **PROC** statement generated a prologue, but there was no **RET** or **IRET** instruction found inside the procedure block.

**A6003      conditional jump lengthened**

A conditional jump was encoded as a reverse conditional jump around a near unconditional jump.

You may be able to rearrange code to avoid the longer form.

**A6004      procedure argument or local not referenced**

You passed a procedure argument or created a variable with the **LOCAL** directive that was not used in the procedure body.

Unnecessary parameters and locals waste code and stack space.

**A6005      expression condition may be pass-dependent**

Under the **/Zm** command-line option or the **OPTION M510** directive, the value of an expression changed between passes.

This error message may indicate that the code is pass-dependent and must be rewritten.

## NMAKE Error Messages

This section lists error messages generated by the NMAKE utility.

Microsoft Program Maintenance Utility (NMAKE) generates the following error messages:

- 🔒 Fatal errors (U1000 through U1099) cause NMAKE to stop execution.
- 🔒 Errors (U2001) do not stop execution but prevent NMAKE from completing the make process.
- 🔒 Warnings (U4001 through U4011) indicate possible problems in the make process.

## NMAKE Fatal Error Messages

**U1000      syntax error : ')' missing in macro invocation**

A left parenthesis, (, appeared without a matching right parenthesis, ), in a macro invocation. The correct form is **\$(name)**, and **\$n** is allowed for one-character names.

**U1001     syntax error : illegal character *character* in macro**

The given character appeared in a macro but was not a letter, number, or underscore (\_).

If the colon (:) is omitted in a macro expansion, the following error occurs:

```
syntax error : illegal character '=' in macro
```

**U1002     syntax error : invalid macro invocation '\$'**

A single dollar sign (\$) appeared without a macro name associated with it.

The correct form is \$(*name*). To specify a dollar sign, use a double dollar sign (\$\$) or precede it with a caret (^).

**U1003     syntax error : '=' missing in macro substitution**

A macro invocation contained a colon (:), which begins a substitution, but it did not contain an equal sign (=).

The correct form is:

```
$(macroname: oldstring=newstring)
```

**U1004     syntax error : macro name missing**

One of the following occurred:

- ☛ The name of a macro being defined was itself a macro invocation that expanded to nothing. For example, if the macro named **ONE** is undefined or has a null value, the following macro definition causes this error:

```
$(ONE)=TWO
```

- ☛ A macro invocation did not specify a name in the parentheses. The following specification causes this error:

```
$()
```

The correct form is:

```
$(name)
```

**U1005     syntax error : text must follow ':' in macro**

A string substitution was specified for a macro, but the string to be changed in the macro was not specified.

**U1006     syntax error : missing closing double quotation mark**

An opening double quotation mark (") appeared without a closing double quotation mark.

**U1007     double quotation mark not allowed in name**

The specified target name or filename contained a double quotation mark (").

Double quotation marks can surround a filename but cannot be contained within it.

- U1017      unknown directive *!directive***  
The specified directive is not one of the recognized directives.
- U1018      directive and/or expression part missing**  
The directive was incompletely specified.  
The expression part of the directive is required.
- U1019      too many nested **!IF** blocks**  
The limit on nesting of **!IF** directives was exceeded.  
The **!IF** preprocessing directives include **!IF**, **!IFDEF**, **!IFNDEF**, **!ELSE IF**, **!ELSE IFDEF**, and **!ELSE IFNDEF**.
- U1020      end-of-file found before next directive**  
An expected directive was missing.  
For example, an **!IF** was not followed by an **!ENDIF**.
- U1021      syntax error : **!ELSE** unexpected**  
An **!ELSE** directive was found that was not preceded by an **!IF** directive, or the directive was placed in a syntactically incorrect place.  
The **!IF** preprocessing directives include **!IF**, **!IFDEF**, **!IFNDEF**, **!ELSE IF**, **!ELSE IFDEF**, and **!ELSE IFNDEF**.
- U1022      missing terminating character for string/program invocation : *char***  
The closing double quotation mark (") in a string comparison in a directive was missing, or the closing bracket (]) in a program invocation in a directive was missing.
- U1023      syntax error in expression**  
An expression was invalid.  
Check the allowed operators and operator precedence.
- U1024      illegal argument to **!CMDSWITCHES****  
An unrecognized command switch was specified.
- U1031      filename missing (or macro is null)**  
An **!INCLUDE** directive was found, but the name of the file to be included was missing or a macro representing the filename expanded to nothing.

**U1033      syntax error : *string* unexpected**

The given string is not part of the valid syntax for a makefile.

The following are examples of causes and results of this error:

- If the closing set of angle brackets for an inline file are not at the beginning of a line, the following error occurs:

```
syntax error : 'EOF' unexpected
```

- If a macro definition in the makefile contained an equal sign (=) without a preceding name or if the name being defined is a macro that expands to nothing, the following error occurs:

```
syntax error : '=' unexpected
```

- If the semicolon (;) in a comment line in TOOLS.INI is not at the beginning of the line, the following error occurs:

```
syntax error : ';' unexpected
```

- If the makefile has been formatted by a word processor, the following error can occur:

```
syntax error : ':' unexpected
```

**U1034      syntax error : separator missing**

The colon (:) that separates targets and dependents is missing.

**U1035      syntax error : expected ':' or '=' separator**

Either a colon (:) or an equal sign (=) was expected.

Possible causes include the following:

- A target was not followed by a colon.
- A single-letter target was followed by a colon and no space (such as **a:**). NMAKE interpreted it as a drive specification.
- An inference rule was not followed by a colon.
- A macro definition was not followed by an equal sign.
- A character followed a backslash (\) that was used to continue a command to a new line.
- A string appeared that did not follow any NMAKE syntax rule.
- The makefile was formatted by a word processor.

**U1036      syntax error : too many names to left of '='**

Only one string is allowed to the left of a macro definition.

- U1037     syntax error : target name missing**  
A colon (:) was found before a target name was found.  
At least one target is required.
- U1038     internal error : lexer**  
Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.
- U1039     internal error : parser**  
Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.
- U1040     internal error : macro expansion**  
Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.
- U1041     internal error : target building**  
Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.
- U1042     internal error : expression stack overflow**  
Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.
- U1043     internal error : temp file limit exceeded**  
Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.
- U1045     spawn failed : *message***  
A program or command, called by NMAKE, failed for the given reason.
- U1047     argument before ')' expands to nothing**  
The parentheses following the preprocessing operator **DEFINED** or **EXIST** either were empty or contained an argument that evaluated to a null string.
- U1048     cannot write to file *filename***  
NMAKE could not write to the given file.  
One cause of this error is a read-only file specified with /X.
- U1049     macro or inline file too long (maximum : 64K)**  
An inline file or a macro exceeded the limit of 64K.
- U1050     *user-specified text***  
The message specified with the **!ERROR** directive was displayed.

**U1051 out of memory**

The makefile was too large or complex for available memory.

**U1052 file *filename* not found**

NMAKE could not find the given file, which was specified with one of the following:

- The /F option
- The **!INCLUDE** preprocessing directive
- The at sign (@) specifier for a response file

Check that the file exists and the filename is spelled correctly.

**U1053 file *filename* unreadable**

The file cannot be read.

One of the following may be a cause:

- The file is in use by another process.
- A bad area exists on disk.
- A bad file-allocation table exists.

**U1054 cannot create inline file *filename***

NMAKE failed to create the given inline file.

One of the following may be a cause:

- A file by that name exists with a read-only attribute.
- The disk is full.

**U1055 out of environment space**

The operating system ran out of room for environment variables.

Either increase the environment space or set fewer environment variables.

**U1056 cannot find command processor**

The command processor was not in the path specified in the COMSPEC or PATH environment variables.

NMAKE uses COMMAND.COM or CMD.EXE as a command processor when executing commands. It looks for the command processor first in the path set in COMSPEC. If COMSPEC does not exist, NMAKE searches the directories specified in PATH.

**U1057 cannot delete temporary file *filename***

NMAKE failed to delete the temporary inline file.

**U1058      terminated by user**

NMAKE was halted by CTRL+C or CTRL+BREAK.

**U1059      syntax error: '}' missing in dependent**

A search path for a dependent was incorrectly specified. Either a space existed in the path or the closing brace (}) was omitted. The syntax for a directory specification for a dependent is:

{ *directories* } dependent

where *directories* specifies one or more paths, each separated by a semicolon (;). No spaces are allowed.

If part or all of a search path is replaced by a macro, be sure that no spaces exist in the macro expansion.

**U1060      unable to close file : *filename***

NMAKE encountered an error while closing a file.

One of the following may be a cause:

- 🔒 The file is a read-only file.
- 🔒 There is a locking or sharing violation.
- 🔒 The disk is full.

**U1061      /F option requires a filename**

The /F command-line option must be followed by either a makefile name or a dash (-), which represents standard input.

**U1062      missing filename with /X option**

The /X command-line option requires the name of the file to which diagnostic error output should be redirected.

To use standard output, specify '-' as the output filename.

**U1063      missing macro name before '='**

A macro definition on the NMAKE command line contained an equal sign (=) without a preceding name.

This error can occur if the macro name being defined is itself a macro that expands to nothing.

**U1064    MAKEFILE not found and no target specified**

The NMAKE command line did not specify a makefile or a target, and the current directory did not contain a file named MAKEFILE.

NMAKE requires either a makefile or a command-line target. To make a makefile available to NMAKE, either specify the /F option or place a file named MAKEFILE in the current directory. NMAKE can create a command-line target by using an inference rule if a makefile is not provided.

**U1065    invalid option *option***

The specified option is not a valid option for NMAKE.

**U1069    no match found for wildcard *filename***

There is no file that matches the given filename, which was specified using one or more wildcards (\* and ?).

A target file specified using a wildcard must exist on disk.

**U1070    cycle in macro definition *macroname***

The given macro definition contained a macro whose definition contained the given macro. Circular macro definitions are invalid.

For example, the following macro definitions:

```
ONE=$(TWO)
TWO=$(ONE)
```

cause the following error:

```
cycle in macro definition 'TWO'
```

**U1071    cycle in dependency tree for target *targetname***

A circular dependency exists in the dependency tree for the given target. The given target is a dependent of one of the dependents of the given target. Circular dependencies are invalid.

**U1072    cycle in include files : *filename***

The given file includes a file that eventually includes the given file. Inclusions (using the **!INCLUDE** preprocessing directive) cannot be circular.

**U1073    don't know how to make *targetname***

The specified target does not exist, and there is no command to execute or inference rule to apply.

One of the following may be a solution:

- 🔍 Check the spelling of the target name.
- 🔍 If *targetname* is a pseudotarget, specify it as a target in another description block.
- 🔍 If *targetname* is a macro invocation, be sure it does not expand to a null string.



**U1076 name too long**

A string exceeded one of the following limits:

- A macro name cannot exceed 1024 characters.
- A target name (including path) cannot exceed 256 characters.
- A command cannot exceed 2048 characters.

**U1077 *program* : return code value**

The given command or program called by NMAKE failed and returned the given exit code.

To suppress this error and continue the NMAKE session, use the /I option, the .IGNORE dot directive, or the dash (–) command modifier. To continue the NMAKE session for unrelated parts of the dependency tree, use the /K option.

**U1078 constant overflow at *expression***

The given expression contained a constant that exceeded the range –2,147,483,648 to 2,147,483,647. The constant appeared in one of the following situations:

- An expression specified with a preprocessing directive
- An error level specified with the dash (–) command modifier

**U1079 illegal expression : divide by zero**

An expression tried to divide by zero.

**U1080 operator and/or operand usage illegal**

The expression incorrectly used an operator or operand.

Check the allowed set of operators and their order of precedence.

**U1081 *filename* : program not found**

NMAKE could not find the given program in order to run it.

Make sure that the program is in a directory specified in the PATH environment variable and is not misspelled.

**U1082 *command* : cannot execute command; out of memory**

There is not enough memory to execute the given command.

**U1083 target macro *target* expands to nothing**

The given target is an invocation of a macro that has not been defined or has a null value. NMAKE cannot process a null target.

**U1084 cannot create temporary file *filename***

NMAKE was unable to create the temporary file it needs when it processes the makefile.

One of the following may be a cause:

- 🔒 The file already exists with a read-only attribute.
- 🔒 There is insufficient disk space to create the file.
- 🔒 The directory specified in the TMP environment variable does not exist.

**U1085 cannot mix implicit and explicit rules**

A target and a pair of inference-rule extensions were specified on the same line. Targets cannot be named in inference rules.

**U1086 inference rule cannot have dependents**

The colon (:) in an inference rule must be followed by one of the following:

- 🔒 A newline character
- 🔒 A semicolon (;), which can be followed by a command
- 🔒 A number sign (#), which can be followed by a comment

**U1087 cannot have : and :: dependents for same target**

A target cannot be specified in both a single-colon (:) and a double-colon (::) dependency.

To specify a target in multiple description blocks, use :: in each dependency line.

**U1088 invalid separator '::' on inference rule**

An inference rule must be followed by a single colon (:).

**U1089 cannot have build commands for directive *targetname***

Dot directives cannot be followed by commands. The dot directives are **.IGNORE**, **.PRECIOUS**, **.SILENT**, and **.SUFFIXES**.

**U1090 cannot have dependents for directive *targetname***

Dot directives cannot be followed by dependents. The dot directives are **.IGNORE**, **.PRECIOUS**, **.SILENT**, and **.SUFFIXES**.

**U1092 too many names in rule**

An inference rule cannot specify more than two extensions.

**U1093 cannot mix dot directives**

Multiple dot directives cannot be specified on one line. The dot directives are **.IGNORE**, **.PRECIOUS**, **.SILENT**, and **.SUFFIXES**.

**U1094     syntax error : only (NO)KEEP allowed here**

Something other than **KEEP** or **NOKEEP** appeared after the closing set of angle brackets (<<) specifying an inline file. Only **KEEP**, **NOKEEP**, or a newline character may follow the angle brackets. No spaces, tabs, or other characters may appear.

**KEEP** preserves the inline file on disk. **NOKEEP** deletes the file after the NMAKE session. The default is **NOKEEP**.

**U1095     expanded command line *commandline* too long**

After macro expansion, the given command line exceeded the limit on length of command lines for the operating system.

MS-DOS permits up to 128 characters on a command line.

If the command is for a program that can accept command-line input from a file, change the command and supply input from either a file on disk or an inline file. For example, LINK and LIB accept input from a response file.

**U1096     cannot open inline file *filename***

NMAKE could not create the given inline file.

One of the following occurred:

• The disk was full.

• A file with that name exists as a read-only file.

**U1097     filename-parts syntax requires dependent**

The current dependency does not have either an explicit dependent or an implicit dependent. Filename-parts syntax, which uses the percent (%) specifier, represents components of the first dependent of the current target.

**U1098     illegal filename-parts syntax in *string***

The given string does not contain valid filename-parts syntax.

**U1099     The makefile being processed was too complex for the current stack allocation in NMAKE. NMAKE has an allocation of 0x3000 (12K).**

To increase NMAKE's stack allocation, run the EXEHDR utility with a larger stack option:

```
EXEHDR /STACK: stacksize
```

where **stacksize** is a number greater than the current stack allocation in NMAKE.

## NMAKE Error Messages

**U2001     no more file handles (too many files open)**

NMAKE could not find a free file handle.

One of the following may be a solution:

- 🔧 Reduce recursion in the build procedures.
- 🔧 In MS-DOS, increase the number of file handles by changing the FILES setting in CONFIG.SYS to allow a larger number of open files. FILES=50 is the recommended setting.

## NMAKE Warning Messages

**U4001     command file can be invoked only from command line**

A command file, which is invoked by the at sign (@) specifier, cannot contain a specification for another command file. Such nesting is not allowed. The specification was ignored.

**U4002     resetting value of special macro *macroname***

The given predefined macro was redefined.

**U4004      too many rules for target *targetname***

More than one description block was specified for the given target using single colons (:) as separators. NMAKE executed the commands in the first description block and ignored later blocks.

To specify the same target in multiple dependencies, use double colons (::) as the separator in each dependency line.

**U4005      ignoring rule *rule* (extension not in .SUFFIXES)**

The given rule contained a suffix that is not specified in the .SUFFIXES list. NMAKE ignored the rule.

This warning appears only when the /P option is used.

**U4006      special macro undefined : *macroname***

The given special macro name is undefined and expands to nothing.

**U4007      filename *filename* too long; truncating to 8.3**

The base name of the given file has more than 8 characters, or the extension has more than three characters. NMAKE truncated the name to an 8-character base and a 3-character extension.

If long filenames are supported by your file system, enclose the name in double quotation marks ("").

**U4008      removed target *target***

NMAKE was interrupted while trying to build the given target, and the target file was incomplete. Because the target was not specified in the .PRECIOUS list, NMAKE deleted the file.

**U4010      *target* : build failed; /K specified, continuing ...**

A command in the commands block for the given target returned a nonzero exit code. The /K option told NMAKE to continue processing unrelated parts of the build and to issue an exit code 1 when the NMAKE session is finished.

If the given target is itself a dependent for another target, NMAKE issues warning U4011 after this warning.

**U4011      *target* : not all dependents available; target not built**

A dependent of the given target either did not exist or was out of date, and a command for updating the dependent returned a nonzero exit code. The /K option told NMAKE to continue processing unrelated parts of the build and to issue an exit code 1 when the NMAKE session is finished.

This warning is preceded by warning U4010 for each dependent that failed to be created or updated.

## PWB Error Messages

PWB displays an error message whenever it detects a command it cannot execute. Most errors terminate the command that is in error, but do not terminate PWB.

For most errors, PWB displays a message box with only the text of the message. The error number does not appear. With these messages, press F1 or click Help when the message box is displayed for Help on the error. Some errors terminate PWB. PWB displays these fatal errors on the command line after returning to the operating system.

This section lists only the fatal PWB errors.

### PWB Fatal Errors

**PWB3089      Out of local memory. Unable to recover.**

PWB has run out of memory and cannot recover. This is a fatal PWB condition. However, PWB is able to save your files, and you can restart PWB to continue.

This can happen when using PWB continuously for a long time.

This can also happen when creating a project with a very large number of files or adding files to a large project. To make the largest amount of memory available to PWB for creating a very large project, load only the PWBUTILS extension and only the language extensions you need for the project. Start PWB with the /DS option, and create the project before doing any other work.

If the project is too large for PWB to handle as a PWB project, you can use a non-PWB makefile for your project.

**PWB3090      Out of virtual memory space. Unable to recover.**

PWB has run out of virtual memory and cannot recover. This is a fatal PWB condition. However, PWB is able to save your files, and you can restart PWB to continue.

**PWB3096      Unsupported video mode. Please change modes and restart.**

A request was made to start PWB with the **Savescreen** switch set to yes (the default), but PWB does not support the current operating-system video mode.

Change the video mode and restart PWB.

- PWB3178      Cannot start: unable to open swapping file**  
PWB is unable to create its virtual-memory file on disk.  
PWB creates this file in the directory pointed to by the TMP environment variable. If no TMP environment variable is set, PWB creates the file in the current directory.  
Check that the disk has at least 2 free megabytes and that the directory can be accessed with permission to create a file. Check that the TMP environment variable lists a single existing directory.
- PWB3180      Cannot start: not enough far memory**  
PWB ran out of memory while starting up.  
Make more memory available to PWB and restart PWB.
- PWB3181      Cannot initialize**  
PWB cannot initialize itself.  
Check that there is enough memory available for PWB. Also, check that there is no conflict with a TSR (terminate-and-stay-resident) program.
- PWB3901      RE: error *number*, line *line***  
PWB has encountered an error while processing a regular expression. The expression may be malformed or too complex.  
Check that the syntax of the regular expression is correct.
- PWB3909      RemoveFile can't find file**  
PWB has encountered an internal error.  
Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the "Microsoft Support Services" section of the introduction to this book.
- PWB3912      Internal VM Error**  
PWB has encountered an internal error.  
Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the "Microsoft Support Services" section of the introduction to this book.
- PWB12078      Cannot access *file*: *reason***  
PWB cannot access the given file for the stated reason.  
Correct the situation and restart PWB.
- PWB12086      Cannot access TMP directory: *reason***  
PWB cannot access the directory listed in the TMP environment variable for the stated reason.  
Correct the situation and restart PWB.

## SBRPACK Error Messages

This section lists error messages generated by the Microsoft Browse Information Compactor (SBRPACK). SBRPACK errors (SB *xxx*) are always fatal.

**SB1000 UNKNOWN ERROR****Contact Microsoft Product Support Services**

SBRPACK detected an unknown error condition.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the “Microsoft Support Services” section of the introduction to this book.

This error ends SBRPACK with exit code 1.

**SB1001 *option* : unknown option**

SBRPACK did not recognize the given option.

This error ends SBRPACK with exit code 1.

**SB1002 *sbrfile* : corrupt file**

The given .SBR file is corrupt or does not have the expected format.



Recompile to regenerate the .SBR file.

This error ends SBRPACK with exit code 2.

**SB1003 *sbrfile* : invalid .SBR file**

SBRPACK did not recognize the given file as an .SBR file.

One of the following may be a solution:

-  Check the spelling of the specified file.
-  Recompile to regenerate the .SBR file.

This error ends SBRPACK with exit code 2.



**SB1004**     *sbrfile* : incompatible .SBR version

The given .SBR file cannot be packed by this version of SBRPACK.

One of the following may be a cause:

- 🔒 The .SBR file was created by a compiler that is not compatible with this version of SBRPACK.
- 🔒 The .SBR file is corrupt.

This error ends SBRPACK with exit code 2.

**SB1005**     *sbrfile* : cannot open file

SBRPACK cannot open the given .SBR file.

One of the following may be a cause:

- 🔒 The .SBR file does not exist. Check the spelling.
- 🔒 The .SBR file was locked by another process.

This error ends SBRPACK with exit code 3.

**SB1006**     cannot create temporary .SBR file

One of the following may have occurred:

- 🔒 No more file handles were available. Increase the number of file handles by changing the FILES setting in CONFIG.SYS to allow a larger number of open files. FILES=50 is recommended.
- 🔒 The disk was full.

This error ends SBRPACK with exit code 4.

