RVCT 3.0 SP1 Build Tools - Errors and Warnings

Last updated March 2008

Introduction

This document illustrates the errors and warning messages that are generated by the Build Tools of ARM RealView Compilation Tools 3.0, 3.0 Service Pack 1, and subsequent 3.0 patches. If you are using ADS (ADS 1.2, 1.1 or 1.0.1) or RVCT 1.2 then please refer to the "ADS 1.2 Build Tools – Errors and Warnings" document instead. If you are using an earlier version of RVCT then use this as a first source of reference but "RVCT 2.0 Build Tools - Errors and Warnings" may also be of use.

This document is divided into the following sections:

1. Introduction
2. ARM C and C++ Compilers (armcc, tcc, armcpp, tcpp)
3. ARM Assembler (armasm)
4. ARM Linker (armlink)
5. ARM ELF Format Conversion Utility (fromelf)
6. ARM Librarian (armar)
7. ARM Via file handling (General to all sections)

The errors and warnings are listed in numeric order. Not all the errors and warnings are yet fully described. The majority of the warnings and errors produced by the build tools are self-explanatory. However, if there are any messages that you do not understand, or which you need more information about, then please contact your supplier, providing as much information as possible about your system and commands used.

Note that this document does not include any reference for errors and warnings emitted by the Licence Management software. For information on this, please see the License Management FAQ at http://www.arm.com/support/licensemanagement.html

This document is intended to complement, not replace, the RVCT documentation. It should be read in conjunction with the RVCT build tools manuals, in particular the section(s) referring to controlling of warning and error message generation. We would also recommend that you consult the RVCT FAQ at http://www.arm.com/support/rvds3_faq.html for further information

Please also ensure that you have the latest "patch" of the build tool(s) you are using. These are downloadable from the appropriate link at http://www.arm.com/support/downloads

Long options should now be prefixed by double minus (--), this replaces the old standard in ADS where all options used single minus (-). The change was made for better compliance with the POSIX standard.

From RVDS 2.2 onwards some error messages now contain a more detailed reason why the error/warning occurred. This is noted as <reason> throughout this document. Some obsolete messages have been included for completeness. These messages have been prefixed with "Obsolete from 2.2 onwards".

Contained in Section 7 are general messages which apply to more than one tool. The x prefixing the message number within this documentation is replaced in the real tool output with the appropriate letter relating to that application.

Copyright © 2006-2008 ARM Limited. All rights reserved.

2. ARM C/C++ Compilers (armcc, tcc, armcpp, tccp)

Internal Errors and other unexpected failures

Internal errors in the compiler are typically errors that have occurred but have not yet been documented, or they may point to a potential "bug" in the compiler itself. If such errors are not present in this document, you will be required to supply an example of the source code that causes the error(s), to your tools supplier.

To facilitate the investigation, please try to send only the single source file or function that is causing the error, plus the compiler options used when compiling the code. It may be necessary to preprocess the file (i.e. to take account of #include'd header files, etc). To do this, pass the file through the preprocessor as follows:

```
        armcc <options> –E sourcefile.c > PPsourcefile.c
or        tcc <options> –E sourcefile.c > PPsourcefile.c
```

where `<options>` are your normal compile switches, (-O2, -g, -I, -D, etc), but *without* -c.

Check that the error is still reproducible with the preprocessed file by compiling it with:

```
        armcc <options> -c PPsourcefile.c
or        tcc <options> –c PPsourcefile.c
```

and then provide the "`PPsourcefile.c`" file, plus the compile `<options>`, to your supplier.

Controlling the Errors and Warnings Messages

This is documented in **RVCT 3.0 Compiler and Libraries Guide Section 2.3.15**. The compiler will normally warn of potential portability problems and other hazards.

When porting legacy code (e.g. in old-style C) to the ARM, many warnings may be reported. It may be tempting to disable all such warnings with "`–W`", however, our recommendation, for portability reasons, is to change the code to make it ANSI compatible, rather than suppressing the warnings.

Some warnings are suppressed by default. To override this, the "`--strict_warnings`" switch can be used to enable all those warnings that are suppressed by default.

List of Errors and Warnings Messages

`1: last line of file ends without a new line`

`2: last line of file ends with a backslash`

`3: #include file <filename> includes itself`

`4: out of memory`

`5: cannot open <type> input file ‘<filename>’: <reason>`
Example:
`#include <file.h>`
`Error:  #5: cannot open source input file "file.h": No such file or directory`
because file.h does not exist in the system include directory.

Copyright © 2006-2008 ARM Limited. All rights reserved.

6: `comment unclosed at end of file`
Comment started with /* but no matching */ to close the comment.

7: `unrecognized token`

8: `missing closing quote`
For example:
```
char foo[] = {"\" };
```

9: `nested comment is not allowed`
For example:
```
/*nested
  /*comment*/
```

10: `"#" not expected here`
A '#' character is in an incorrect position

11: `unrecognized preprocessing directive`
For example:
```
#foo
```

12: `parsing restarts here after previous syntax error`

13: `expected a file name`
For example:
```
#include <stdio.h>
```

14: `extra text after expected end of preprocessing directive`
For example:
```
#if EMBEDDED foo
```

Or:
```
#include <stdio.h> foo
```

Or:
```
#ifdef SOMETHING
      :
#endif SOMETHING
```

The `#endif` does not expect or require any argument. Enclosing the trailing part of the line in a comment should cure the problem, e.g.
```
#endif /* SOMETHING */
```

16: `<filename> is not a valid source file name`

17: `expected a "]"`

18: `expected a ")"`
For example:
```
int main(void
{
```

where there is a missing ")".

19: `extra text after expected end of number`
For example:
```
int a = 37r;
```

20: `identifier <identifier> is undefined`
Example when compiled for C++:
```
void foo( arg ) { }
```


Copyright © 2006-2008 ARM Limited. All rights reserved.

gives:
```
Error:  #20: identifier <arg> is undefined
```

This is a common error that occurs where there is no prototype for a function.
e.g. when `printf()` is used with no `#include <stdio.h>`, the warning occurs:

```
void foo(void)
{
      printf("foo");
}
```

gives:
```
Error:  #20: identifier "printf" is undefined
```
Example:
```
int foo(void)
{
      int a = 4;
      a = i;
}
```
results in the error:

```
Error:  #20: identifier "i" is undefined
```
because "i" has not been declared.

```
21: type qualifiers are meaningless in this declaration
```

```
22: invalid hexadecimal number
```

```
23: integer constant is too large
```

```
24: invalid octal digit
```
digit 8 or 9 found in octal number
For example:
```
int a = 0378;
```

```
25: quoted string should contain at least one character
```
For example:
```
char a ='';
```

```
26: too many characters in character constant
```
For example:
```
char a ='abcd';
```

```
27: character value is out of range
```
For example:
```
char foo[] = {"\xBBBB" };
```
gives:
```
Warning:  #27-D: character value is out of range
```

```
28: expression must have a constant value
```

```
29: expected an expression
```

```
30: floating constant is out of range
```

```
31: expression must have integral type
```

```
32: expression must have arithmetic type
```

```
33: expected a line number
```


Copyright © 2006-2008 ARM Limited. All rights reserved.

34: invalid line number

35: #error directive: <number>

36: the #if for this directive is missing

37: the #endif for this directive is missing
An open #if was still active, but was not closed with #endif before the End Of File.

38: directive is not allowed -- an #else has already appeared

39: division by zero

40: expected an identifier
This error is raised if preprocessor statements are incorrectly formatted. For example if the identifier which immediately should follow a preprocessor command is missing, e.g.
Missing identifier after #define, results in:
```
Error:  #40: expected an identifier
```

This error can also occur when C code containing C++ keywords is compiled with the C++ compiler, for example:
```
int *new(void *p) { return p; }
```
because "new" is a keyword in C++.

41: expression must have arithmetic or pointer type

42: operand types are incompatible (<type> and <type>)

44: expression must have pointer type

45: #undef may not be used on this predefined name

46: this predefined name may not be redefined

47: incompatible redefinition of macro <entity>
`Macro` has been defined twice (with different replacement strings).
If you need to do this, undefine the macro (`#undef`) before the second definition.
Example:

```
#define TEST 0
#define TEST 1
```

Causes the compiler to produce:
```
Warning:  #47-D: incompatible redefinition of macro "TEST" (declared at line 1)
```

There is no way to control this error directly via a compiler option, but you can use conditional preprocessing. For example:

```
#ifdef TEST_EQUALS_ZERO
#define TEST 0
#else
#define TEST 1
#endif
```

Compiling with "armcc -c foo.c" will define TEST to be 1 (the default).
Compiling with "armcc -c -DTEST_EQUALS_ZERO foo.c" will define TEST to be 0.

49: duplicate macro parameter name

50: "##" may not be first in a macro definition

Copyright © 2006-2008 ARM Limited. All rights reserved.

51: "##" may not be last in a macro definition

52: expected a macro parameter name

53: expected a ":"

54: too few arguments in macro invocation

55: too many arguments in macro invocation

56: operand of sizeof may not be a function

57: this operator is not allowed in a constant expression

58: this operator is not allowed in a preprocessing expression

59: function call is not allowed in a constant expression

60: this operator is not allowed in an integral constant expression

61: integer operation result is out of range

62: shift count is negative

63: shift count is too large

64: declaration does not declare anything

For example:

```
int;
```

65: expected a ";"

66: enumeration value is out of "int" range

This diagnostic message will be generated by the compiler when an enum constant is outside the range of a signed int. For example:

```
typedef enum
{
    Bit31 = 0x80000000
} Bits;
```

When compiled in C mode by RVCT 3.0 this will generate the above message as a warning. Note that the compilers behaviour has changed between past versions and also when using "--enum_is_int" and "--strict" switches:

C Mode:

- By default RVCT 2.1 will treat all constants larger than INT_MAX as signed, without any error or warning. RVCT 2.2 and later will promote the constants to unsigned, however this will produce the warning.
- With "--enum_is_int", RVCT 2.1 will again treat the constant as signed and give no message. RVCT 2.2 will treat it as signed but will give a warning. In RVCT 2.2SP1 and later the warning will still be produced but the constant will be promoted to unsigned.
- For RVCT 2.1, 2.2, 2.2SP1 and later the switch "--strict" will always produce this message as an error.

C++ Mode:

Copyright © 2006-2008 ARM Limited. All rights reserved.

- By default the out-of-range constants are promoted to unsigned without a warning and also when "--strict" is used.
- With "--enum_is_int", RVCT 2.1 will treat the constant as signed without any message unless "--strict" is also supplied in which case the message becomes an error. For RVCT 2.2 with "--enum_is_int" the constant will be treated as signed, however a warning will be generated, even without "--strict". In RVCT 2.2SP1 and later the constants will be promoted to unsigned without a warning or an error, even if --strict is specified.

As a work around for cases where the message is an error use the following code example:

```
typedef enum
{
    Bit31 = (int)0x80000000
} Bits;
```

An overflow no longer occurs, and so no error is reported. Note, however, that the value of Bit31 is now negative because it is a signed int.

See RVCT 3.0 Compilers and Libraries Guide, section 3.5.4, "Structures, unions, enumerations, and bitfields" for more information.

67: `expected a "}"`

68: `integer conversion resulted in a change of sign`
The constant is too large to be represented in a signed long, and therefore has been given unsigned type.
Example:
```
long l = 2147483648;
```
gives:
```
Warning:  #68-D: integer conversion resulted in a change of sign
```

69: `integer conversion resulted in truncation`

70: `incomplete type is not allowed`
Example:
```
typedef struct {
        unsigned char size;
        char string[];
} FOO;
```

By not declaring a size for the array in the structure, the compiler will not be able to allocate a size of the structure.

71: `operand of sizeof may not be a bit field`

75: `operand of "*" must be a pointer`

76: `argument to macro is empty`

77: `this declaration has no storage class or type specifier`

78: `a parameter declaration may not have an initializer`

79: `expected a type specifier`
The ellipses to denote variadic functions, e.g. printf(), must follow at least one parameter, e.g change:
```
int foo( ... );
```


Copyright © 2006-2008 ARM Limited. All rights reserved.

to:
```
int foo( int bar, ... );
```

80: a storage class may not be specified here

81: more than one storage class may not be specified

82: storage class is not first

83: type qualifier specified more than once

84: invalid combination of type specifiers
The type name or type qualifier cannot be used in the same declaration as the second type name or type qualifier. For example:
```
typedef int int;
```

85: invalid storage class for a parameter

86: invalid storage class for a function

87: a type specifier may not be used here

88: array of functions is not allowed

89: array of void is not allowed

90: function returning function is not allowed

91: function returning array is not allowed

92: identifier-list parameters may only be used in a function definition

93: function type may not come from a typedef

94: the size of an array must be greater than zero
Zero-sized arrays are not allowed. For example:
```
char name[0] = "Hello";
```

95: array is too large
There is a limit of 4GB on the maximum size of arrays or structures.

96: a translation unit must contain at least one declaration

97: a function may not return a value of this type

98: an array may not have elements of this type

99: a declaration here must declare a parameter

100: duplicate parameter name

101: <param> has already been declared in the current scope

102: forward declaration of enum type is nonstandard

103: class is too large

104: struct or union is too large

105: invalid size for bit field
Bit fields must not be larger than the size of the type. Example (with --strict):

```
struct X{
```


Copyright © 2006-2008 ARM Limited. All rights reserved.

```
int y:5000;
};
```

106: invalid type for a bit field

Bit fields must have integral type. Example:

```
struct X{
float x:5;
float y:2;
};
```

107: zero-length bit field must be unnamed

108: signed bit field of length 1

109: expression must have (pointer-to-) function type

110: expected either a definition or a tag name

111: statement is unreachable

112: expected "while"

114: <entity> was referenced but not defined

115: a continue statement may only be used within a loop

116: a break statement may only be used within a loop or switch

Example:

```
void foo(void){
  int a=0;
  continue;
}
```

or:

```
void bar(void){
  int a=0;
  break;
}
```

117: non-void <entity> should return a value

118: a void function may not return a value

119: cast to type <type> is not allowed

120: return value type does not match the function type

121: a case label may only be used within a switch

122: a default label may only be used within a switch

123: case label value has already appeared in this switch

124: default label has already appeared in this switch

125: expected a "("

126: expression must be an lvalue

127: expected a statement

Copyright © 2006-2008 ARM Limited. All rights reserved.

128: loop is not reachable from preceding code

129: a block-scope function may only have extern storage class

130: expected a "{"

131: expression must have pointer-to-class type

132: expression must have pointer-to-struct-or-union type

133: expected a member name

134: expected a field name

135: <entity> has no member <member>

136: <entity> has no field <field>

137: expression must be a modifiable lvalue

138: taking the address of a register variable is not allowed

139: taking the address of a bit field is not allowed

140: too many arguments in function call
Function declaration does not match the number of parameters in an earlier function prototype. Example:

```
extern void foo(int x);
void bar(void)
{
      foo(1,2);
}
```

141: unnamed prototyped parameters not allowed when body is present

142: expression must have pointer-to-object type

143: program too large or complicated to compile

144: a value of type <type> cannot be used to initialize an entity of type <type>
The initializing string for a fixed size character array is exactly as long as the array size, leaving no room for a terminating \0, for example:

```
char name[5] = "Hello";
```

The name array can hold up to 5 characters. "Hello" will not fit because C strings are always null-terminated (e.g. "Hello\0"). So for the example above the compiler reports:

```
Error:  #144: a value of type "const char [6]" cannot be used to initialize an
entity of type "char [5]"
```

A similar error will also be raised if there is an implicit cast of non-0 int to pointer, e.g:

```
void foo_func( void )
{
        char *foo=1;
}
```

Gives:

```
#144: a value of type "int" cannot be used to initialize an entity of type
"char *"
```

For the second case this error can be suppressed with the use of the "--loose_implicit_cast" switch.

Copyright © 2006-2008 ARM Limited. All rights reserved.

145: <entity> may not be initialized

146: too many initializer values

147: declaration is incompatible with <entity>

148: <entity> has already been initialized

149: a global-scope declaration may not have this storage class

150: a type name may not be redeclared as a parameter

151: a typedef name may not be redeclared as a parameter

152: conversion of nonzero integer to pointer

153: expression must have class type

154: expression has struct or union type

155: old-fashioned assignment operator

156: old-fashioned initializer

157: expression must be an integral constant expression

158: expression must be an lvalue or a function designator

159: declaration is incompatible with previous "<entity>"

160: name conflicts with previously used external name <name>

161: unrecognized #pragma

163: could not open temporary file <filename>: <reason>

164: name of directory for temporary files is too long (<dirname>)

165: too few arguments in function call

Function prototype is defined with X number of parameters and does not match the number of parameters passed in the function call.

For example:

```
extern void foo(int x);
void bar(void)
{
      foo();
}
```

Gives:

```
#165: too few arguments in function call
```

166: invalid floating constant

167: argument of type <type> is incompatible with parameter of type <type>

168: a function type is not allowed here

169: expected a declaration

When attempting to compile some C++ header files with the C compiler instead of the C++ compiler, this error can be reported.

170: pointer points outside of underlying object

Copyright © 2006-2008 ARM Limited. All rights reserved.

171: invalid type conversion

172: external/internal linkage conflict with previous declaration
Errors about linkage disagreements where functions are implicitly declared as extern and then later re-declared as `static` are suppressed unless compiled with --strict.
Example:
```
extern void foo(void);
static void foo(void){}
```

173: floating-point value does not fit in required integral type

174: expression has no effect

175: subscript out of range

177: <entity> was declared but never referenced
By default, unused declaration warnings are given for:
- local (within a function) declarations of variables, typedefs, and functions
- labels (always within a function)
- top-level static functions and static variables.
The "`--diag_suppress 177`" option suppresses these warnings.

178: "&" applied to an array has no effect

179: right operand of "%" is zero

180: argument is incompatible with formal parameter

181: argument is incompatible with corresponding format string conversion
For example when compiling with --strict:
```
unsigned long foo = 0x1234;
printf("%0X", foo);
```
To avoid the warning, the code could be rewritten as:
```
unsigned long foo = 0x1234;
printf("%0lX", foo);
```
or perhaps:
```
unsigned int foo = 0x1234;
printf("%0X", foo);
```

"`%0X`" may be used for `char`, `short` or `int`. Use "`lX`" for a long integer, despite both `int`s and `long`s being 32 bits wide on an ARM.

182: could not open source file <filename> (no directories in search list)

183: type of cast must be integral

184: type of cast must be arithmetic or pointer

185: dynamic initialization in unreachable code

186: pointless comparison of unsigned integer with zero
Example:
```
unsigned short foo;
if (foo<0) printf("This never happens");
```

This is warning that the comparison between an unsigned (char, int, etc) value and zero will always evaluate to false.

187: use of "=" where "==" may have been intended

Copyright © 2006-2008 ARM Limited. All rights reserved.

Example:

```
int main(void)
{
int a;
const int b =1;
if (a=b)
}
```

188: `enumerated type mixed with another type`

189: `error while writing <filename> file`

190: `invalid intermediate language file`

191: `type qualifier is meaningless on cast type`

The C specification states that a cast does not yield an lvalue, so a cast to a qualified type has the same effect as a cast to the unqualified version of the type. This warning is just to inform the user that the type qualifier has no effect, although the code is still legal. The warning is suppressible with --diag_suppress 191. Example:

"`val2 = (const float)val1;`" is equivalent to "`val2 = (float)val1;`"

192: `unrecognized character escape sequence`

This error is commonly associated with the attempted use of non-ASCII character sets, such as 16-bit Unicode characters. The RVCT 3.0 compiler supports multibyte character sets, such as Unicode. Source files are compiled according to the selected locale of that machine. It is possible to use "Escape processing" (as recommended by Kernighan and Richie, section A2.5.2) to encode specific values instead. For example:

```
char *p = "\x12\x34\x56\x78";   // 12 34 56 78
```

In character and string escapes, if the character following the \ has no special meaning, the value of the escape is the character itself, for example, \s is the same as s and the warning will be given.

There is some example code provided with the RVCT tools which can be found in:
"ARM tools directory"\RVDS\Examples\3.x\xx\windows\unicode.

193: `zero used for undefined preprocessing identifier`

194: `expected an asm string`

195: `an asm function must be prototyped`

196: `an asm function may not have an ellipsis`

219: `error while deleting file <filename>: <reason>`

220: `integral value does not fit in required floating-point type`

221: `floating-point value does not fit in required floating-point type`

222: `floating-point operation result is out of range`

223: `function declared implicitly`

This is a common warning that occurs where there is no prototype for a function.
Example:
When `printf()` is used with no `#include <stdio.h>`, the warning occurs:

```
void foo(void)
{
      printf("foo");
```


Copyright © 2006-2008 ARM Limited. All rights reserved.

```
}
```

For ANSI C, this warning can be suppressed with "`--diag_suppress 223`" - useful when compiling old-style C in ANSI C mode.

224: the format string requires additional arguments

225: format string ends before this argument

226: invalid format string conversion

227: macro recursion

228: trailing comma is nonstandard

229: bit field cannot contain all values of the enumerated type

230: nonstandard type for a bit field

In strict ANSI C, the only types allowed for a bit field are `int`, `signed int` and `unsigned int`. Example:

```
struct X{
char y:2;
};
```

231: declaration is not visible outside of function

232: old-fashioned typedef of "void" ignored

233: left operand is not a struct or union containing this field

234: pointer does not point to struct or union containing this field

235: variable <var> was declared with a never-completed type

236: controlling expression is constant

237: selector expression is constant

238: invalid specifier on a parameter

239: invalid specifier outside a class declaration

240: duplicate specifier in declaration

241: a union is not allowed to have a base class

242: multiple access control specifiers are not allowed

243: class or struct definition is missing

244: qualified name is not a member of class <type> or its base classes

245: a nonstatic member reference must be relative to a specific object

246: a nonstatic data member may not be defined outside its class

247: <entity> has already been defined

A typical example of this is where a variable name has been used more than once.
This can sometimes occur when compiling legacy code that relies on tentative declarations.
Tentative declarations allow a variable to be declared and initialised as separate statements, e.g.

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


Copyright © 2006-2008 ARM Limited. All rights reserved.

In RVCT 3.x tentative declarations are allowed by default for C code, but produce an error with C++ code.

248: pointer to reference is not allowed

249: reference to reference is not allowed

250: reference to void is not allowed

251: array of reference is not allowed

252: reference <entity> requires an initializer

253: expected a ","

254: type name is not allowed

This occurs when a typedef name is being used directly in an expression, e.g:

```
typedef int footype;
int x = footype;   // reports Error: #254: type name is not allowed
```

To fix this, create an instance of that type (e.g. a variable of the new type) first, e.g:

```
typedef int footype;
footype bar = 1;
int x = bar;
```

255: type definition is not allowed

256: invalid redeclaration of type name "<entity>"

257: const <entity> requires an initializer

258: "this" may only be used inside a nonstatic member function

259: constant value is not known

260: explicit type is missing ("int" assumed)

261: access control not specified (<xxxx> by default)

262: not a class or struct name

263: duplicate base class name

264: invalid base class

265: <entity> is inaccessible

For C++ only, the "--diag_warning 265" option downgrades access control errors to warnings.

Example:

```
class A { void f() {}; };  // private member
A a;
void g() { a.f(); }        // erroneous access
```

gives:

```
Error:  #265-D: function "A::f" is inaccessible
```

266: "<entity>" is ambiguous

267: old-style parameter list (anachronism)

268: declaration may not appear after executable statement in block

269: conversion to inaccessible base class <type> is not allowed

Copyright © 2006-2008 ARM Limited. All rights reserved.

274: improperly terminated macro invocation

276: name followed by "::" must be a class or namespace name

277: invalid friend declaration

278: a constructor or destructor may not return a value

279: invalid destructor declaration

280: declaration of a member with the same name as its class

281: global-scope qualifier (leading "::") is not allowed

282: the global scope has no <xxxx>

283: qualified name is not allowed

284: NULL reference is not allowed

285: initialization with "{...}" is not allowed for object of type <type>

286: base class <type> is ambiguous

287: derived class <type> contains more than one instance of class <type>

288: cannot convert pointer to base class <type> to pointer to derived class <type>
-- base class is virtual

289: no instance of constructor "<entity>" matches the argument list

290: copy constructor for class <type> is ambiguous

291: no default constructor exists for class <type>

292: <xxxx> is not a nonstatic data member or base class of class <type>

293: indirect nonvirtual base class is not allowed

294: invalid union member -- class <type> has a disallowed member function

296: invalid use of non-lvalue array

297: expected an operator

298: inherited member is not allowed

299: cannot determine which instance of <entity> is intended

300: a pointer to a bound function may only be used to call the function

301: typedef name has already been declared (with same type)

302: <entity> has already been defined

304: no instance of <entity> matches the argument list

305: type definition is not allowed in function return type declaration

306: default argument not at end of parameter list

307: redefinition of default argument

Copyright © 2006-2008 ARM Limited. All rights reserved.

308: more than one instance of <entity> matches the argument list:

309: more than one instance of constructor "<entity>" matches the argument list:

310: default argument of type <type> is incompatible with parameter of type <type>

311: cannot overload functions distinguished by return type alone

312: no suitable user-defined conversion from <type> to <type> exists

313: type qualifier is not allowed on this function

314: only nonstatic member functions may be virtual

315: the object has cv-qualifiers that are not compatible with the member function

316: program too large to compile (too many virtual functions)

317: return type is not identical to nor covariant with return type <type> of overridden virtual function <entity>

318: override of virtual <entity> is ambiguous

319: pure specifier ("= 0") allowed only on virtual functions

320: badly-formed pure specifier (only "= 0" is allowed)

321: data member initializer is not allowed

322: object of abstract class type <type> is not allowed:

323: function returning abstract class <type> is not allowed:

324: duplicate friend declaration

325: inline specifier allowed on function declarations only

326: "inline" is not allowed

327: invalid storage class for an inline function

328: invalid storage class for a class member

329: local class member <entity> requires a definition

330: <entity> is inaccessible

332: class <type> has no copy constructor to copy a const object

333: defining an implicitly declared member function is not allowed

334: class <type> has no suitable copy constructor

335: linkage specification is not allowed

336: unknown external linkage specification

337: linkage specification is incompatible with previous "<entity>"

If the linkage for a function is redeclared with an incompatible specification to a previous declaration this error will be produced.
Example:

```
int foo(void);
```


Copyright © 2006-2008 ARM Limited. All rights reserved.

```
int bar(void)
{
   int x;
   x = foo();
   return x;
}

extern "C" int foo(void)
{
   return 0;
}
```

Gives:

```
Error:  #337: linkage specification is incompatible with previous "foo"
```

338: more than one instance of overloaded function "<entity>" has "C" linkage

339: class <type> has more than one default constructor

340: value copied to temporary, reference to temporary used

341: <operator> must be a member function

342: operator may not be a static member function

343: no arguments allowed on user-defined conversion

344: too many parameters for this operator function

345: too few parameters for this operator function

346: nonmember operator requires a parameter with class type

347: default argument is not allowed

348: more than one user-defined conversion from <type> to <type> applies:

349: no operator <operator> matches these operands

350: more than one operator <operator> matches these operands:

351: first parameter of allocation function must be of type "size_t"

352: allocation function requires "void *" return type

353: deallocation function requires "void" return type

354: first parameter of deallocation function must be of type "void *"

356: type must be an object type

357: base class <type> has already been initialized

358: base class name required -- <type> assumed (anachronism)

359: <entity> has already been initialized

360: name of member or base class is missing

361: assignment to "this" (anachronism)

362: "overload" keyword used (anachronism)

363: invalid anonymous union -- nonpublic member is not allowed

Copyright © 2006-2008 ARM Limited. All rights reserved.

364: invalid anonymous union -- member function is not allowed

365: anonymous union at global or namespace scope must be declared static

366: <entity> provides no initializer for:

367: implicitly generated constructor for class <type> cannot initialize:

368: <entity> defines no constructor to initialize the following:
This indicates that you have a const structure or structure containing a const. It is issued as a "friendly" warning to assist with error 369. This can safely be ignored providing that the const members of structures are appropriately initialised.

369: <entity> has an uninitialized const or reference member
This indicates that you have a instance of a const structure or structure containing a const which has not been correctly initialised. You should either initialise it correctly for every instance or provide a constructor to initialise it.

370: <entity> has an uninitialized const field

371: class <type> has no assignment operator to copy a const object

372: class <type> has no suitable assignment operator

373: ambiguous assignment operator for class <type>

375: declaration requires a typedef name

377: "virtual" is not allowed

378: "static" is not allowed

379: cast of bound function to normal function pointer (anachronism)

380: expression must have pointer-to-member type

381: extra ";" ignored
In C, this can be caused by an unexpected semicolon at the end of a declaration line, for example:
`int x;;`
This may occur inadvertently when using macros.

Similarly, in C++, this may be caused by constructions like:
```
class X { ... } ; ;
```
which probably resulted from some macro usage:
```
#define M(c) class c { ... } ;
    M(X);
```
The extra semicolon is illegal because empty declarations are illegal.

382: nonstandard member constant declaration (standard form is a static const integral member)

384: no instance of overloaded "<entity>" matches the argument list

386: no instance of <entity> matches the required type

387: delete array size expression used (anachronism)

389: a cast to abstract class <type> is not allowed:

390: function "main" may not be called or have its address taken

Copyright © 2006-2008 ARM Limited. All rights reserved.

391: a new-initializer may not be specified for an array

392: member function "<entity>" may not be redeclared outside its class

393: pointer to incomplete class type is not allowed

394: reference to local variable of enclosing function is not allowed

395: single-argument function used for postfix <xxxx> (anachronism)

397: implicitly generated assignment operator cannot copy:

398: cast to array type is nonstandard (treated as cast to <type>)

399: <entity> has an operator newxxxx() but no default operator deletexxxx()

400: <entity> has a default operator deletexxxx() but no operator newxxxx()

401: destructor for base class <type> is not virtual

403: <entity> has already been declared

404: function "main" may not be declared inline

405: member function with the same name as its class must be a constructor

406: using nested <entity> (anachronism)

407: a destructor may not have parameters

408: copy constructor for class <type> may not have a parameter of type <type>

409: <entity> returns incomplete type <type>

410: protected <entity> is not accessible through a <type> pointer or object

411: a parameter is not allowed

412: an "asm" declaration is not allowed here

413: no suitable conversion function from <type> to <type> exists

414: delete of pointer to incomplete class

415: no suitable constructor exists to convert from <type> to <type>

416: more than one constructor applies to convert from <type> to <type>:

417: more than one conversion function from <type> to <type> applies:

418: more than one conversion function from <type> to a built-in type applies:

424: a constructor or destructor may not have its address taken

425: dollar sign ("$") used in identifier

426: temporary used for initial value of reference to non-const (anachronism)

427: qualified name is not allowed in member declaration

428: enumerated type mixed with another type (anachronism)

429: the size of an array in "new" must be non-negative

Copyright © 2006-2008 ARM Limited. All rights reserved.

```
 430: returning reference to local temporary

 432: "enum" declaration is not allowed

 433: qualifiers dropped in binding reference of type <type> to initializer of type
<type>

 434: a reference of type <type> (not const-qualified) cannot be initialized with a
value of type <type>

 435: a pointer to function may not be deleted

 436: conversion function must be a nonstatic member function

 437: template declaration is not allowed here

 438: expected a "<"

 439: expected a ">"

 440: template parameter declaration is missing

 441: argument list for <entity> is missing

 442: too few arguments for <entity>

 443: too many arguments for <entity>

 445: <entity> is not used in declaring the parameter types of <entity>

 449: more than one instance of <entity> matches the required type

 450: the type "long long" is nonstandard

 451: omission of <xxxx> is nonstandard

 452: return type may not be specified on a conversion function

 456: excessive recursion at instantiation of <entity>

 457: <xxxx> is not a function or static data member

 458: argument of type <type> is incompatible with template parameter of type <type>

 459: initialization requiring a temporary or conversion is not allowed

 460: declaration of <xxxx> hides function parameter

 461: initial value of reference to non-const must be an lvalue

 463: "template" is not allowed

 464: <type> is not a class template

 466: "main" is not a valid name for a function template

 467: invalid reference to <entity> (union/nonunion mismatch)

 468: a template argument may not reference a local type

 469: tag kind of <type> is incompatible with declaration of <entity>

 470: the global scope has no tag named <tagname>
```


Copyright © 2006-2008 ARM Limited. All rights reserved.

471: <entity> has no tag member named <membername>

473: <entity> may be used only in pointer-to-member declaration

475: a template argument may not reference a non-external entity

476: name followed by "::~" must be a class name or a type name

477: destructor name does not match name of class <type>

478: type used as destructor name does not match type <type>

479: <entity> redeclared "inline" after being called

481: invalid storage class for a template declaration

484: invalid explicit instantiation declaration

Example:

```
template <class T> struct X { }; // is illegal
```

485: <entity> is not an entity that can be instantiated

486: compiler generated <entity> cannot be explicitly instantiated

487: inline <entity> cannot be explicitly instantiated

489: <entity> cannot be instantiated -- no template definition was supplied

490: <entity> cannot be instantiated -- it has been explicitly specialized

493: no instance of <entity> matches the specified type

494: declaring a void parameter list with a typedef is non-standard

This error may be produced, when the compiler is in ANSI C mode, by a function declaration `f(V)` where V is a void type. In the special syntax `f(<void>)` which indicates that f is a function taking no arguments, the keyword `<void>` is required: the name of a void type cannot be used instead.

496: template parameter <param> may not be redeclared in this scope

497: declaration of <param> hides template parameter

498: template argument list must match the parameter list

500: extra parameter of postfix <operatorxxxx> must be of type "int"

501: an operator name must be declared as a function

502: operator name is not allowed

503: <entity> cannot be specialized in the current scope

504: nonstandard form for taking the address of a member function

505: too few template parameters -- does not match previous declaration

506: too many template parameters -- does not match previous declaration

507: function template for operator delete(void *) is not allowed

508: class template and template parameter may not have the same name

510: a template argument may not reference an unnamed type

Copyright © 2006-2008 ARM Limited. All rights reserved.

511: enumerated type is not allowed

512: type qualifier on a reference type is not allowed

513: a value of type <type> cannot be assigned to an entity of type <type>

514: pointless comparison of unsigned integer with a negative constant

515: cannot convert to incomplete class <type>

516: const object requires an initializer

517: object has an uninitialized const or reference member

518: nonstandard preprocessing directive

519: <entity> may not have a template argument list

520: initialization with "{...}" expected for aggregate object

521: pointer-to-member selection class types are incompatible (<type> and <type>)

522: pointless friend declaration

524: non-const function called for const object (anachronism)

525: a dependent statement may not be a declaration

526: a parameter may not have void type

For example:

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

529: this operator is not allowed in a template argument expression

530: try block requires at least one handler

531: handler requires an exception declaration

532: handler is masked by default handler

533: handler is potentially masked by previous handler for type <type>

534: use of a local type to specify an exception

535: redundant type in exception specification

536: exception specification is incompatible with that of previous <entity>

540: support for exception handling is disabled

541: omission of exception specification is incompatible with previous <entity>

542: could not create instantiation request file <filename>

543: non-arithmetic operation not allowed in nontype template argument

544: use of a local type to declare a nonlocal variable

545: use of a local type to declare a function

546: transfer of control bypasses initialization

Example:

```
int main(void){
```


Copyright © 2006-2008 ARM Limited. All rights reserved.

```
int choice = 1;
int z =1;
switch(choice)
{
        case 1:
           int y = 1;
           z = y + z;
           break;
        case 2:
           break;
}
return 0;
```

Here, 'y' is an initialized variable that is in scope (but unused) in the other cases. The C++ Standard says in section 6.7: "It is possible to transfer into a block, but not in a way that bypasses declarations with initialization. A program that jumps *) from a point where a local variable with automatic storage duration is not in scope to a point where it is in scope is ill-formed unless the variable has POD type (3.9) and is declared without an initializer (8.5)."
*) The transfer from the condition of a switch statement to a case label is considered a jump in this respect.

The usual way to fix this is to enclose the case that declares 'y' in braces:

```
    case 1:
           {
           int y = 1;
           z = y + z;
           }
           break;
```

"y" is a POD (Plain Old Data) type, so an alternative would be to not use initialization:

```
    case 1:
           int y;
           y = 1;
           z = y + z;
           break;
```

548: transfer of control into an exception handler

549: <entity> is used before its value is set

550: <entity> was set but never used

551: <entity> cannot be defined in the current scope

552: exception specification is not allowed

553: external/internal linkage conflict for <entity>

554: <entity> will not be called for implicit or explicit conversions

555: tag kind of <type> is incompatible with template parameter of type <type>

556: function template for operator new(size_t) is not allowed

558: pointer to member of type <type> is not allowed

559: ellipsis is not allowed in operator function parameter list

560: "<entity>" is reserved for future use as a keyword

561: invalid macro definition:

Copyright © 2006-2008 ARM Limited. All rights reserved.

562: invalid macro undefinition:

563: invalid <type> output file <filename>

564: cannot open <type> output file <filename>: <reason>

570: error in debug option argument

571: invalid option:

574: invalid number:

576: invalid instantiation mode:

578: invalid error limit:

585: virtual function tables can only be suppressed when compiling C++

586: anachronism option can be used only when compiling C++

587: instantiation mode option can be used only when compiling C++

588: automatic instantiation mode can be used only when compiling C++

589: implicit template inclusion mode can be used only when compiling C++

590: exception handling option can be used only when compiling C++

593: missing source file name

594: output files may not be specified when compiling several input files

595: too many arguments on command line

596: an output file was specified, but none is needed

598: a template parameter may not have void type

599: excessive recursive instantiation of <entity> due to instantiate-all mode

600: strict ANSI mode is incompatible with allowing anachronisms

601: a throw expression may not have void type

602: local instantiation mode is incompatible with automatic instantiation

603: parameter of abstract class type <type> is not allowed:

604: array of abstract class <type> is not allowed:

605: floating-point template parameter is nonstandard

606: this pragma must immediately precede a declaration

607: this pragma must immediately precede a statement

608: this pragma must immediately precede a declaration or statement

609: this kind of pragma may not be used here

611: overloaded virtual function "<entity>" is only partially overridden in <entity>

612: specific definition of inline template function must precede its first use

Copyright © 2006-2008 ARM Limited. All rights reserved.

613: invalid error tag in diagnostic control option:

614: invalid error number in diagnostic control option:

615: parameter type involves pointer to array of unknown bound

616: parameter type involves reference to array of unknown bound

617: pointer-to-member-function cast to pointer to function

618: struct or union declares no named members

619: nonstandard unnamed field

620: nonstandard unnamed member

Obsolete from 2.2 onwards; 622: invalid precompiled header output file

Obsolete from 2.2 onwards; 623: cannot open precompiled header output file

624: <typename> is not a type name

625: cannot open precompiled header input file <filename>: <reason>

626: precompiled header file <filename> is either invalid or not generated by this version of the compiler

627: precompiled header file <filename> was not generated in this directory

628: header files used to generate precompiled header file <filename> have changed

629: the command line options do not match those used when precompiled header file <filename> was created

630: the initial sequence of preprocessing directives is not compatible with those of precompiled header file <filename>

631: unable to obtain mapped memory: <reason>

632: <xxxx>: using precompiled header file <filename>

633: <xxxx>: creating precompiled header file <filename>

634: memory usage conflict with precompiled header file <filename>

635: invalid PCH memory size

636: PCH options must appear first in the command line

637: insufficient memory for PCH memory allocation

638: precompiled header files may not be used when compiling several input files

639: insufficient preallocated memory for generation of precompiled header file (<number> bytes required)

640: very large entity in program prevents generation of precompiled header file

641: <dir> is not a valid directory

642: cannot build temporary file name

643: "restrict" is not allowed

Copyright © 2006-2008 ARM Limited. All rights reserved.

644: a pointer or reference to function type may not be qualified by "restrict"

645: <xxxx> is an unrecognized __declspec attribute

646: a calling convention modifier may not be specified here

647: conflicting calling convention modifiers

650: calling convention specified here is ignored

651: a calling convention may not be followed by a nested declarator

652: calling convention is ignored for this type

654: declaration modifiers are incompatible with previous declaration

655: the modifier <modifier> is not allowed on this declaration

656: transfer of control into a try block

657: inline specification is incompatible with previous "<entity>"

658: closing brace of template definition not found

659: wchar_t keyword option can be used only when compiling C++

660: invalid packing alignment value

661: expected an integer constant

662: call of pure virtual function

A pure virtual function <pvfn> is being called.

Example:

```
struct T { T(); virtual void pvfn() = 0; };  // a pure virtual function
T::T() { pvfn(); }                           // warning given here
```

By default, this results in a call to the library function __pvfn(), which raises the signal SIGPVFN, which is trapped by the default_signal_handler, which displays "Pure virtual fn called" on the console using semihosting. See RVCT 3.0 Compilers and Libraries Guide, Table 5-19, "Signals used by the C and C++ libraries", and Appendix C.2.

663: invalid source file identifier string

664: a class template cannot be defined in a friend declaration

665: "asm" is not allowed

666: "asm" must be used with a function definition

667: "asm" function is nonstandard

668: ellipsis with no explicit parameters is nonstandard

669: "&..." is nonstandard

670: invalid use of "&..."

672: temporary used for initial value of reference to const volatile (anachronism)

673: a reference of type <type> cannot be initialized with a value of type <type>

674: initial value of reference to const volatile must be an lvalue

Copyright © 2006-2008 ARM Limited. All rights reserved.

676: using out-of-scope declaration of <entity>

678: call of <entity> (declared at line <linenumber>) cannot be inlined

679: <entity> cannot be inlined

680: invalid PCH directory:

688: <xxxx> not found on pack alignment stack

689: empty pack alignment stack

690: RTTI option can be used only when compiling C++

691: <entity>, required for copy that was eliminated, is inaccessible

692: <entity>, required for copy that was eliminated, is not callable because reference parameter cannot be bound to rvalue

693: <typeinfo> must be included before typeid is used

694: <xxxx> cannot cast away const or other type qualifiers

695: the type in a dynamic_cast must be a pointer or reference to a complete class type, or void *

696: the operand of a pointer dynamic_cast must be a pointer to a complete class type

697: the operand of a reference dynamic_cast must be an lvalue of a complete class type

698: the operand of a runtime dynamic_cast must have a polymorphic class type

699: bool option can be used only when compiling C++

701: an array type is not allowed here

702: expected an "="

703: expected a declarator in condition declaration

704: <xxxx>, declared in condition, may not be redeclared in this scope

705: default template arguments are not allowed for function templates

706: expected a "," or ">"

707: expected a template parameter list

708: incrementing a bool value is deprecated

709: bool type is not allowed

710: offset of base class "<entity>" within class "<entity>" is too large

711: expression must have bool type (or be convertible to bool)

712: array new and delete option can be used only when compiling C++

713: <entity> is not a variable name

Copyright © 2006-2008 ARM Limited. All rights reserved.

717: the type in a const_cast must be a pointer, reference, or pointer to member to an object type

718: a const_cast can only adjust type qualifiers; it cannot change the underlying type

719: mutable is not allowed

720: redeclaration of <entity> is not allowed to alter its access

Obsolete from 2.2 onwards; 721: nonstandard format string conversion

722: use of alternative token "<:" appears to be unintended

723: use of alternative token "%:" appears to be unintended

724: namespace definition is not allowed

725: name must be a namespace name

726: namespace alias definition is not allowed

727: namespace-qualified name is required

728: a namespace name is not allowed

730: <entity> is not a class template

731: array with incomplete element type is nonstandard

732: allocation operator may not be declared in a namespace

733: deallocation operator may not be declared in a namespace

734: “<entity>“ conflicts with using-declaration of <entity>

735: using-declaration of <entity> conflicts with <entity>

736: namespaces option can be used only when compiling C++

737: using-declaration ignored -- it refers to the current namespace

738: a class-qualified name is required

742: <entity> has no actual member <member>

744: incompatible memory attributes specified

745: memory attribute ignored

746: memory attribute may not be followed by a nested declarator

747: memory attribute specified more than once

748: calling convention specified more than once

749: a type qualifier is not allowed

750: <entity> (declared at line <linenumber>) was used before its template was declared

751: static and nonstatic member functions with same parameter types cannot be overloaded

Copyright © 2006-2008 ARM Limited. All rights reserved.

752: no prior declaration of <entity>

753: a template-id is not allowed

754: a class-qualified name is not allowed

755: <entity> may not be redeclared in the current scope

756: qualified name is not allowed in namespace member declaration

757: <entity> is not a type name

758: explicit instantiation is not allowed in the current scope

759: <entity> cannot be explicitly instantiated in the current scope

760: <entity> explicitly instantiated more than once

761: typename may only be used within a template

763: typename option can be used only when compiling C++

764: implicit typename option can be used only when compiling C++

765: nonstandard character at start of object-like macro definition

766: exception specification for virtual <entity> is incompatible with that of overridden <entity>

767: conversion from pointer to smaller integer

768: exception specification for implicitly declared virtual <entity> is incompatible with that of overridden <entity>

769: "<entity>", implicitly called from <entity>, is ambiguous

770: option "explicit" can be used only when compiling C++

771: "explicit" is not allowed

772: declaration conflicts with <xxxx> (reserved class name)

773: only "()" is allowed as initializer for array <entity>

774: "virtual" is not allowed in a function template declaration

775: invalid anonymous union -- class member template is not allowed

776: template nesting depth does not match the previous declaration of <entity>

777: this declaration cannot have multiple "template <...>" clauses

778: option to control the for-init scope can be used only when compiling C++

779: <xxxx>, declared in for-loop initialization, may not be redeclared in this scope

780: reference is to <entity> (declared at line <linenumber>) -- under old for-init scoping rules it would have been <entity>

781: option to control warnings on for-init differences can be used only when compiling C++

782: definition of virtual <entity> is required here

Copyright © 2006-2008 ARM Limited. All rights reserved.

783: empty comment interpreted as token-pasting operator "##"

784: a storage class is not allowed in a friend declaration

785: template parameter list for "<entity>" is not allowed in this declaration

786: <entity> is not a valid member class or function template

787: not a valid member class or function template declaration

788: a template declaration containing a template parameter list may not be followed by an explicit specialization declaration

789: explicit specialization of <entity> must precede the first use of <entity>

790: explicit specialization is not allowed in the current scope

791: partial specialization of <entity> is not allowed

792: <entity> is not an entity that can be explicitly specialized

793: explicit specialization of <entity> must precede its first use

794: template parameter <param> may not be used in an elaborated type specifier

795: specializing <entity> requires "template<>" syntax

798: option "old_specializations" can be used only when compiling C++

799: specializing <entity> without "template<>" syntax is nonstandard

800: this declaration may not have extern "C" linkage

801: <xxxx> is not a class or function template name in the current scope

802: specifying a default argument when redeclaring an unreferenced function template is nonstandard

803: specifying a default argument when redeclaring an already referenced function template is not allowed

804: cannot convert pointer to member of base class <type> to pointer to member of derived class <type> -- base class is virtual

805: exception specification is incompatible with that of <entity>

806: omission of exception specification is incompatible with <entity>

807: unexpected end of default argument expression

808: default-initialization of reference is not allowed

809: uninitialized <entity> has a const member

810: uninitialized base class <type> has a const member

811: const <entity> requires an initializer -- class <type> has no explicitly declared default constructor

812: const object requires an initializer -- class <type> has no explicitly declared default constructor

814: strict ANSI mode is incompatible with long preserving rules

Copyright © 2006-2008 ARM Limited. All rights reserved.

815: type qualifier on return type is meaningless

For example:

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

__packed is ignored here because the return type cannot be __packed.

816: in a function definition a type qualifier on a "void" return type is not allowed

817: static data member declaration is not allowed in this class

818: template instantiation resulted in an invalid function declaration

819: "..." is not allowed

821: extern inline <entity> was referenced but not defined

822: invalid destructor name for type <type>

824: destructor reference is ambiguous -- both <entity> and <entity> could be used

825: virtual inline <entity> was never defined

826: <entity> was never referenced

827: only one member of a union may be specified in a constructor initializer list

828: support for "new[]" and "delete[]" is disabled

829: "double" used for "long double" in generated C code

830: <entity> has no corresponding operator deletexxxx (to be called if an exception is thrown during initialization of an allocated object)

831: support for placement delete is disabled

832: no appropriate operator delete is visible

833: pointer or reference to incomplete type is not allowed

834: invalid partial specialization -- <entity> is already fully specialized

835: incompatible exception specifications

836: returning reference to local variable

837: omission of explicit type is nonstandard ("int" assumed)

A function has been declared or defined with no return type.

Example:

```
foo(void){
int a;
}
```

An `int` result will be assumed. If you want it to return no result, use `void` as the return type. This is widespread in old-style C.

The "`--diag_suppress 837`" option suppresses this warning.

838: more than one partial specialization matches the template argument list of <entity>

840: a template argument list is not allowed in a declaration of a primary template

Copyright © 2006-2008 ARM Limited. All rights reserved.

841: partial specializations may not have default template arguments

842: <entity> is not used in template argument list of <entity>

Obsolete from 2.2 onwards; 843: the type of partial specialization template parameter <entity> depends on another template parameter

844: the template argument list of the partial specialization includes a nontype argument whose type depends on a template parameter

845: this partial specialization would have been used to instantiate <entity>

846: this partial specialization would have been made the instantiation of <entity> ambiguous

847: expression must have integral or enum type

848: expression must have arithmetic or enum type

849: expression must have arithmetic, enum, or pointer type

850: type of cast must be integral or enum

851: type of cast must be arithmetic, enum, or pointer

852: expression must be a pointer to a complete object type

854: a partial specialization nontype argument must be the name of a nontype parameter or a constant

855: return type is not identical to return type <type> of overridden virtual function <entity>

856: option "guiding_decls" can be used only when compiling C++

857: a partial specialization of a class template must be declared in the namespace of which it is a member

858: <entity> is a pure virtual function

859: pure virtual <entity> has no overrider

860: __declspec attributes ignored

861: invalid character in input line

862: function returns incomplete type <type>

863: effect of this "#pragma pack" directive is local to <entity>

864: <xxxx> is not a template

865: a friend declaration may not declare a partial specialization

866: exception specification ignored

867: declaration of "size_t" does not match the expected type <type>

868: space required between adjacent ">" delimiters of nested template argument lists (">>" is the right shift operator)

869: could not set locale <xxxx> to allow processing of multibyte characters

870: invalid multibyte character sequence

Copyright © 2006-2008 ARM Limited. All rights reserved.

871: template instantiation resulted in unexpected function type of <type> (the meaning of a name may have changed since the template declaration -- the type of the template is <type>)

872: ambiguous guiding declaration -- more than one function template "<entity>" matches type <type>

873: non-integral operation not allowed in nontype template argument

884: pointer-to-member representation <xxxx> has already been set for <entity>

885: <type> cannot be used to designate constructor for <type>

886: invalid suffix on integral constant

888: invalid GUID string in __declspec(uuid("..."))

889: option "vla" can be used only when compiling C

890: variable length array with unspecified bound is not allowed

891: an explicit template argument list is not allowed on this declaration

892: an entity with linkage cannot have a type involving a variable length array

893: a variable length array cannot have static storage duration

894: <entity> is not a template

896: expected a template argument

898: nonmember operator requires a parameter with class or enum type

901: qualifier of destructor name <type> does not match type <type>

902: type qualifier ignored

912: ambiguous class member reference -- <entity> (declared at line <linenumber>) used in preference to <entity> (declared at line <linenumber>)

915: a segment name has already been specified

916: cannot convert pointer to member of derived class <type> to pointer to member of base class <type> -- base class is virtual

917: invalid directory for instantiation files:

919: invalid output file: <filename>

920: cannot open output file: <filename>

921: an instantiation information file name may not be specified when compiling several input files

923: more than one command line option matches the abbreviation "--<command>"

925: type qualifiers on function types are ignored

926: cannot open definition list file: "<filename>"

928: incorrect use of va_start

929: incorrect use of va_arg

Copyright © 2006-2008 ARM Limited. All rights reserved.

930: incorrect use of va_end

931: pending instantiations option can be used only when compiling C++

932: invalid directory for #import files:

934: a member with reference type is not allowed in a union

935: "typedef" may not be specified here

936: redeclaration of <entity> alters its access

937: a class or namespace qualified name is required

938: return type "int" omitted in declaration of function "main"

`main()` has been declared or defined with no return type.

Example:

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

If compiled with --strict the compiler reports this as an error.

If you want it to return no result, use `void` as the return type. This is widespread in old-style C. For ANSI C, the "`--diag_suppress 938`" option suppresses this warning. For C++, this always results in an error.

939: pointer-to-member representation <xxxx> is too restrictive for <entity-kind>

940: missing return statement at end of non-void <entity-kind>

A return type has been defined for a function, but no value is returned. Example:

```
int foo(int a)
{
      printf("Hello %d", a);
}
```

941: duplicate using-declaration of "<entity>" ignored

942: enum bit-fields are always unsigned, but enum <type> includes negative enumerator

943: option "class_name_injection" can be used only when compiling C++

944: option "arg_dep_lookup" can be used only when compiling C++

945: option "friend_injection" can be used only when compiling C++

946: name following "template" must be a member template

948: nonstandard local-class friend declaration -- no prior declaration in the enclosing scope

949: specifying a default argument on this declaration is nonstandard

951: return type of function "main" must be "int"

952: a nontype template parameter may not have class type

953: a default template argument cannot be specified on the declaration of a member of a class template outside of its class

Copyright © 2006-2008 ARM Limited. All rights reserved.

954: a return statement is not allowed in a handler of a function try block of a constructor

955: ordinary and extended designators cannot be combined in an initializer designation

956: the second subscript must not be smaller than the first

959: declared size for bit field is larger than the size of the bit field type; truncated to <number> bits

960: type used as constructor name does not match type <type>

961: use of a type with no linkage to declare a variable with linkage

962: use of a type with no linkage to declare a function

963: return type may not be specified on a constructor

964: return type may not be specified on a destructor

965: incorrectly formed universal character name

966: universal character name specifies an invalid character

967: a universal character name cannot designate a character in the basic character set

968: this universal character is not allowed in an identifier

969: the identifier __VA_ARGS__ can only appear in the replacement lists of variadic macros

970: the qualifier on this friend declaration is ignored

971: array range designators cannot be applied to dynamic initializers

972: property name cannot appear here

973: "inline" used as a function qualifier is ignored

975: a variable-length array type is not allowed

976: a compound literal is not allowed in an integral constant expression

977: a compound literal of type <type> is not allowed

978: a template friend declaration cannot be declared in a local class

979: ambiguous "?" operation: second operand of type <type> can be converted to third operand type <type>, and vice versa

980: call of an object of a class type without appropriate operator() or conversion functions to pointer-to-function type

982: there is more than one way an object of type <type> can be called for the argument list:

983: typedef name has already been declared (with similar type)

984: operator new and operator delete cannot be given internal linkage

985: storage class "mutable" is not allowed for anonymous unions

Copyright © 2006-2008 ARM Limited. All rights reserved.

```
 986: invalid precompiled header file

 987: abstract class type <type> is not allowed as catch type:

 988: a qualified function type cannot be used to declare a nonmember function or a
static member function

 989: a qualified function type cannot be used to declare a parameter

 990: cannot create a pointer or reference to qualified function type

 991: extra braces are nonstandard

 992: invalid macro definition:
```
Incorrect use of –D on the compile line, for example, "`-D##`"

```
 993: subtraction of pointer types <type> and <type> is nonstandard

 994: an empty template parameter list is not allowed in a template template
parameter declaration

 995: expected "class"

 996: the "class" keyword must be used when declaring a template template parameter

 997: <entity> is hidden by "<entity>" -- virtual function override intended?

 998: a qualified name is not allowed for a friend declaration that is a function
definition

 999: <entity> is not compatible with <entity>

1000: a storage class may not be specified here

1001: class member designated by a using-declaration must be visible in a direct base
class

1006: a template template parameter cannot have the same name as one of its template
parameters

1007: recursive instantiation of default argument

1009: <entity> is not an entity that can be defined

1010: destructor name must be qualified

1011: friend class name may not be introduced with "typename"

1012: a using-declaration may not name a constructor or destructor

1013: a qualified friend template declaration must refer to a specific previously
declared template

1014: invalid specifier in class template declaration

1015: argument is incompatible with formal parameter

1016: prefix form of ARM function qualifier not permitted in this position

1017: Duplicate ARM function qualifiers not permitted

1018: ARM function qualifiers not permitted on this declaration/definition
```
"ARM function qualifiers" include qualifiers such as __swi, __pure and __irq amongst others.

Copyright © 2006-2008 ARM Limited. All rights reserved.
```
```

For more information refer to Chapter 3 Compilers and Libraries Guide: 3.1.2.

1019: function qualifier <type> not permitted on a non-static member function

1020: __irq functions must take no arguments

1021: __irq functions must return no result

1022: cannot have pointer nor reference to <xxxx> function

1023: __global_reg not allowed on this declaration

1024: invalid global register number; 1 to 8 allowed

An invalid register is being used in "__global_reg".
For Example:

```
__global_reg(786) int x;
```

1025: __swi parameter <param> is not within permitted range (0 to 0xffffff) for ARM SWI instruction

SWI numbers are limited to the range 0 to 0xffffff for the ARM compilers, and 0 to 0xFF for the Thumb compilers. For standard "semihosting" SWI's, 0x123456 is used for ARM, 0xAB is used for Thumb.

1026: taking the address of a global register variable is not allowed

1027: __swi_indirect function must have arguments

1028: conflicting global register declaration with <entity>

1029: __packed ignored for non-pointer parameter

1030: <xxxx> <type> previously declared without __packed

1031: Definition of "<type>" in packed "<type>" must be __packed

The RVCT 3.0 Compiler Guide, section 3.1.5, 'Type qualifiers', says:
"All substructures of a packed structure must be declared using __packed."
This rule applies for all releases of RVCT, ADS and the earlier SDT 2.5x.

The compiler will fault a non-packed child structure contained in a packed parent structure. This includes the case where the substructure is an array, for example:

```
typedef struct ChildStruct {
   int a;
} ChildStruct;

typedef __packed struct ParentStruct {
  ChildStruct child[1];
} ParentStruct;
```

correctly gives:

```
Error:  #1031: Definition of "ChildStruct" in packed "ParentStruct" must be
__packed
```

1032: Definition of nested anonymous <xxxx> in packed "<type>" must be __packed

1033: "<xxxx>" incompatible with function definition

1034: __irq functions must not be the target of a function call

1037: __global_reg is not valid on this declaration

Copyright © 2006-2008 ARM Limited. All rights reserved.

1038: invalid alignment specified; only integer powers of 2 allowed

1039: conflicting alignment declaration with <entity>

1040: under-alignment not allowed

1041: alignment for an auto object may not be larger than 8

For example:

```
int main(void){
__align(16) int foo = 10;
}
```

is not allowed for a local variable foo, so the error is given.

1042: <entity> (declared at line <linenumber>) cannot be dynamically initialized when compiled position independent

1043: <entity> cannot be const because it contains a mutable member

1044: option "dep_name" can be used only when compiling C++

1045: loop in sequence of "operator->" functions starting at class <type>

1046: <entity> has no member class "<class>"

1047: the global scope has no class named "<class>"

1048: recursive instantiation of template default argument

1049: access declarations and using-declarations cannot appear in unions

1050: "<entity>" is not a class member

1051: nonstandard member constant declaration is not allowed

1053: option "parse_templates" can be used only when compiling C++

1054: option "dep_name" cannot be used with "no_parse_templates"

1055: language modes specified are incompatible

1056: invalid redeclaration of nested class

1057: type containing an unknown-size array is not allowed

1058: a variable with static storage duration cannot be defined within an inline function

1059: an entity with internal linkage cannot be referenced within an inline function with external linkage

1060: argument type <type> does not match this type-generic function macro

1062: friend declaration cannot add default arguments to previous declaration

1063: <entity> cannot be declared in this scope

1064: the reserved identifier "<xxxx>" may only be used inside a function

1065: this universal character cannot begin an identifier

1066: expected a string literal

1070: incorrect use of va_copy

Copyright © 2006-2008 ARM Limited. All rights reserved.

1071: <xxxx> can only be used with floating-point types

1072: complex type is not allowed

1073: invalid designator kind

1074: floating-point value cannot be represented exactly

1075: complex floating-point operation result is out of range

1076: conversion between real and imaginary yields zero

1077: an initializer cannot be specified for a flexible array member

1078: imaginary *= imaginary sets the left-hand operand to zero

1079: standard requires that <entity> be given a type by a subsequent declaration ("int" assumed)

1080: a definition is required for inline <entity>

1081: conversion from integer to smaller pointer

1082: a floating-point type must be included in the type specifier for a _Complex or _Imaginary type

_Complex and _Imaginary are not yet supported in RVCT although they are part of the C99 Standard.

1083: Inline assembler syntax error

1084: This instruction not permitted in inline assembler

1085: Missing operand

1086: Operand is wrong type

1087: Operand should be constant

1088: Wrong number of operands

1089: Invalid PSR operand

1090: Expected PSR operand

1091: Invalid shift specified

1092: Should be acc0

1093: Must be a modifiable lvalue

1094: Expected a register expression

1095: Expected a label or function name

1096: Instruction cannot be conditional

1097: Expected a [or]

1098: Expected a shift operation

1099: Unexpected]

1100: Register specified shift not allowed

Copyright © 2006-2008 ARM Limited. All rights reserved.

1101: Pre-Indexed addressing not allowed

1102: Post-Indexed addressing not allowed

1103: Writeback not allowed in the addressing mode

1104: Expected {

1105: Expected }

1106: Too many registers in register list

1107: Only ^ valid here

1108: Cannot mix virtual register and C/C++ expressions in register list

1109: Only virtual registers can be specified in a register range

1110: User mode register selection/CPSR update not supported in inline assembler. Use embedded assembler or out-of-line assembler

1111: Expected a coprocessor name
1112: Expected a coprocessor register name

These errors are given by the inline assembler if the coprocessor number is accidentally omitted from an MCR or MRC instruction, or if an invalid coprocessor number/coprocessor register number has been given. A correct use is shown below:

```
void foo()
{
  int reg0;
  __asm
  {
    MRC  p15, 0, reg0, c1, c0, 0
  }
}
```

1113: Inline assembler not permitted when generating Thumb code

The Thumb inline assembler was supported in ADS, but support was withdrawn in RVCT 2.0. ARM inline assembly continues to be supported. The Thumb Instruction Set was designed based on the output of the C compiler, and so there should be no need to write explicitly in Thumb inline assembler. Alternatively use the embedded assembler which can use Thumb code.

1114: This instruction not supported on target architecture/processor

Example when compiled with "armcc -cpu 4T".

```
int main(void) {
int a,b,c;
      __asm {
            QADD a,b,c
      }
      return(a);
}
```

This is because the saturated add instruction is only supported in Architectures 5ET and above.

1115: Cannot assign to const operand

1116: Register list cannot be empty

1117: Unqualified virtual function not allowed

Copyright © 2006-2008 ARM Limited. All rights reserved.

1118: Expected a newline

1119: Reference to static variable not allowed in __asm function

1120: Reference to static function not allowed in __asm function

1121: Pointer to data member not allowed in __asm function

1122: __asm function cannot have static qualifier

1123: base class <type> is a virtual base class of <type>

1124: base class <type> is not virtual base class of <type>

1125: <entity> has no member function "<func>"

1126: "__asm" is not allowed in this declaration

1127: Member initializer list not permitted for __asm constructors

1128: try block not permitted for __asm constructors

1129: Order of operands not compatible with previous compiler versions

1130: __align not permitted in typedef

1131: Non portable instruction (LDM with writeback and base in reg. list, final value of base unpredictable)

1132: Non portable instruction (STM with writeback and base not first in reg. list, stored value of base unpredictable)

1133: Expression operands not permitted with virtual base register

1134: literal treated as "long long"

The constant <Number> is too large to be represented in a signed long, and therefore has been treated as a (signed) long long

Example:

```
int foo(unsigned int bar)
{
   return (bar == 2147483648);
}
```

gives a warning because 2147483648 is one greater than the maximum value allowed for a signed long. The "ll" suffix means that the constant will be treated as a (64-bit) "long long" type rather than a signed long. See section 3.2.1 of the RVCT 3.0 Compilers and Libraries Guide.

To eliminate the warning, explicitly add the "ll" or "LL" suffix to your constants, e.g.:

```
int foo(unsigned int bar)
{
   return (bar == 2147483648LL);
}
```

1135: literal treated as "unsigned long long"

The constant <Number> is too large to be represented in a signed long long, and therefore has been given type unsigned long long.

1137: Expected a comma

1138: Unexpected comma after this expression

Copyright © 2006-2008 ARM Limited. All rights reserved.

1139: MRRC operation opcode must lie in range 0-15

1140: MCRR operation opcode must lie in range 0-15

1141: CDP operation opcode must lie in range 0-15

1142: MRC operation opcode must lie in range 0-7

1143: MCR operation opcode must lie in range 0-7

1144: opcode_2 must lie in range 0-7

1145: LDC/STC extra opcode must lie in range 0-255

1146: LDC/STC offset must lie in range -1020 to +1020 and be word aligned

1147: Constant operand out of range

1148: floating-point operator is not permitted with -fpu none

1149: floating-point return type in function definition is not permitted with -fpu none

1150: floating-point parameter type in function definition is not permitted with -fpu none

1151: floating-point variable definition with initialiser is not permitted with -fpu none

1152: polymorphic base classes need to be exported as well

1153: Cannot assign physical registers in this register list

1154: Can only specify an even-numbered physical register here

1155: Can only specify an assignment to a physical register here

1156: Can only specify an assignment from a physical register here

1157: Can only specify physical registers in a corrupted register list

1158: PSR operand not valid here

1159: Expected an unambiguous label or function name

1160: Calls to destructors for temporaries will overwrite the condition flags updated by this instruction

1161: Cannot directly modify the stack pointer SP (r13)

1162: Cannot directly modify the link register LR (r14)

1163: Cannot directly modify the program counter PC (r15)

1164: Offset must be word-aligned

1165: types cannot be declared in anonymous unions

1166: returning pointer to local variable

1167: returning pointer to local temporary

1168: option "export" can be used only when compiling C++

Copyright © 2006-2008 ARM Limited. All rights reserved.

1169: option "export" cannot be used with "no_dep_name"

1170: option "export" cannot be used with "implicit_include"

1171: declaration of <entity> is incompatible with a declaration in another translation unit

1172: the other declaration is at line <linenumber>

1175: a field declaration cannot have a type involving a variable length array

1176: declaration of <entity> had a different meaning during compilation of "<xxxx>"

1177: expected "template"

1178: "export" cannot be used on an explicit instantiation

1179: "export" cannot be used on this declaration

1180: a member of an unnamed namespace cannot be declared "export"

1181: a template cannot be declared "export" after it has been defined

1182: a declaration cannot have a label

1183: support for exported templates is disabled

1184: cannot open exported template file: "<filename>"

1185: <entity> already defined during compilation of "<xxxx>"

1186: <entity> already defined in another translation unit

1188: the option to list makefile dependencies may not be specified when compiling more than one translation unit

1190: the option to generate preprocessed output may not be specified when compiling more than one translation unit

1191: a field with the same name as its class cannot be declared in a class with a user-declared constructor

1192: "implicit_include" cannot be used when compiling more than one translation unit

1193: exported template file "<filename>" is corrupted

1194: <entity> cannot be instantiated -- it has been explicitly specialized in the translation unit containing the exported definition

1196: the object has cv-qualifiers that are not compatible with the member <entity>

1197: no instance of <entity> matches the argument list and object (the object has cv-qualifiers that prevent a match)

1198: an attribute specifies a mode incompatible with <type>

1199: there is no type with the width specified

1200: invalid alignment value specified by attribute

1201: invalid attribute for <type>

1202: invalid attribute for <entity>

Copyright © 2006-2008 ARM Limited. All rights reserved.

1203: invalid attribute for parameter

1204: attribute "<attribute>" does not take arguments

1206: expected an attribute name

1207: attribute "<attribute>" ignored

1208: attributes may not appear here

1209: invalid argument to attribute "<attribute>"

1210: the "packed" attribute is ignored in a typedef

1211: in "goto *expr", expr must have type "void *"

1212: "goto *expr" is nonstandard

1213: taking the address of a label is nonstandard

1214: file name specified more than once:

1215: #warning directive: <xxxx>

1216: attribute "<attribute>" is only allowed in a function definition

1217: the "transparent_union" attribute only applies to unions, and <type> is not a union

1218: the "transparent_union" attribute is ignored on incomplete types

1219: <type> cannot be transparent because <entity> does not have the same size as the union

1220: <type> cannot be transparent because it has a field of type <type> which is not the same size as the union

1221: only parameters can be transparent

1222: the "<attribute>" attribute does not apply to local variables

1224: attributes are not permitted in a function definition

1225: declarations of local labels should only appear at the start of statement expressions

1226: the second constant in a case range must be larger than the first

1227: an asm name is not permitted in a function definition

1228: an asm name is ignored in a typedef

1229: unknown register name "<xxxx>"

1230: modifier letter '<xxxx>' ignored in asm operand

1231: unknown asm constraint modifier '<xxxx>'

1232: unknown asm constraint letter '<xxxx>'

1233: asm operand has no constraint letter

1234: an asm output operand must have one of the '=' or '+' modifiers

Copyright © 2006-2008 ARM Limited. All rights reserved.

1235: an asm input operand may not have the '=' or '+' modifiers

1236: too many operands to asm statement (maximum is 10)

1237: too many colons in asm statement

1238: register "<register>" used more than once

1239: register "<register>" is both used and clobbered

1240: register "<register>" clobbered more than once

1241: register "<register>" has a fixed purpose and may not be used in an asm statement

1242: register "<register>" has a fixed purpose and may not be clobbered in an asm statement

1243: an empty clobbers list must be omitted entirely

1244: expected an asm operand

1245: expected a register to clobber

1246: "format" attribute applied to <entity> which does not have variable arguments

1247: first substitution argument is not the first variable argument

1248: format argument index is greater than number of parameters

1249: format argument does not have string type

1250: the "template" keyword used for syntactic disambiguation may only be used within a template

1253: attribute does not apply to non-function type <type>

1254: arithmetic on pointer to void or function type

1255: storage class must be auto or register

1256: <type> would have been promoted to <type> when passed through the ellipsis parameter; use the latter type instead

1257: "<xxxx>" is not a base class member

1262: mangled name is too long

1263: Offset must be half-word aligned

1264: Offset must be double-word aligned

1265: converting to and from floating-point type is not permitted with -fpu none

1266: Operand should be a constant expression

1267: Implicit physical register <register> should be defined as a variable

1268: declaration aliased to unknown entity "<xxxx>"

1269: declaration does not match its alias <entity>

1270: entity declared as alias cannot have definition

Copyright © 2006-2008 ARM Limited. All rights reserved.

1271: variable-length array field type will be treated as zero-length array field type

1272: nonstandard cast on lvalue not supported

1273: unrecognized flag name

1274: void return type cannot be qualified

1275: the auto specifier is ignored here (invalid in standard C/C++)

1276: a reduction in alignment without the "packed" attribute is ignored

1277: a member template corresponding to "<entity>" is declared as a template of a different kind in another translation unit

1278: excess initializers are ignored

1279: va_start should only appear in a function with an ellipsis parameter

1282: variable <var> cannot be used in a register range

1283: A physical register name is required here

1284: A register range cannot be specified here

1285: Implicit physical register <register> has not been defined

1286: LDRD/STRD instruction will be expanded
When LDRD and STRD instructions are used in inline assembler the compiler will expand these into two LDR or STR instructions before being passsed through the compiler optimization stage. The optimization stage will normally combine the two LDR or STR instruction back into a single LDRD or STRD instruction, however it is possible in some cases that a LDRD or STRD will not be used.

1287: LDM/STM instruction may be expanded
When LDM and STM instructions are used in inline assembler the compiler will expand these into a number of LDR or STR instructions before being passsed through the compiler optimization stage. The optimization stage will normally combine the two LDR or STR instruction back into LDM or STM instruction(s), however it is possible that in some cases that a single LDM or STM instruction will not be used.

1288: Implicit ARM register "<register>" was not defined due to name clash

1289: statement expressions are only allowed in block scope

1291: an asm name is ignored on a non-register automatic variable

1292: inline function also declared as an alias; definition ignored

1293: assignment in condition
In a context where a boolean value is required (the controlling expression for <if>, <while>, <for> or the first operand of a conditional expression, an expression contains one of:
- a bitwise not operator (~). It is likely that a logical not operator (!) was intended.
- an assignment operator (=). This could be a mistyped equality operator (==).
In either case if the operator is intended adding an explicit comparison against 0 may suppress the warning. This warning can be suppressed with the "--diag_suppress 1293" option.

Example:

Copyright © 2006-2008 ARM Limited. All rights reserved.

```
int main(void)
{
int a,b;
if (a=b)
}
```

1294: `Old-style function <function>`

The compilers accept both old-style and new-style function declarations.
The difference between an old-style and a new-style function declaration is as follows.

```
// new style
int add2(int a, int b)
{
  return a+b;
}

// old style
int oldadd2(a,b)
  int a;
  int b;
{
  return a+b;
}
```

When compiling old style functions in C mode the compiler reports:
`Warning:  #1294-D: Old-style function oldadd2`

1295: `Deprecated declaration <xxxx> - give arg types`

This warning is normally given when a declaration without argument types is encountered in ANSI C mode. In ANSI C, declarations like this are deprecated. However, it is sometimes useful to suppress this warning with the "`--diag_suppress 1295`" option when porting old code. In C++, `void foo();` means `void foo(void);` and no warning is generated.

1296: `extended constant initialiser used`

The expression used as a constant initialiser may not be portable.
This warns that there is a constant that does not follow the strict rules of ANSI C even though there is a clause to allow it in the ANSI C specification.
Example compiled with --c90 switch:

```
const int foo_table[]  = { (int)"foo", 0, 1, 2};
```

This is not ANSI C standard compliant. Compiling with "`--diag_suppress 1296`" will suppress the warning.

1297: `Header file not guarded against multiple inclusion`

This warning is given when an unguarded header file is `#included`.
An unguarded header file is a header file not wrapped in a declaration such as:

```
#ifdef foo_h
#define foo_h
/* body of include file */
#endif
```

This warning is off by default. It can be enabled with "`--diag_warning 1297`".

1298: `Header file is guarded by '<xxxx>', but does not #define it`

Example:

```
#ifndef MYHEADER_H
//#define MYHEADER_H
#endif
```

To correct the code remove the comment slashes (//). This warning is off by default. It can be enabled with "`--diag_warning 1298`".

Copyright © 2006-2008 ARM Limited. All rights reserved.

1299: members and base-classes will be initialized in declaration order, not in member initialisation list order

1300: <xxxx> inherits implicit virtual

This warning is issued when a non-virtual member function of a derived class hides a virtual member of a parent class. For example:

```
struct Base { virtual void f(); };
struct Derived : Base { void f(); };
```

gives:

```
Warning:  #1300-D: f inherits implicit virtual
  struct Derived : Base { void f(); };
                                ^
```

Adding the `virtual` keyword in the derived class prevents the warning. For C++, the "`--diag_suppress 1300`" option suppresses the implicit virtual warning.

1301: padding inserted in struct <struct>

For the members of the structure to be correctly aligned, some padding has been inserted between members. This warning is off by default and can be enabled with "--diag_warning 1301" or generated by adding “--remarks” to the end of the command line.

Example:

```
struct X {
char x;
int y;
}
```

1302: type too large to be returned in registers - __value_in_regs ignored

1303: using --force_new_nothrow: added "throw()"

1304: operator new missing exception specification

1305: using --force_new_nothrow: added "(::std::nothrow)"

Obsolete from 2.2 onwards; 1306: cannot open ASM output file

1307: floating point argument not permitted with -fpu none

1308: Base class <type> of __packed class <type> must be __packed

1310: shared block size does not match one previously specified

1311: bracketed expression is assumed to be a block size specification rather than an array dimension

1312: the block size of a shared array must be greater than zero

1313: multiple block sizes not allowed

1314: strict or relaxed requires shared

1316: block size specified exceeds the maximum value of <val>

1317: function returning shared is not allowed

1320: shared type inside a struct or union is not allowed

1321: parameters may not have shared types

1323: shared variables must be static or extern

Copyright © 2006-2008 ARM Limited. All rights reserved.

1327: affinity expression must have a shared type or point to a shared type

1328: affinity has shared type (not pointer to shared)

1329: shared void* types can only be compared for equality

1331: null (zero) character in input line ignored

1332: null (zero) character in string or character constant

1333: null (zero) character in header name

1334: declaration in for-initializer hides a declaration in the surrounding scope

1335: the hidden declaration is at line <linenumber>

1336: the prototype declaration of <entity> (declared at line <linenumber>) is ignored after this unprototyped redeclaration

1338: <entity> (declared at line <linenumber>) must have external C linkage

1339: variable declaration hides declaration in for-initializer

1340: typedef "<xxxx>" may not be used in an elaborated type specifier

1341: call of zero constant ignored

1342: parameter "<param>" may not be redeclared in a catch clause of function try block

1343: the initial explicit specialization of <entity> must be declared in the namespace containing the template

1345: "template" must be followed by an identifier

1347: layout qualifier cannot qualify pointer to shared

1348: layout qualifier cannot qualify an incomplete array

1349: declaration of "<xxxx>" hides handler parameter

1350: nonstandard cast to array type ignored

1351: this pragma cannot be used in a _Pragma operator (a #pragma directive must be used)

1352: field uses tail padding of a base class

1353: GNU C++ compilers may use bit field padding

1354: memory mapping conflict with precompiled header file <filename>

1355: abstract class <type> has a non-virtual destructor, calling delete on a pointer to this class is undefined behaviour

1356: an asm name is not allowed on a nonstatic member declaration

1357: static initialisation of <entity> using address of <xxxx> may cause link failure --ropi
See 1359

1358: static initialisation of extern const <entity> using address of <xxxx> cannot be lowered for ROPI

Copyright © 2006-2008 ARM Limited. All rights reserved.

1359: static initialisation of <entity> using address of <xxxx> may cause link failure --rwpi

Warnings 1357 and 1359 warn against the use of non-PI code constructs and that a subsequent link step may fail. For example:

```
char *str = "test";  /* global pointer */
```

when compiled with `--apcs /ropi` gives:

```
Warning:  #1357-D: static initialisation of variable "str" using address of string literal may cause link failure --ropi
```

because the global pointer "str" will need to be initialized to the address of the char string "test" in the .constdata section, but absolute addresses cannot be used in a PI system.

```
int bar;
int *foo = &bar;     /* global pointer */
```

when compiled with `--apcs /rwpi` gives:

```
Warning:  #1359-D: static initialisation of variable "foo" using address of bar may cause link failure --rwpi
```

because the global pointer "foo" will need to be initialized to the address of "bar" in the .data section, but absolute addresses cannot be used in a PI system.

The workaround is to change your code to avoid use of a global pointer, e.g. use a global array or local pointer instead.

See also FAQ "What does "Error: L6248E: cannot have address type relocation" mean?" at: http://www.arm.com/support/rvds3_faq.html

1360: static initialisation of extern const <entity> using address of <xxxx> cannot be lowered for RWPI

For example:

```
extern int y;
int* const x = &y;

int* foo()
{
    return(x);
}
```

When this is compiled with "--apcs /rwpi" it produces a warning. This is due to the compiler being unable to define a direct address offset between the variables x and y because y is prefixed by extern.

1361: Type of result operand is narrower than actual result

1362: use of <xxxx> is deprecated

1363: unrecognized format function type <type> ignored

1364: base class <class> uses tail padding of base class <class>

1365: the "init_priority" attribute can only be used for definitions of static data members and namespace scope variables of class types

1367: this anonymous union/struct field is hidden by <xxxx>

1368: invalid error number

1369: invalid error tag

1370: expected an error number or error tag

1371: size of class is affected by tail padding

Copyright © 2006-2008 ARM Limited. All rights reserved.

1372: labels can be referenced only in function definitions

1373: transfer of control into a statement expression is not allowed

1374: transfer of control out of a statement expression is not allowed

1375: this statement is not allowed inside of a statement expression

1376: a non-POD class definition is not allowed inside of a statement expression

1377: destructible entities are not allowed inside of a statement expression

1378: a dynamically-initialized local static variable is not allowed inside of a statement expression

1379: a variable-length array is not allowed inside of a statement expression

1380: a statement expression is not allowed inside of a default argument

1381: pragma secondname only applies to function definitions

1382: Type of result operand is narrower than actual result

1383: nonstandard conversion between pointer to function and pointer to data

1384: interface types cannot have virtual base classes

1385: interface types cannot specify "private" or "protected"

1386: interface types can only derive from other interface types

1387: <type> is an interface type

1388: interface types cannot have typedef members

1389: interface types cannot have user-declared constructors or destructors

1390: interface types cannot have user-declared member operators

1391: interface types cannot be declared in functions

1392: cannot declare interface templates

1393: interface types cannot have data members

1394: interface types cannot contain friend declarations

1395: interface types cannot have nested classes

1396: interface types cannot be nested class types

1397: interface types cannot have member templates

1398: interface types cannot have static member functions

1399: this pragma cannot be used in a_pragma operator (a #pragma directive must be used)

1400: qualifier must be base class of <type>

1401: declaration must correspond to a pure virtual member function in the indicated base class

Copyright © 2006-2008 ARM Limited. All rights reserved.

1402: integer overflow in internal computation due to size or complexity of <type>

1403: integer overflow in internal computation

1404: __w64 can only be specified on int, long, and pointer types

1405: potentially narrowing conversion when compiled in an environment where int, long, or pointer types are 64 bits wide

1406: current value of pragma pack is <value>

1407: arguments for pragma pack(show) are ignored

1408: invalid alignment specifier value

1409: expected an integer literal

1410: earlier __declspec(align(...)) ignored

1411: expected an argument value for the <attribute> attribute parameter

1412: invalid argument value for the <attribute> attribute parameter

1413: expected a boolean value for the <attribute> attribute parameter

1414: a positional argument cannot follow a named argument in an attribute

1415: attribute <attribute> has no parameter named <param>

1416: expected an argument list for the <attribute> attribute

1417: expected a "," or "]"

1418: attribute argument <arg> has already been given a value

1419: a value cannot be assigned to the <attribute> attribute

1420: a throw expression may not have pointer-to-incomplete type

1421: alignment-of operator applied to incomplete type

1422: <attribute> may only be used as a standalone attribute

1423: <attribute> attribute cannot be used here

1424: unrecognized attribute <attribute>

1425: attributes are not allowed here

1426: invalid argument value for the <attribute> attribute parameter

1427: too many attribute arguments

1428: conversion from inaccessible base class <type> is not allowed

1429: option "export" requires distinct template signatures

1430: narrow and wide string literals cannot be concatenated

1431: GNU layout bug not emulated because it places virtual base <no1> outside <no2> object boundaries

1432: virtual base <no1> placed outside <no2> object boundaries

Copyright © 2006-2008 ARM Limited. All rights reserved.

1433: nonstandard qualified name in namespace member declaration

1434: reduction in alignment ignored

1435: const qualifier ignored

1436: breakpoint argument must be an integral compile-time constant

1437: breakpoint argument must be within 0-65535 when compiling for ARM

1438: breakpoint argument must be within 0-255 when compiling for Thumb

1439: BKPT instruction is not supported on target architecture/processor

1440: oversize bitfield layout will change -- consider preceeding with "%s:0

1441: nonstandard cast on lvalue

1442: polymorphic base classes need to be exported if they are to be used for exported derivation

1443: polymorphic base classes inherited via virtual derivation need to be exported

1444: polymorphic base classes inherited via virtual derivation need all virtual functions to be exported

1446: non-POD class type passed through ellipsis
1447: a non-POD class type cannot be fetched by va_arg
The C++ ISO Specification defines that the non-required arguments of a variadic function must be of type POD (plain-old-data), such as an int or a char, but not structs or classes. To avoid the error/warning the address of a class or struct could be given instead.

1448: the 'u' or 'U' suffix must appear before the 'l' or 'L' suffix in a fixed-point literal

1450: integer operand may cause fixed-point overflow

1451: fixed-point constant is out of range

1452: fixed-point value cannot be represented exactly

1453: constant is too large for long long

1454: layout qualifier cannot qualify pointer to shared void

1456: a strong using-directive may only appear in a namespace scope

1457: <xxxx> declares a non-template function -- add <> to refer to a template instance

1458: operation may cause fixed-point overflow

1459: expression must have integral, enum, or fixed-point type

1460: expression must have integral or fixed-point type

1461: function declared with "noreturn" does return

1462: asm name ignored because it conflicts with a previous declaration

1463: class member typedef may not be redeclared

1464: taking the address of a temporary

Copyright © 2006-2008 ARM Limited. All rights reserved.

1465: attributes are ignored on a class declaration that is not also a definition

1466: fixed-point value implicitly converted to floating-point type

1467: fixed-point types have no classification

1468: a template parameter may not have fixed-point type

1469: hexadecimal floating-point constants are not allowed

1471: floating-point value does not fit in required fixed-point type

1472: value cannot be converted to fixed-point value exactly

1473: fixed-point conversion resulted in a change of sign

1474: integer value does not fit in required fixed-point type

1475: fixed-point operation result is out of range

1481: fixed-point value does not fit in required floating-point type

1482: fixed-point value does not fit in required integer type

1483: value does not fit in required fixed-point type

1485: a named-register storage class is not allowed here

1486: <xxxx> redeclared with incompatible named-register storage class

1487: named-register storage class cannot be specified for aliased variable

1488: named-register storage specifier is already in use

1491: invalid predefined macro entry at line <linenumber>: <reason>

1492: invalid macro mode name <macroname>

1493: incompatible redefinition of predefined macro <macroname>

1494: redeclaration of <xxxx> is missing a named-register storage class

1495: named register is too small for the type of the variable

1496: arrays cannot be declared with named-register storage class

1497: const_cast to enum type is nonstandard

1499: __swi parameter <param> is not within permitted range (0 to 0xff) for Thumb SWI instruction

1500: too many arguments for __swi or __swi_indirect function

1501: arguments for __swi or __swi_indirect function must have integral type

1502: __swi_indirect function must have arguments

1503: first argument for __swi_indirect function must have integral type

1504: result of __swi or __swi_indirect function must be returned in integer registers

1505: source file <filename> has bad format

Copyright © 2006-2008 ARM Limited. All rights reserved.

1506: error while writing <filename> file: <reason>

1507: cannot overload functions distinguished by function qualifier alone

1508: function qualifier <xxxx> not permitted on a virtual member function

1509: function "__attribute__((__<xxxx>__))" present on overridden virtual function <xxxx> must be present on overridding function

1510: function qualifier <xxxx> is not identical on overridden virtual function <xxxx>

1511: function qualifier <xxxx> present on overridden virtual function <xxxx> must be present on overridding function

The following old-style error and warning messages can still be given:

C2005E: an __irq function cannot call functions that use stack checking

Interrupt handlers can be written in C using the compiler keyword __irq, however, this should be used with care. This is because the IRQ handler will be executed in IRQ mode rather than e.g. User mode, so any stack accesses will use the IRQ stack rather than the User stack. Remember to initialize the stack pointer for IRQ mode (SP_IRQ)! Also, do not compile with '--apcs /swst', because the IRQ function will not be compiled with a stack check. It must not call a subroutine in IRQ mode which has been compiled with stack checking because of the risk that SL (Stack Limit) has not been set up for IRQ mode.

C2012U: Too many symbols in object file

In ADS 1.2, RVCT 1.2, RVCT 2.0 and later the compiler limit for the number of symbols in an object is 2^{24}.

C2061E: specified processor or architecture does not support ARM.

This error occurs when dealing with a device that does not support ARM instruction sets, e.g. the ARM Cortex M3, which only supports the Thumb2 instruction set.

C2068E: Uninitialised or corrupted use of PSR. This code may not work correctly

See C3001E.

Obsolete from 2.2 onwards; C2067I: option -zas will not be supported in future releases of the compiler

The "-zas" option is provided for backward compatibility only.
This warning is enabled by default, but may be suppressed with the "-Wy" switch.
Refer to the compiler documentation for more information about "-zas" and "-Wy".

C2070W: Memory access attributes below the minimum requirement for ARM/Thumb

An invalid –memaccess option has been specified

C2079E: unsupported CPU <cpu>

C2083W: -g defaults to -O2 if no optimization level is specified

If the user does not specify an optimisation level, the default –O2 will be used during file conversion. You may see this warning within CodeWarrior, because it is not possible to specify –O2 on the command line because it is the default. In this case the warning can be safely ignored or suppressed.

Copyright © 2006-2008 ARM Limited. All rights reserved.

`C2084W: support for --apcs /adsabi is deprecated`

See: http://www.arm.com/support/faqdev/1347.html

`C2085E: support for architecture 5ExP is deprecated`

`C2530-D: padding added to end of struct "anonymous"`

This diagnostic can be generated by adding “--remarks” to the end of the command line. It will display “anonymous” when no ‘tag’ is specified.

`C3000E: SWI number <number> too large`

`C3001E: <Reg> corrupted but possibly reused later. This code may not work correctly`

Example:

```
unsigned int foo(void)
{
    unsigned int linkReg;
    __asm{ mov linkReg, r14 }
    return linkReg;
}
```

The compiler is warning that the code may not behave as expected. In particular, r14 may not always contain the "return address" at that point, because the compiler may have inlined the function, or may have pushed LR onto the stack to be able to re-use r14 for temporary storage. The preferred solution for the above case is to use the new __return_address() intrinsic added in RVCT 2.0. Please refer to Chapter 3, RVCT 3.0 Compiler and Libraries documentation - 3.1.4.

`C3002W: illegal unaligned load or store access - use __packed instead`

`C3003E: FPU <s> is incompatible with selected CPU option`

`Example: armcc –cpu arm10200 –fpu fpa`

will fail because the arm10200 contains a vfp, not fpa.

`C3004E: apcs /interwork is only allowed when compiling for processors that support Thumb instructions`

Example:

```
armcc -c --apcs /interwork --cpu strongarm1 main.c
```

will fail because the StrongARM processor does not support Thumb

`C3005E: specified processor or architecture does not support Thumb instructions`

Example:

```
tcc -c --cpu strongarm1 main.c
```

will fail because the StrongARM processor does not support Thumb

`C3006E: specified processor or architecture does not support ARM instructions.`

This error occurs when dealing with a device that does not support ARM instruction sets, e.g. the ARM Cortex M3, which only supports the Thumb2 instruction set.

`C3007E: Uninitialised or corrupted use of PSR. This code may not work correctly`

See `C3001E`.

`C3008W: splitting LDM/STM has no benefit`

Inappropriate use of the switch "`--split_ldm`". This option has no significant benefit for cached systems, or for processors with a write buffer.

`C3009E: unsupported CPU <xxxx>`

`C3010W: -g defaults to -O2 if no optimization level is specified`

If the user does not specify an optimisation level, the default –O2 will be used during file conversion.

Copyright © 2006-2008 ARM Limited. All rights reserved.

C3011E: support for --apcs /adsabi is deprecated
See: http://www.arm.com/support/faqdev/1347.html

C3012E: support for architecture 5ExP is deprecated

C3013W: support for --apcs {<option>} is deprecated

C3014W: software stackchecking is no longer supported

C3015E: Unbalanced pragma pop, ignored
"#pragma push" and "#pragma pop" save and restore the current pragma state.
A pop must be paired with a push. An error is given for e.g.:

```
#pragma push
:
#pragma pop
:
#pragma pop
```

C3016W: unknown option '<option>': ignored

C3017W: <name> may be used before being set
The compiler's data flow analysis feature is now on by default in RVCT 2.1 and later. In RVCT 2.0.1 and earlier, it had to be enabled with the "-fa" switch. Be aware that data flow analysis is always disabled at -O0 (even if -fa is specified in RVCT 2.0.1 and earlier).

The compiler performs data flow analysis as part of its optimization process, and this information can be used to identify potential problems in the code (e.g variables being used before being set). However, this is really a by-product of optimization rather than a feature its own right, and the data flow analysis that detects 'used before being set' only analyses hardware register use, i.e. variables that are held in processor registers. It does not analyse variables/structures etc that are allocated on the stack, i.e. stored in memory rather than in processor registers. As code generated (and hence register/memory usage) by the compiler varies with the level of optimization, the warning could appear for code compiled at one level of optimization but not others, e.g. you might see it at -O2, but not -O1.

So beware that the current data flow analysis is not intended to be a fully complete feature. You should treat the C2874W warnings given by the compiler as a guide, but should not rely on these warnings to identify faulty code reliably. The compiler will never provide as much information as a special purpose tool such as Lint.

C3018W: division by zero <xxxx>
Constant propagation shows that a divide or remainder operator has a second operand with value `0`. It will be an error if execution reaches this expression.

C3039E: I/O error on object stream

C3041U: I/O error writing <xxxx>

Obsolete from 2.2 onwards; C3046U: out of store (for error buffer)

C3047U: Too many errors

C3048U: out of store while compiling with -g. Allocation size was <size1>, system size is <size2>
C3049U: out of store. Allocation size was <size1>, system size is <size2>

Copyright © 2006-2008 ARM Limited. All rights reserved.

A storage allocation request by the compiler failed. Compilation of the debugging tables requested with the `-g` option may require a great deal of memory. Recompiling without `-g`, or with the program split into smaller pieces, may help.

`C3050U: Compilation aborted.`

`C3051E: couldn't write file '<filename>'`
`C3052E: couldn't read file '<filename>'`
`C3055U: internal fault in inferFileName`

`C3056E: bad option <s>`
`C3057E: bad option <s1 s2>`

For example, the switches "--apcs /softfp", " --apcs /narrow", "--apcs /wide" which were supported in SDT, are no longer supported in ADS or RVCT and so must be removed from the compiler command-line.

`Obsolete from 2.2 onwards; C3059E: Missing file argument for '<option>'`

<option> requires a file parameter, e.g. `–-errors err.txt`

`Obsolete from 2.2 onwards; C3060E: No argument to compiler option`

`Obsolete from 2.2 onwards; C3061E, C3062E, C3063E: unknown option`

Examples:

```
Error: C3061E: unknown option '-fz': ignored
Error: C3063E: unknown option '-za1': ignored
```

These compiler options were commonly used in build scripts for SDT, however these are no longer supported by ADS or RVCT. You should remove these switches from any makefiles.

`C3064E: Overlong filename: <filename>`

`C3065E: type of input file <filename> unknown`

`C3066E: The code space needed for this object is too large for this version of the compiler`

Split the source file into smaller pieces.

`Obsolete from 2.2 onwards; C3067E: Couldn't write installation configuration`

`Obsolete from 2.2 onwards; C3074E: Can't open -via file filename`

`C3075E: Can't open <filename> for output`

`C3078E: stdin ('-') combined with other files`

`C3079E: <xxxx> command with no effect`

`C3396E: Source file-name 'filename' cannot be configured`

`C3403E: __alloca_state not defined`

`Obsolete from 2.2 onwards; C3410W: Option <option> has been obsoleted`

For example:

```
armcc --dwarf1
Error: C3410W: Option --dwarf1 has been obsoleted
```

Use --dwarf2 instead

`C3419W: dynamic stack alignment veneer inserted in <xxxx>`

Copyright © 2006-2008 ARM Limited. All rights reserved.

This warning is given when compiling `__irq` functions for `--cpu=Cortex-M3-rev0` to force the stack to be 8-byte aligned on entry into the interrupt.

`C3421W: write to string literal`

There is a write through a pointer, which has been assigned to point at a literal string. The behaviour is undefined by to the ANSI standard; a subsequent read from the location written may not reflect the write.

`C3435E: reference to <xxxx> not allowed`

`C3447E: option ‘-E’ and input file <filename> type conflict`

Example: `"armcc -E foo.s"`

`Obsolete from 2.2 onwards; C3455E: cannot form pointer to <function> function`

`C3463E: Invalid combination of memory access attributes`

`C3464E: Maximum pointer alignment must be a power of 2`

`Obsolete from 2.2 onwards; C3465E: The in-memory file system is obsolete, use the normal include mechanisms`

Please see Section 2.2.2 and Section 2.2.3 of the RVCT 3.0 Compiler and Libraries Guide for more information or http://www.arm.com/support/faqdev/1363.html: Libraries - Header file searching with -I and -J.

`C3466W: Feedback line ignored, unrecognised pattern`

`Obsolete from 2.2 onwards; C3473E: unknown CPU <cpu>`
`Obsolete from 2.2 onwards; C3481E: Bad --diag_style argument style`

`C3484E: Minimum toplevel array alignment must be 1, 2, 4 or 8`

`C3486W: option '<option>' causes input file '<filename>' to be ignored`

`C3487E: read from variable '<var>' with offset out of bounds`
`C3488E: write to variable '<var>' with offset out of bounds`

For example :

```
void foo(void) {
  unsigned int pntr;

  pntr = (unsigned int)&pntr;
  pntr -=4;
  pntr = *(unsigned int*)pntr;
}
```

produces the warning C3487E.

`C3489E: vfp_status() intrinsic not supported for targets without VFP`

`C3490W: instruction set switching using file extension is deprecated`

`C3491E: stack frame <xxxx> is larger than stack limit`

`C3493E: Function alignment must be a power of 2 and greater than 1`

Copyright © 2006-2008 ARM Limited. All rights reserved.

3. ARM Assembler (armasm) Errors and Warnings

`A1017E: :INDEX: cannot be used on a pc-relative expression`

The `:INDEX:` expression operator has been applied to a PC-relative expression, most likely a program label. `:INDEX:` returns the offset from the base register in a register-relative expression. If you wish to obtain the offset of a label called `<label>` within an area called `<areaname>`, use `<label> - <areaname>`. See RVCT 3.0 Assembler Guide, section 3.6.10, "Unary operators"

`A1020E: Bad predefine: <directive>`

The operand to the `--predefine (-pd)` command line option was not recognized. The directive must be enclosed in quotes if it contains spaces, for example on
Windows:

```
--predefine "versionnum SETA 5"
```

If the `SETS` directive is used, the argument to the directive must also be enclosed in quotes, which may need to be escaped depending upon operating system and shell. For example:

```
--predefine "versionstr SETS \"5A\""
```

`A1021U: No input file`

No input file was specified on the command line. This may be because there was no terminating quote on a quoted argument.

`A1023E: File "<filename>" could not be opened: <reason>`

`A1024E: File "<filename>" could not all be loaded: <reason>`

`A1042E: Unrecognized APCS qualifier '<qualifier>'`

There is an error in the argument given to the `--apcs` command line option.
Check the spelling of `<qualifier>`.

`A1051E: Cannot open --depend file '<filename>': <reason>`

`A1055E: Cannot open --errors file '<filename>': <reason>`

`A1056E: Target cpu <cpu> not recognized`

The name given in the `--cpu <cpu>` command line option was not a recognized processor name. Check the spelling of the argument.

`A1067E: Output file specified as '<file1>', but it has already been specified as '<file2>'`

More than one output file has been specified on the command line. Misspelling a command line option can cause this.

`A1071E: Cannot open listing file '<filename>'`

The file given in the `--list <filename>` command line option could not be opened. This could be because the given name is not valid, there is no space, a read-only file with the same name already exists, or the file is in use by another process. Check that the correct path for the file is specified.

`A1072E: The specified listing file '<filename>' must not be a .s or .o file`

The filename argument to the --list command line option has an extension that indicates it is a source or object file. This may be because the filename argument was accidentally omitted from the command line. Check that the correct argument is given to the `--list` command line option.

`A1073E: The specified output file '<filename>' must not be a source file`

Copyright © 2006-2008 ARM Limited. All rights reserved.

The object file specified on the command line has a filename extension that indicates it is a source file. This may be because the object filename was accidentally omitted from the command line.

`A1074E: The specified depend file '<filename>' must not be a source file`
`A1075E: The specified errors file '<filename>' must not be a source file`

The filename argument to the `--depend` / `--errors` command line option has an extension that indicates it is a source (.s) file. This may be because the filename argument was accidentally omitted from the command line. Check that the correct arguments are given.

`A1085E: Forced user-mode LDM/STM must not be followed by use of banked R8-R14`

The ARM architecture does not allow you to access the 'banked' registers on the instruction following a 'USER registers' LDM or STM. The ARM Architecture Reference Manual says this form of LDM must not be followed by an instruction, which accesses banked registers (a following NOP is a good way to ensure this)

Example:

```
stmib   sp, {r0-r14}^  ;  Return a pointer to the frame in a1.
mov     r0, sp
```

change to:

```
stmib   sp, {r0-r14}^  ;  Return a pointer to the frame in a1.
nop
mov     r0, sp
```

`A1088W: Faking declaration of area AREA |$$$$$$$|`

This is given when no `AREA` is given (see `A1105E`)

`A1099E: Structure stack overflow max stack size`

`A1100E: Structure stack underflow`

`A1105E: Area directive missing`

This is given when no `AREA` is given (see `A1088W`)

`A1106E: Missing comma`

`A1107E: Bad symbol type, expect label`

`A1108E: Multiply defined symbol <name>`

`A1109E: Bad expression type`

`A1110E: Expected constant expression`

A constant expression was expected after, e.g. `SETA`. See the RVCT 3.0 Assembler Guide, section 3.6.3, "Numeric expressions"

`A1111E: Expected constant or address expression`
`A1112E: Expected address expression`

`A1113E: Expected string expression`

A string expression was expected after, e.g. `SETS`. See the RVCT 3.0 Assembler Guide, section 3.6.1, "String expressions"

`A1114E: Expected register relative expression`

Examples:

The generic form: `LDR r4,[r9,offset]`

must be rewritten as: `LDR r4,[r9,#offset]`

Copyright © 2006-2008 ARM Limited. All rights reserved.

A1116E: String operands can only be specified for DCB

A1117E: Register symbol <name> already defined

A1118E: No current macro expansion

A1119E: MEND not allowed within conditionals

MEND means "END of Macro" (not the English word "mend"). See the RVCT 3.0 Assembler Guide, section 2.8, "Using macros".

A1120E: Bad global name

A1121E: Global name <name> already exists

A1122E: Locals not allowed outside macros

A1123E: Bad local name

A1125E: Unknown or wrong type of global/local symbol <name>

A1126E: Bad alignment boundary, must be a multiple of 2

A1127E: Bad IMPORT/EXTERN name

A1128E: Common name <sym> already exists
A1129E: Imported name <sym> already exists

A1130E: Bad exported name
A1131E: Bad symbol type for exported symbol <sym>

A1133E: Bad required symbol name
A1134E: Bad required symbol type, expect (symbol is either external or label) and (symbol is relocatable and absolute)

A1135E: Area name missing

AREA names starting with any non-alphabetic character must be enclosed in bars, e.g:
change:

```
AREA 1_DataArea, CODE, READONLY
```

to:

```
AREA |1_DataArea|, CODE, READONLY
```

A1136E: Entry address already set

A1137E: Unexpected characters at end of line

This is given when extra characters, which are not part of an (ARM) instruction, are found on an instruction line, for example:

```
ADD r0, r0, r1 comment
```

Could be changed to:

```
ADD r0, r0, r1 ; comment
```

A1138E: String <string> too short for operation, length must be > <oplength>
A1139E: String overflow, string exceeds <max> characters

A1140E: Bad operand type

A1141E: Relocated expressions may only be added or subtracted

A1142E: Subtractive relocations not supported for ELF format output

Copyright © 2006-2008 ARM Limited. All rights reserved.

This can occur when trying to access data in another area. For example, using:

```
LDR r0, [pc, #label - . - 8]
```

or its equivalent:

```
LDR r0, [pc, #label-{PC}-8]
```

where 'label' is defined in a different AREA.

These 'subtractive relocations' were allowed with SDT AOF, but not with ELF, so this error message can sometimes appear when migrating an SDT project to ADS or RVCT.

To resolve this change your code to use the simpler, equivalent syntax:

```
LDR r0, label
```

This works in both cases of 'label' being either in the same area or in a different area.

`A1145E: Undefined exported symbol <sym>`

`A1146E: Unable to open output file <codeFileName>: <reason>`

`A1147E: Bad shift name`
`A1148E: Unknown shift name <name>, expected one of LSL, LSR, ASR, ROR, RRX`

`A1149E: Shift option out of range`

Example:

```
mov   r0, r0, LSR #0x0
add   r0, r0, r1, LSR #0x0
```

Strictly, according to the ARM Architecture Reference Manual, `LSR #0` does not exist. You should use `LSL #0`, or even just omit the shift as:

```
mov   r0, r0
add   r0, r0, r1
```

`A1150E: Bad symbol, not defined or external`

This typically occurs in two cases:

1) when the current file requires another file to be `INCLUDE`d to define some symbols, for example:

```
"init.s", line 2: Error: A1150E: Bad symbol
   2 00000000   DCD  EBI_CSR_0
```

typically requires a definitions file to be included, e.g:

```
INCLUDE     targets/eb40.inc
```

2) when the current file requires some symbols to be `IMPORT`ed, for example:

```
"init.s", line 4: Error: A1150E: Bad symbol
    4 00000000      LDR r0, =||Image$$RAM$$ZI$$Limit||
```

typically requires the symbol to be imported, e.g:

```
IMPORT ||Image$$RAM$$ZI$$Limit||
```

`A1151E: Bad register name symbol`

Example:

```
MCR p14, 3, R0, Cr1, Cr2
```

The coprocessor registers `"CR"` must be labelled as a lowercase 'c' for the code to build. The ARM Register can be '`r`' or '`R`', hence:

```
MCR     p14, 3, r0, c1, c2
```

or

```
MCR     p14, 3, R0, c1, c2
```

`A1152E: Unexpected operator`
`A1153E: Undefined symbol`
`A1154E: Unexpected operand, operator expected`
`A1155E: Unexpected unary operator equal to or equivalent to <operator>`

Copyright © 2006-2008 ARM Limited. All rights reserved.

A1156E: Missing open bracket

A1157E: Syntax error following directive

A1158E: Illegal line start should be blank

Some directives, e.g. ENTRY, IMPORT, EXPORT, GET must be on a line without a label at the start of the line. This error will be given if a label is present.

A1159E: Label missing from line start

Some directives, e.g. FUNCTION or SETS, require a label at the start of the line, for example:

```
my_func FUNCTION
```

or

```
label SETS
```

This error will be given if the label is missing.

A1160E: Bad local label number

A local label is a number in the range 0-99, optionally followed by a name. See RVCT 3.0 Assembler Guide, section 3.5.6, "Local labels."

A1161E: Syntax error following local label definition

A1162E: Incorrect routine name <name>

A1163E: Unknown opcode <name> , expecting opcode or Macro

The most common reasons for this are:

1) Forgetting to put some white space on the left hand side margin, before the instruction, for example change:

```
MOV PC,LR
```

to

```
    MOV PC,LR
```

2) Use of a hardware floating point instruction without using the --fpu switch, for example:

```
FMXR FPEXC, r1 ; must be assembled with armasm --fpu vfp
```

or

```
LDFD f0, [r0]  ; must be assembled with armasm --fpu fpa
```

3) Mis-typing the opcode, e.g ADDD instead of ADD

A1164E: Opcode not supported on selected processor

The processor selected on the armasm command line does not support this instruction. Check the ARM Architecture Reference Manual. This may occur when attempting to use halfword instructions on an old architecture that does not support halfwords, e.g. "STRH r0,[r1]" assembled with "--cpu 3"

A1165E: Too many actual parameters, expecting <actual> parameters

A1166E: Syntax error following label

A1167E: Invalid line start

A1168E: Translate not allowed in pre-indexed form

A1169E: Missing close square bracket

A1170E: Immediate <n> out of range for this operation must be below <m>

This error is given if a MOV or MVN instruction is used with a constant that cannot be assembled. See RVCT 3.0 Assembler Guide, section 2.6.1, " Direct loading with MOV and MVN".

Copyright © 2006-2008 ARM Limited. All rights reserved.

A1171E: Missing close bracket

A1172E: Bad rotator <rotator>, must be even and between 0 and 30

A1173E: ADR/L cannot be used on external symbols

The ADR and ADRL pseudo-instructions may only be used with labels within the same code section. To load an out-of-area address into a register, use LDR instead.

A1174E: Data transfer offset 0x<val> out of range. Permitted values are 0x<min> to 0x<max>

A1175E: Bad register range

A1176E: Branch offset 0x<val> out of range. Permitted values are 0x<min> to 0x<max>

Branches are PC relative, and have a limited range. If you are using "local labels", you can use the ROUT directive to limit the scope of local labels, to help avoid referring to a wrong label by accident. See RVCT 3.0 Assembler Guide, section 3.5.6, "Local labels".

A1179E: Bad hexadecimal number

A1180E: Missing close quote

A1181E: Bad operator

A1182E: Bad based <base> number

A1183E: Numeric overflow

A1184E: Externals not valid in expressions

A1185E: Symbol missing

A1186E: Code generated in data area

A1187E: Error in macro parameters

A1188E: Register value <val> out of range. Permitted values are <min> to <max>

A1189E: Missing '#'

A1190E: Unexpected '<character>'

A1191E: Floating point register number out of range 0 to <maxi>

A1192E: Coprocessor register number out of range 0 to 15
A1193E: Coprocessor number out of range 0 to 15

A1194E: Bad floating-point number

A1195W: Small floating point value converted to 0.0

A1196E: Too late to ban floating point

A1198E: Unknown operand

This can occur when an operand is accidentally mistyped, for example:
`armasm init.s -g -PD "ROM_RAM_REMAP SETL {FALS}"`
should be:

Copyright © 2006-2008 ARM Limited. All rights reserved.

```
armasm init.s -g -PD "ROM_RAM_REMAP SETL {FALSE}"
```
See RVCT 3.0 Assembler Guide, section 3.5.4, "Assembly time substitution of variables"

`A1199E: Coprocessor operation out of range 0 to <max>`

`A1200E: Structure mismatch expect While/Wend`

`A1201E: Substituted line too long, maximum length <max>`
`A1202E: No pre-declaration of substituted symbol <name>`
See RVCT 3.0 Assembler Guide, section 3.5.4, "Assembly time substitution of variables"

`A1203E: Illegal label parameter start in macro prototype`

`A1204E: Bad macro parameter default value`

`A1205E: Register <reg> occurs multiply in list`

`A1206E: Registers should be listed in increasing register number order`
This warning is given if registers in e.g. LDM or STM instructions are not specified in increasing order *and* the `--checkreglist` option is used.

`A1207E: Bad or unknown attribute`
Example:
```
AREA test,CODE,READONLY,HALFWORD,INTERWORK
```
The `HALFWORD` and `INTERWORK` attributes are obsolete - simply remove them.

`A1209E: ADRL can't be used with PC as destination`

`A1210E: Non-zero data within uninitialized area <name>`
`A1211E: Missing open square bracket`

`A1212E: Division by zero`

`A1213E: Attribute <Attr1> cannot be used with attribute <Attr2>`

`A1214E: Too late to define symbol <sym> as register list`

`A1215E: Bad register list symbol`

`A1216E: Bad string escape sequence`

`A1217E: Error writing to code file <codeFileName>: <reason>`

`A1219E: Bad CPSR or SPSR designator`
For example:
```
MRS r0, PSR
```
It is necessary to specify which status register to use (`CPSR` or `SPSR`), e.g:
```
MRS r0, CPSR
```

`A1220E: BLX <xxxx> must be unconditional`

`A1221E: Area attribute '%s' not supported for %s object file format`

`A1223E: Comdat Symbol <symbol> is not defined`

`A1224E: %s format does not allow PC-relative data transfers between areas`

`A1225E: ASSOC attribute is not allowed in non-comdat areas`

Copyright © 2006-2008 ARM Limited. All rights reserved.

A1226E: SELECTION attribute is not allowed in non-comdat areas

A1227E: Comdat Associated area <area> undefined at this point in the file

A1228E: Comdat Associated area <area> is not an area name

A1229E: Missing COMDAT symbol

A1237E: Invalid register or register combination for this operation

A1238E: Immediate value must be word aligned when used in this operation

A1240E: Immediate value cannot be used with this operation

A1241E: Must have immediate value with this operation

A1242E: Offset must be word aligned when used with this operation
A1243E: Offset must be halfword aligned with this operation

A1244E: Missing '!'

A1245E: B or BL from Thumb code to ARM code

A1246E: B or BL from ARM code to Thumb code

A1247E: BLX from ARM code to ARM code, use BL
A1248E: BLX from Thumb code to Thumb code, use BL

This occurs when there is a `BLX <label>` branch from ARM code to ARM code (or from Thumb code to Thumb code) within this assembler file. This is not allowed because `BLX <label>` always results in a state change. The usual solution is to use BL instead.

A1249E: Post indexed addressing mode not available
A1250E: Pre indexed addressing mode not available for this instruction, use [Rn, Rm]

A1253E: Thumb branch to external symbol cannot be relocated: not representable in ARM ELF.

Branch "`B foo`" (foo is an extern) is not allowed in Thumb assembler code, because there is no corresponding ELF relocation.

A1254E: Halfword literal values not supported

Example:

```
LDRH  R3, =constant
```

Change the `LDRH` into `LDR`, which is the standard way of loading constants into registers.

A1256E: DATA directive can only be used in CODE areas

A1259E: Invalid PSR field specifier, syntax is <PSR>_ where <PSR> is either CPSR or SPSR

A1260E: PSR field '%c' specified more than once

A1261E: MRS cannot select fields, use CPSR or SPRS directly

This is caused by an attempt to use fields for CPSR or SPSR with an MRS instn, e.g:

```
MRS r0, CPSR_c
```

A1262U: Expression storage allocator failed

A1265U: Structure mismatch: IF or WHILE unmatched at end of INCLUDE file

Copyright © 2006-2008 ARM Limited. All rights reserved.

A1267E: Bad GET or INCLUDE for file <filename>

A1268E: Unmatched conditional or macro

A1270E: File "<filename>" not found

A1271E: Line too long, maximum line length is <MaxLineLength>

A1272E: End of input file

A1273E: '\\' should not be used to split strings

A1274W: '\\' at end of comment

A1283E: Literal pool too distant, use LTORG to assemble it within 1KB

For Thumb code, the literal pool must be within 1KB of the LDR instruction to access it. See A1284E and A1471W.

A1284E: Literal pool too distant, use LTORG to assemble it within 4KB

For ARM code, the literal pool must be within 4KB of the LDR instruction to access it. To solve this, add an LTORG directive into your assembler source file at a convenient place. Refer to the RVCT 3.0 Assembler Guide, section 2.6.2, "Loading with LDR Rd, =const" and section 7.3.1, "LTORG". See A1471W.

A1285E: Bad macro name
A1286E: Macro already exists
A1287E: Illegal parameter start in macro prototype
A1288E: Illegal parameter in macro prototype
A1289E: Invalid parameter separator in macro prototype
A1290E: Macro definition too big, maximum length <max>
A1291E: Macro definitions cannot be nested

The macro definition is invalid.

A1310W: Symbol attribute not recognized

A1311U: macro definition attempted within expansion

A1312E: Assertion failed

A1313W: Missing END directive at end of file

The assembler requires an END directive to know when the code in the file terminates - you can add comments or other such information in 'free' format after this directive.

A1314W: Reserved instruction (using NV condition)
A1315E: NV condition not supported on targeted CPU

A1316E: Shifted register operand to MSR has undefined effect

A1319E: Undefined effect (using PC as Rs)
A1320E: Undefined effect (using PC as Rn or Rm in register specified shift)
A1321E: Undefined effect (using PC as offset register)
A1322E: Unaligned transfer of PC, destination address must be 4 byte aligned
A1323E: Reserved instruction (Rm = Rn with post-indexing)
A1324E: Undefined effect (PC + writeback)

A1327W: Non portable instruction (LDM with writeback and base in reg. list, final value of base unpredictable)

LDM Operand restrictions:

Copyright © 2006-2008 ARM Limited. All rights reserved.

If the base register <Rn> is specified in <registers>, and base register writeback is specified, the final value of <Rn> is UNPREDICTABLE.

A1328W: Non portable instruction (STM with writeback and base not first in reg. list, stored value of base unpredictable)

STM Operand restrictions:

If <Rn> is specified as <registers> and base register writeback is specified:

* If <Rn> is the lowest-numbered register specified in <register_list>, the original value of <Rn> is stored.
* Otherwise, the stored value of <Rn> is UNPREDICTABLE.

A1329W: Unsafe instruction (forced user mode transfer with write-back to base)

A1331W: Unsafe instruction (PC as source or destination)
A1332W: Undefined effect (PC-relative SWP)
A1334E: Undefined effect (use of PC/PSR)
A1335W: Useless instruction (PC can't be written back)
A1337W: Useless instruction (PC is destination)
A1338W: Dubious instruction (PC used as an operand)

A1339W: Undefined if any of RdLo, RdHi, Rm are the same register

A1341E: Branch to unaligned destination, expect destination to be <max> byte aligned

A1342W: ADR/ADRL of symbol in another AREA will cause link-time failure if symbol is not close enough to this instruction

This warning is now superseded by:

Error: A1486E: ADR/ADRL of a symbol in another AREA is not supported in ELF.

A1355U: A Label was found which was in no AREA

Example:

This can occur where no white-space precedes an assembler directive.

Assembler directives must be indented with white-space, for example:

use:

```
IF :DEF: FOO
; code
ENDIF
```

not:

```
IF :DEF: FOO
; code
ENDIF
```

Symbols in the left hand column 1 are assumed to be labels, hence the error message.

A1356W: Instruction not supported on targeted CPU

This will occur if you try to use an instruction that is not supported by armasm's default architecture/processor, for example:

```
SMULBB  r0,r0,r1 ; may be assembled with armasm --cpu 5TE
```

The processor selected on the armasm command line does not support this instruction. Check the ARM Architecture Reference Manual.

A1406E: Bad decimal number
A1407E: Overlarge floating point value
A1408E: Overlarge (single precision) floating point value
A1409W: Small (single precision) floating value converted to 0.0
A1410E: This floating-point value cannot be specified as an immediate operand, permitted constants are 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 0.5 and 10.0
A1411E: Closing '>' missing from vector specifier

Copyright © 2006-2008 ARM Limited. All rights reserved.

A1412E: Bad vector length, should be between <min> and <max>
A1413E: Bad vector stride, should be between <min> and <max>
A1414E: Vector wraps round over itself, length * stride should not be greater than <max>
A1415E: VFPASSERT must be followed by 'VECTOR' or 'SCALAR'
A1416E: Vector length does not match current vector length <len>
A1417E: Vector stride does not match current vector stride
A1418E: Register has incorrect type '<type>' for instruction, expect floating point/double register type
A1419E: Scalar operand not in first bank
A1420E: Lengths of vector operands are different
A1421E: Strides of vector operands are different
A1422E: This combination of vector and scalar operands is not allowed
A1423E: This operation is not vectorizable
A1424E: Vector specifiers not allowed in operands to this instruction
A1425E: Destination vector must not be in a scalar bank
A1426E: Source vector must not be in a scalar bank
A1427E: Operands have a partial overlap
A1428E: Register list contains registers of varying types
A1429E: Expected register list

The VFP instructions are malformed. See RVCT3.0 Assembler Guide, section 5, "Vector Floating-point Programming"

A1430E: Unknown frame directive
A1431E: Frame directives are not accepted outside of PROCs/FUNCTIONs

Invalid FRAME directive. See RVCT 3.0 Assembler Guide, 6.5, "Frame description directives"

A1432E: Floating-point register type not consistent with selected floating-point architecture

A1433E: Only the writeback form of this instruction exists

The addressing mode specified for the instruction did not include the writeback specifier (a '!' after the base register), but the instruction set only supports the writeback form of the instruction. Either use the writeback form, or replace with instructions that have the desired behaviour.

A1434E: Architecture attributes '<attr1>' and '<attr2>' conflict

A1435E: {PCSTOREOFFSET} is not defined when assembling for an architecture

{PCSTOREOFFSET} is only defined when assembling for a processor, not for an architecture.

A1437E: {ARCHITECTURE} is undefined

A1446E: Bad or unknown attribute 'INTERWORK'. Use --apcs /interwork instead

Example:

```
AREA test1, CODE, READONLY
AREA test, CODE, READONLY, INTERWORK
```

This code may have originally been intended to work with SDT. The INTERWORK area attribute is now obsolete. To eliminate the warning:

a) remove the ", INTERWORK" from the AREA line.

b) assemble with 'armasm --apcs /interwork foo.s' instead

A1447W: Missing END directive at end of file, but a label named END exists. Perhaps you intended to put this in a column other than 1.

A1448W: Deprecated form of PSR field specifier used (use _f)
A1449W: Deprecated form of PSR field specifier used (use _c)
A1450W: Deprecated form of PSR field specifier used (use _cxsf for future compatibility)

Copyright © 2006-2008 ARM Limited. All rights reserved.

The ARM assembler (armasm) supports the full range of MRS and MSR instructions, in the form:

```
MRS(cond) Rd, CPSR
MRS(cond) Rd, SPSR
MSR(cond) CPSR_fields, Rm
MSR(cond) SPSR_fields, Rm
MSR(cond) CPSR_fields, #immediate
MSR(cond) SPSR_fields, #immediate
```

where 'fields' can be any combination of "cxsf".

Note that `MSR CPSR_c, #immediate` is a legitimate instruction (despite what is written in early versions of the ARM ARM), so a sequence of two instructions like:

```
MOV r0, #0x1F
MSR CPSR_c, r0
```

as commonly found in boot code, can be combined into one instruction, like:

```
MSR CPSR_c, #0x1F ; go to System mode, IRQ & FIQ enabled
```

Earlier releases of the assembler allowed other forms of the MSR instruction to modify the control field and flags field:

```
cpsr or cpsr_all    Control and flags field.
cpsr_flg            Flags field only.
cpsr_ctl            Control field only
```

and similarly for SPSR.
These forms are now deprecated, so should not be used. If your legacy code contains them, the assembler will report "Deprecated form of PSR field specifier used (use _cxsf)"
To avoid the warning, in most cases you should simply modify your code to use '_c', '_f', '_cf' or '_cxsf' instead.

For more information, see RVCT 3.0 Assembler Guide, Section 2.2.7 “Instruction capabilities”, and also FAQ “armasm: use of MRD and MSR instructions (‘Deprecated form of PSR field specifier’) at http://www.arm.com/support/faqdev/1472.html

`A1454E: FRAME STATE RESTORE directive without a corresponding FRAME STATE REMEMBER`

Invalid FRAME directive. See RVCT 3.0 Assembler Guide, 6.5, "Frame description directives"

`A1456W: INTERWORK area directive is obsolete. Continuing as if --apcs /inter selected.`

Example:

```
AREA test, CODE, READONLY, INTERWORK
```

This code may have originally been intended to work with SDT. The `INTERWORK` area attribute is now obsolete. To eliminate the warning:
a) remove the `", INTERWORK"` from the AREA line.
b) assemble with `'armasm --apcs /interwork foo.s'` instead

`A1457E: Cannot mix INTERWORK and NOINTERWORK code areas in same file.`

`INTERWORK` and (default) `NOINTERWORK` code areas cannot be mixed in same file. This code may have originally been intended to work with SDT. The `INTERWORK` area attribute is obsolete in RVCT.
Example:

```
AREA test1, CODE, READONLY
:
AREA test2, CODE, READONLY, INTERWORK
```

To eliminate the error:
a) move the two AREAs into separate assembler files, e.g. test1.s and test2.s
b) remove the `", INTERWORK"` from the AREA line in test2.s

Copyright © 2006-2008 ARM Limited. All rights reserved.

c) assemble test1.s with 'armasm --apcs /nointerwork'
d) assemble test2.s with 'armasm --apcs /interwork'
e) at link time, the linker will add any necessary interworking veneers

A1458E: DCFD or DCFDU not allowed when fpu is None.

A1459E: cannot B or BL to a register.
This form of the instruction is not allowed – consult the ARM ARM for the allowed forms.

A1461E: specified processor or architecture does not support Thumb instructions
Example:
It is likely that you are specifying a specific architecture or cpu using the --cpu option and then incorporating some Thumb code in the AREA that is generating this error.
For example: armasm --cpu 4 code.s
StrongARM is an architecture 4 (not 4T) processor and does not support Thumb code.

A1462E: specified memory attributes do not support this instruction

A1463E: SPACE directive too big to fit in area, area size limit 2^32

A1464W: ENDP/ENDFUNC without corresponding PROC/FUNC

A1466W: Operator precedence means that expression would evaluate differently in C
armasm has always evaluated certain expressions in a different order to C. This warning may help C programmers from being caught out when writing in assembler.
To avoid the warning, modify the code to make the evaluation order explicit (i.e. add more brackets), or suppress the warning with '--unsafe' switch.
See RVCT 3.0 Assembler Guide, section 3.6.9, "Operator precedence".

A1467W: FRAME ADDRESS with negative offset <offset> is not recommended
A1468W: FRAME SAVE saving registers above the canonical frame address is not recommended
A1469E: FRAME STATE REMEMBER directive without a corresponding FRAME STATE RESTORE
Invalid FRAME directive. See RVCT 3.0 Assembler Guide, 7.5, "Frame description directives"

A1471W: directive LTORG may be in an executable position
This can occur with e.g. the LTORG directive (see A1283E & A1284E). LTORG instructs the assembler to dump literal pool DCD data at this position. The data must be placed where the processor cannot execute them as instructions, otherwise this warning is given. A good place for an LTORG is immediately after an unconditional branch, or after the return instruction at the end of a subroutine. As a last resort, you could add a branch 'over' the LTORG, to avoid the data being executed, for example:

```
B unique_label
    LTORG
unique_label
```

A1475W: At least one register must be transferred, otherwise result is UNPREDICTABLE.

A1476W: BX r15 at non word-aligned address is UNPREDICTABLE

A1477W: This register combination results in UNPREDICTABLE behaviour

A1479W: Requested alignment <alignreq> is greater than area alignment <align>, which has been increased

Copyright © 2006-2008 ARM Limited. All rights reserved.

This is warning about an ALIGN directive which has a coarser alignment boundary than its containing AREA, which is not allowed. To compensate, the assembler automatically increases the alignment of the containing AREA for you. A simple test case that gives the warning is:

```
AREA test, CODE, ALIGN=3
ALIGN 16
mov pc, lr
END
```

In this example, the alignment of the AREA (ALIGN=3) is 2^3=8 byte boundary, but the mov pc,lr instruction will be on a 16 byte boundary, hence the error. (Note the difference in how the two alignment types are specified). These two types of alignment control are described in detail in the RVCT 3.0 Assembler Guide, section 6.8.1, "ALIGN" and 6.8.2, "AREA".

A1480W: Macro cannot have same name as a directive or instruction

A1481E: Object file format does not support this area alignment

A1482E: Shift option out of range, allowable values are from <min> to <max>

A1484E: Obsolete shift name 'ASL', use LSL instead

The ARM architecture does not have an ASL shift operation. The ARM barrel shifter only has the following 4 shift types: ROR, ASR, LSR, and LSL. An arithmetic (i.e. signed) shift left is the same as a logical shift left, because the sign bit always gets shifted out. Earlier versions of the assembler would silently convert ASL to LSL. This error can be downgraded to a warning by using the "`-unsafe`" switch.

A1485E: LDM/STM instruction exceeds maximum register count <max> allowed with --split_ldm

A1486E: ADR/ADRL of a symbol in another AREA is not supported in ELF.

The ADR and ADRL pseudo-instructions may only be used with labels within the same code section. To load an out-of-area address into a register, use LDR instead.

A1487E: Obsolete instruction name 'ASL', use LSL instead

The Thumb instruction ASL is now faulted. See the corresponding ARM ASL message A1484E.

A1488W: PROC/FUNC at line <lineno> without matching ENDP/ENDFUNC

A1489E: {FPU} is undefined

A1490E: {CPU} is undefined

A1491W: Internal error: Found relocation at offset <offset> with incorrect alignment

This may indicate an assembler fault – please contact your supplier.

A1492E: Immediate 0x<val> out of range for this operation. Permitted values are 0x<min> to 0x<max>

A1493E: REQUIRE must be in an AREA

A1495E: Target of branch is a data address

A1496E: Absolute relocation of ROPI address with respect to symbol <X> at offset 0x<X> may cause link failure

For example, when assembling with `--apcs /ropi`:

```
    AREA code, CODE
codeaddr DCD codeaddr
    END
```

because this generates an absolute relocation (R_ARM_ABS32) to a PI code symbol.

Copyright © 2006-2008 ARM Limited. All rights reserved.

A1497E: Absolute relocation of RWPI address with respect to symbol <X> at offset 0x<X> may cause link failure

For example, when assembling with `--apcs /rwpi`:

```
    AREA data, DATA
dataaddr DCD dataaddr
    END
```

because this generates an absolute relocation (R_ARM_ABS32) to a PI data symbol.

A1498E: Unexpected characters following Thumb instruction

For example:

```
ADD r0, r0, r1
```

is accepted as a valid instruction, for both ARM and Thumb, but:

```
ADD r0, r0, r1, ASR #1
```

is a valid instruction for ARM, but not for Thumb, so the "unexpected characters" are `", ASR #1"`.

A1499E: Register pair is not a valid contiguous pair

A1500E: Unexpected characters when expecting '<eword>'

A1501E: Shift option out of range, allowable values are 0, 8, 16 or 24

A1502W: Register <reg> is a caller-save register, not valid for this operation

A1505E: Bad expression type, expect logical expression

A1506E: Accumulator should be in form accx where x ranges from 0 to <max>

A1507E: Second parameter of register list must be greater than or equal to the first

A1508E: Structure mismatch expect Conditional

A1509E: Bad symbol type, expect label, or weak external symbol

A1510E: Immediate 0x<imm> cannot be represented by 0-255 and a rotation

A1511E: Immediate cannot be represented by combination of two data processing instructions

A1512E: Immediate 0x<val> out of range for this operation. Permitted values are <min> to <max>

A1513E: Symbol not found or incompatible Symbol type for <name>

A1514E: Bad global name <name>

A1515E: Bad local name <name>

A1516E: Bad symbol <name>, not defined or external

A1517E: Unexpected operator equal to or equivalent to <operator>

A1539E: Link Order dependency <name> not an area

A1540E: Cannot have a link order dependency on self

A1541E: <code> is not a valid condition code

A1542E: Macro names <name1> and <name2> conflict

A1543W: Empty macro parameter default value

A1544W: Invalid empty PSR field specifier, field must contain at least one of c,x,s,f

A1545E: Too many sections for one ELF file

A1546W: Stack pointer update potentially breaks 8 byte stack alignment

Copyright © 2006-2008 ARM Limited. All rights reserved.

Example: `PUSH {r0}`

The stack needs to be 8 byte aligned so pushing an odd number of registers will cause this warning to be given. This warning is suppressed by default. To enable this warning use "`--diag_warning 1546`". For more information please refer to Chapter 7, RVCT 3.0 Assembler guide: 7.8.15.

`A1547W: PRESERVE8 directive has automatically been set`

Example: `PUSH {r0,r1}`

This warning has been given because the `PRESERVE8` directive has not been explicitly set by the user, but the assembler has set this itself automatically. This warning is suppressed by default. To enable this warning use "`--diag_warning 1547`". For more information please refer to Chapter 7, RVCT 3.0 Assembler guide: 7.8.15.

`A1548W: Code contains LDRD/STRD indexed/offset from SP but REQUIRE8 is not set`

Example:
```
PRESERVE8
STRD r0,[sp,#8]
```

This warning is given when the `REQUIRE8` directive is not set when needed.

`A1549W: Setting of REQUIRE8 but not PRESERVE8 is unusual.`

Example:
```
PRESERVE8 {FALSE}
REQUIRE8
STRD r0,[sp,#8]
```

`A1550E: Input and output filenames are the same.`

`A1551E: Cannot add Comdef area <string> to non-comdat group`

`A1560E: Non-constant byte literal values not supported`

`A1561E: MERGE and STRING sections must be data sections`

`A1562E: Entry size for Merge section must be greater than 0`

`A1563W: Instruction stalls CPU for <stalls> cycle(s)`

The assembler can give information about possible interlocks in your code caused by the pipeline of the processor chosen by the --cpu option. This can be enabled with:
`armasm --diag_warning 1563`
Note: Where the --cpu option specifies a multi-issue processor such as Cortex-A8, the interlock warnings are unreliable.

`A1572E: Operator SB_OFFSET_11_0 only allowed on LDR/STR instructions`

`A1573E: Operator SB_OFFSET_19_12 only allowed on Data Processing instructions`

`A1574E: Expected one or more flag characters from <str>`

`A1575E: BLX with bit[0] equal to 1 is architecturally UNDEFINED`

`A1576E: Bad coprocessor register name symbol`

`A1577E: Bad coprocessor name symbol`

`A1578E: Bad floating point register name symbol ‘<sym>’`

`A1581W: Added <no_padbytes> bytes of padding at address <address>`

The assembler will warn by default when padding bytes are added to the generated code. This will occur whenever an instruction/directive is used at an address that requires a higher alignment, for example, to ensure ARM instructions start on a 4-byte boundary after some Thumb instructions, or where there is a DCB followed by DCD.

For example:

Copyright © 2006-2008 ARM Limited. All rights reserved.

```
        AREA Test, CODE, READONLY
        THUMB
    ThumbCode
        MOVS r0, #1
        ADR  r1, ARMProg
        BX   r1

    ;    ALIGN    ; <<< add to avoid the first warning
        ARM
    ARMProg
        ADD r0,r0,#1
        BX LR
        DCB 0xFF
        DCD 0x1234
        END
```

Results in the warnings:

```
A1581W: Added 2 bytes of padding at address 0x6
    8 00000008     ARM
A1581W: Added 3 bytes of padding at address 0x11
   13 00000014     DCD 0x1234
```

The warning may also occur when using ADR in Thumb-only code. The ADR Thumb pseudo-instruction can only load addresses that are word aligned, but a label within Thumb code might not be word aligned. Use ALIGN to ensure four-byte alignment of an address within Thumb code.

A1582E: Link Order area <name> undefined

A1583E: Group symbol <name> undefined

A1584W: Mode <mode> not allowed for this instruction

A1585E: Bad operand type (<typ1>) for operator <op>",

A1586E: Bad operand types (<typ1>, <typ2>) for operator <op>

A1587E: Too many registers <count> in register list, maximum of <max>

A1588E: Align only available on VLD and VST instructions

A1589E: Element index must remain constant across all registers

A1590E: Mix of subscript and non-subscript elements not allowed

A1593E: Bad Alignment, must match transfer size UIMM * <dt>

A1595E: Bad Alignment, must match <st> * <dt>, or 64 when <st> is 4

A1596E: Invalid alignment <align> for dt st combination

A1597E: Register increment of 2 not allowed when dt is 8

A1598E: Bad Register list length

A1599E: Out of range subscript, must be between 0 and <max_index>

A1600E: Section type must be within range SHT_LOOS and SHT_HIUSER

A1601E: Immediate cannot be represented

A1603W: This instruction inside IT block has UNPREDICTABLE results

A1604W: Thumb Branch to destination without alignment to <max> bytes

Copyright © 2006-2008 ARM Limited. All rights reserved.

A1606E: Symbol attribute <attr1> cannot be used with attribute <attr2>

A1607E: Thumb-2 wide branch instruction used, but offset could fit in Thumb-1 narrow branch instruction

A1608W: MOV pc,<rn> instruction used, but BX <rn> is preferred

A1609W: MOV <rd>,pc instruction does not set bit zero, so does not create a return address

A1611E: Register list increment of 2 not allowed for this instruction

A1612E: <type> addressing not allowed for <instr>

A1613E: Invalid register or register combination for this operation, <rcvd>, expected one of <expect>

A1614E: Scalar access not allowed when dt is 64

A1615E: Store of a single element or structure to all lanes is UNDEFINED

A1616E: Instruction, offset, immediate or register combination is not supported by the current instruction set

A1617E: Specified width is not supported by the current instruction set

A1618E: Specified instruction is not supported by the current instruction set

A1619E: Specified condition is not consistent with previous IT

A1620E: Error writing to file '<filename>': <reason>

A1621E: CBZ or CBNZ from Thumb code to ARM code

A1622E: Negative register offsets are not supported by the current instruction set

A1623E: Offset not supported by the current instruction set

A1624E: Branch from Thumb code to ARM code

A1625E: Branch from ARM code to Thumb code

A1626E: BL from Thumb code to ARM code

A1627E: BL from ARM code to Thumb code

This occurs when there is a branch from ARM code (`CODE32`) to Thumb code (`CODE16`) (or vice-versa) within this file. The usual solution is to move the Thumb code into a separate assembler file. Then, at link-time, the linker will add any necessary interworking veneers.

A1630E: Specified processor or architecture does not support ARM instructions

Certain processors such as Cortex-M3 implement only the Thumb instruction set, not the ARM instruction set. It is likely that the assembly file contains some ARM-specific instructions and is being built for one of these processors.

A1631E: Only left shifts of 1, 2 and 3 are allowed on load/stores

A1632E: Else forbidden in IT AL blocks

A1633E: LDR rx,= pseudo instruction only allowed in load word form

A1634E: LDRD/STRD has no register offset addressing mode in Thumb

A1635E: CBZ/CBNZ can not be made conditional

Copyright © 2006-2008 ARM Limited. All rights reserved.

A1636E: Flag setting MLA is not supported in Thumb

A1637E: Error reading line: <reason>

A1638E: Writeback not allowed on register offset loads or stores in Thumb

A1639E: Conditional DCI only allowed in Thumb mode

A1640E: Offset must be a multiple of four

A1641E: Forced user-mode LDM/STM not supported in Thumb

A1642W: Relocated narrow branch is not recommended

A1643E: Cannot determine whether instruction is working on single or double precision values.

A1644E: Cannot use single precision registers with FLDMX/LSTMX

A1645W: Substituted <old> with <new>

armasm can warn when it substitutes an instruction when assembling.

For example, ADD of a negative number can be transformed into SUB of a positive number; MOV negative => MVN positive, CMP negative => CMN positive.

For Thumb-2, unpredictable single register LDMs are transformed into LDRs.

This warning is suppressed by default, but can be enabled with `--diag_warning 1645`

For example:

```
AREA foo, CODE

ADD r0, #-1
MOV r0, #-1
CMP r0, #-1
```

When assembled with...:

```
armasm --diag_warning 1645
```

...the assembler reports...:

```
Warning: A1645W: Substituted ADD with SUB
    3 00000000  ADD r0, #-1
Warning: A1645W: Substituted MOV with MVN
    4 00000004  MOV r0, #-1
Warning: A1645W: Substituted CMP with CMN
    5 00000008  CMP r0, #-1
```

...and the resulting code generated is...:

```
    foo
        0x00000000:    e2400001    ..@.    SUB      r0,r0,#1
        0x00000004:    e3e00000    ....    MVN      r0,#0
        0x00000008:    e3700001    ..p.    CMN      r0,#1
```

A1646W: VMOV pseudo-instruction for a register to register move is deprecated. Please use a VORR instruction instead.

A1647E: Bad register name symbol. Expected Integer register.

A1648E: Bad register name symbol. Expected Wireless MMX SIMD register.

A1649E: Bad register name symbol. Expected Wireless MMX Status/Control or General Purpose register.

A1650E: Bad register name symbol. Expected any Wireless MMX register.

A1651E: TANDC, TEXTRC and TORC instructions with destination register other than R15 is undefined

The 6 messages above relate to Wireless MMX.

Copyright © 2006-2008 ARM Limited. All rights reserved.

A1652W: FLDMX/FSTMX instructions are deprecated in ARMv6. Please use FLDMD/FSTMD instructions to save and restore unknown precision values.

A1653E: Shift instruction using a status or control register is undefined

A1654E: Cannot access external symbols when loading/storing bytes or halfwords

A1655W: Instruction is UNPREDICTABLE if halfword/word/doubleword is unaligned

A1656E: Target must be at least word-aligned when used with this instruction

A1657E: Cannot load a byte/halfword literal using WLDRB/WLDRH =constant

A1658W: Support for <opt> is deprecated

The option passed to armasm is now deprecated. Use "armasm --help" to view the currently available options, or refer to the assembler documentation.

A1659E: Cannot B/BL/BLX between ARM/Thumb and Thumb-2EE

A1660E: Cannot specify scalar index on this register type

A1661E: Cannot specify alignment on this register

A1662E: Cannot specify a data type on this register type

A1663E: A data type has already been specified on this register

A1664E: Data type specifier not recognised

A1665E: Data type size must be one of 8, 16, 32 or 64

A1666E: Data type size for floating-point must be 32 or 64

A1667E: Data type size for polynomial must be 8 or 16

A1668E: Too many data types specified on instruction

A1669E: Data type specifier not allowed on this instruction

A1670E: Expected 64-bit doubleword register expression

A1671E: Expected 128-bit quadword register expression

A1672E: Expected either 64-bit or 128-bit register expression

A1673E: Both source data types must be same type and size.

A1674E: Source operand 1 should have integer type and be double the size of source operand 2

A1675E: Data types and sizes for destination must be same as source

A1676E: Destination type must be integer and be double the size of source

A1677E: Destination type must be same as source, but half the size

A1678E: Destination must be untyped and same size as source

A1679E: Destination type must be same as source, but double the size

A1680E: Destination must be unsigned and half the size of signed source

Copyright © 2006-2008 ARM Limited. All rights reserved.

A1681E: Destination must be unsigned and have same size as signed source

A1682E: Destination must be un/signed and source floating, or destination floating and source un/signed, and size of both must be 32-bits

A1683E: Data type specifiers do not match a valid encoding of this instruction

A1684E: Source operand type should be signed or unsigned with size between <min> and <max>

A1685E: Source operand type should be signed, unsigned or floating point with size between <min> and <max>

A1686E: Source operand type should be signed or floating point with size between <min> and <max>

A1687E: Source operand type should be integer or floating point with size between <min> and <max>

A1688E: Source operand type should be untyped with size between <min> and <max>

A1689E: Source operand type should be <n>-bit floating point

A1690E: Source operand type should be signed with size between <min> and <max>

A1691E: Source operand type should be integer, floating point or polynomial with size between <min> and <max>

A1692E: Source operand type should be signed, unsigned or polynomial with size between <min> and <max>

A1693E: Source operand type should be unsigned or floating point with size between <min> and <max>

A1694E: Instruction cannot be conditional in the current instruction set

Conditional instructions are not allowed in the specified instruction set, e.g. the instruction moveq is only allowed in ARM and Thumb-2 assembler, but not Thumb-1.

A1695E: Scalar index not allowed on this instruction

A1696E: Expected either 32-bit, 64-bit or 128-bit register expression

A1697E: Expected either 32-bit or 64-bit VFP register expression

A1698E: Expected 32-bit VFP register expression

A1699E: 64-bit data type cannot be used with these registers

A1700E: Source operand type should be integer with size between <min> and <max>

A1701E: 16-bit polynomial type cannot be used for source operand

A1702E: Register Dm can not be scalar for this instruction

A1704E: Register Dm must be in the range D0-D<upper> for this data type

A1705E: Assembler converted Qm register to D<rnum>[<idx>]

A1706E: Register Dm must be scalar

A1707I: <command> line

A1708E: 3rd operand to this instruction must be a constant expression

Copyright © 2006-2008 ARM Limited. All rights reserved.

A1709E: Expected ARM or scalar register expression

A1710E: Difference between current and previous register should be <diff>

A1711E: Scalar registers cannot be used in register list for this instruction

A1712W: This combination of LSB and WIDTH results in UNPREDICTABLE behaviour

A1713E: Invalid field specifiers for APSR: must be APSR_ followed by at least one of n, z, c, v, q or g

A1714E: Invalid combination of field specifiers for APSR

A1715E: PSR not defined on target architecture

A1716E: Destination for VMOV instruction must be ARM integer, 32-bit single-precision, 64-bit doubleword register or 64-bit doubleword scalar register

A1717E: Source register must be an ARM integer, 32-bit single-precision or 64-bit doubleword scalar register

A1718E: Source register must be an ARM integer register or same as the destination register

A1719W: This PSR name is deprecated and may be removed in a future release

A1720E: Source register must be a 64-bit doubleword scalar register

A1721E: Destination register may not have all-lanes specifier

A1722E: Labels not allowed inside IT blocks

A1723E: RELOC is deprecated, please use the new RELOC directive

A1724E: RELOC may only be used immediately after an instruction or data generating directive

A1725W: 'armasm inputfile outputfile' form of command-line is deprecated

A1726E: Decreasing --max_cache below 8MB is not recommended

A1727W: Immediate could have been generated using the 16-bit Thumb MOVS instruction

A1728E: Source register must be same type as destination register

A1729E: Register list may only contain 32-bit single-precision or 64-bit doubleword registers

A1730E: Only IA or DB addressing modes may be used with these instructions

A1731E: Register list increment of 2 or more is not allowed for quadword registers

A1732E: Register list must contain between 1 and 4 contiguous doubleword registers

A1733E: Register list must contain 2 or 4 doubleword registers, and increment 2 is only allowed for 2 registers

A1734E: Register list must contain {n|%ld} doubleword registers with increment 1 or 2

A1735E: Post-indexed offset must equal the number of bytes loaded/stored (<n>)

A1736E: Number of registers in list must equal number of elements

Copyright © 2006-2008 ARM Limited. All rights reserved.

A1737E: PC or SP can not be used as the offset register

A1738E: Immediate too large for this operation

A1739W: Constant generated using single VMOV instruction; second instruction is a NOP

A1740E: Number of bytes in FRAME PUSH or FRAME POP directive must not be less than zero

A1741E: Instruction cannot be conditional

A1742E: Expected LSL #Imm

A1744E: Alignment on register must be a multiple of 2 in the range 16 to 256

A1745W: This register combination is DEPRECATED

A1746W: Instruction stall diagnostics may be unreliable for this CPU

A1753E: Unrecognised memory barrier option

A1754E: Cannot change the type of a scalar register

A1755E: Scalar index has already been specified on this register

A1756E: Data type must be specified on all registers

A1762E: Branch offset 0x<val> out of range of 16-bit Thumb branch, but offset encodable in 32-bit Thumb branch.

A1763W: Inserted an IT block for this instruction

A1764W: <name> instructions are deprecated in architecture <arch> and above

Copyright © 2006-2008 ARM Limited. All rights reserved.

4. ARM Linker (armlink) Errors and Warnings

All linker warnings are suppressible with "--diag_suppress" in the same way as for compiler warnings, e.g "--diag_suppress 6306".

Some errors such as L6220E, L6238E and L6784E can be downgraded to a warning by using "--diag_warning".

`L6000U: Out of memory.`
This may occur when linking very large objects/libraries together, or you have very large regions defined in your scatter-file. In these cases, your workstation may run out of (virtual) memory.

`L6001U: Could not read from file <filename>.`
`L6002U: Could not open file <filename>: <reason>.`
`L6003U: Could not write to file <filename>.`
An file I/O error occurred while reading/opening/writing to the specified file.

`L6004U: Missing library member in member list for <library>.`
This can occur where there is whitespace in the list of library objects. See below:

Fails:
```
armlink x.lib(foo.o, bar.o)
Fatal error: L6004U: Missing library member in member list for x.lib.
```

Succeeds:
```
armlink x.lib(foo.o,bar.o)
```

Another less common occurrence is caused by a corrupt library, or possibly a library in an unsupported format.

`L6005U: Extra characters on end of member list for <library>.`

`L6007U: Could not recognize the format of file <filename>.`
The linker can recognize object files in the ELF format, and library files in AR formats. The specified file is either corrupt, or is in a file format that the linker cannot recognize. The file could be a AOF or ALF format which was produced by SDT. These file formats became deprecated in RVCT 2.1 and obsolete in 2.2.

`L6008U: Could not recognize the format of member <mem> from <lib>.`
The linker can recognize library member objects in the ELF file format. The specified library member is either corrupt, or is in a file format that the linker cannot recognize. The file could be a AOF or ALF format which was produced by SDT. These file formats became deprecated in RVCT 2.1 and obsolete in 2.2.

`L6009U: File <filename> : Endianness mismatch.`
The endianness of the specified file/object did not match the endianness of the other input files. The linker can handle input of either big endian or little endian objects in a single link step, but not a mixed input of some big and some little endian objects.

`L6010U: Could not reopen stderr to file <filename>: <reason>.`
An file I/O error occurred while reading /opening/writing to the specified file.

`L6011U: Invalid integer constant : <number>.`

Copyright © 2006-2008 ARM Limited. All rights reserved.

Specifying an illegal integer constant causes this. An integer can be entered in hexadecimal format by prefixing '&' or '0x' or '0X'. A suffix of 'k' or 'm' can be used to specify a multiple of 1024 or 1024*1024.

L6012U: Missing argument for option '<option>'.
The specified option requires an argument.

L6013U: Relocation #NN in <objname>(<secname>) has invalid/unknown type.
See L6027U.

L6014U: Unrecognised option <option>.
The linker does not recognize this option. This could be due to a spelling error, or due to the use of an unsupported abbreviation of an option.

L6015U: Could not find any input files to link.
The linker must be provided with at least one object file to link.
Example:
If you try to link with

```
armlink -o foo.axf
```

you will get the above error. Instead, you must use, for example:

```
armlink foo_1.o foo_2.o -o foo.axf
```

L6016U: Symbol table missing/corrupt in object/library <object>.
This may occur when linking with libraries built with the GNU tools. This is because GNU 'ar' can generate incompatible information. The workaround is to replace 'ar' with 'armar' and use the same command line arguments. Alternatively, the error is recoverable by using "armar -s" to rebuild the symbol table.

L6017U: Library <library> symbol table contains an invalid entry.
The library may be corrupted - try rebuilding it.

L6018U: <filename> is not a valid ELF file.
L6019U: <filename> is not a valid 64 bit ELF file.
L6020U: <filename> is not a valid 32 bit ELF file.

L6021U: Symbol <symbol> has unsupported attribute <attribute>.
L6022U: Object <objname> has multiple <table>.
L6023U: <objecttype> object <objname> does not contain any <part>.
The object file is faulty or corrupted. This may indicate a compiler fault – please contact your supplier.

L6024U: Library <library> contains an invalid member name.
L6025U: Cannot extract members from a non-library file <library>.
The file specified is not a valid library file, is faulty or corrupted - try rebuilding it.

L6026U: ELF file <filename> has neither little or big endian encoding
The ELF file is invalid - try rebuilding it.

L6027U: Relocation #NN in <objname>(<secname>) has invalid/unknown type.
This may occur in rare cases when linking legacy SDT AOF objects with the RVCT linker. Note: Support for AOF was deprecated in RVCT 2.1 and became obsolete in RVCT 2.2.

Some obscure AOF relocations cannot be translated into ELF, and are faulted. If so, the linker may report e.g.:

```
Error : (Fatal) L6027U: Relocation #17 in obj.o (SYMBOL_NAME) has
invalid/unknown type.
```


Copyright © 2006-2008 ARM Limited. All rights reserved.

To resolve this, the object/library must be rebuilt with RVCT.

`L6028U: Relocation #NN in <objname>(<secname>) has invalid offset.`
The relocation has an invalid offset. This may indicate a compiler fault – please contact your supplier.

`L6029U: Relocation #NN in <objname>(<secname>) is wrt invalid/missing symbol.`
The relocation is with respect to a symbol, which is either invalid or missing from the object symbol table, or is a symbol that is not suited to be used by a relocation. This may indicate a compiler fault – please contact your supplier.

`L6031U: Could not open scatter description file <filename>: <reason>.`
An I/O error occurred while trying to open the specified file. This could be due to an invalid filename.

`L6032U: Invalid <text> <value> (maximum <max_value>) found in <object>`

`L6033U: Symbol <symbolname> in <objname> is defined relative to an invalid section.`
When linking with GNU C libraries, the error may occur as:
`L6033U: Symbol in crt1.o is defined relative to an invalid section`
In the CodeSourcery 2006-Q1-3 release, the crt1.o object file has not been correctly stripped. This has been fixed in the 2006-Q1-6 CodeSourcery release. Alternatively you can strip the crt1.o object yourself.
Otherwise, the object file is faulty or corrupted. This may indicate a compiler fault – please contact your supplier.

`L6034U: Symbol <symbolname> in <objname> has invalid value.`
This can be caused by a section relative symbol having a value that exceeds the section boundaries. This may indicate a compiler fault – please contact your supplier.

`L6035U: Relocation #NN in ZI Section <objname>(<secname>) has invalid type.`
ZI Sections cannot have relocations other than of type R_ARM_NONE.

`L6036U: Could not close file <filename>: <reason>.`
An I/O error occurred while closing the specified file.

`L6037U: '<arg>' is not a valid argument for option '<option>'.`
The argument is not valid for this option. This could be due to a spelling error, or due to the use of an unsupported abbreviation of an argument.

`L6038U: Could not create a temporary file to write updated SYMDEFS.`
An I/O error occurred while creating the temporary file required for storing the `SYMDEFS` output.

`L6039U: Multiple --entry options cannot be specified.`
Only one instance of `--entry` is permitted, to specify the unique entry point for the ELF image.

`L6040U: Object <objname> contains corrupt symbol table entry for symbol <symbolname>.`
The object file is faulty or corrupted. This may indicate a compiler fault – please contact your supplier.

`L6041U: An internal error has occurred (<clue>).`
Contact your supplier.

`L6042U: Relocation #NN in <objname>(<secname>) is wrt a mapping symbol(<idx>, Last Map Symbol = <last>)`

Copyright © 2006-2008 ARM Limited. All rights reserved.

Relocations with respect to mapping symbols are not allowed. This may indicate a compiler fault – please contact your supplier.

`L6043U: Relocation Relocation #<rel_number> in <objname>(<secname>) is wrt an out of range symbol(#<val>, Range = 1-<max>).`

Relocations can only be made wrt symbols in the range (1-n), where n is the number of symbols.

`L6044U: Invalid relocation <rel_number> in <objname>(<secname>). Type <type> is reserved for ARM LINUX.`

`L6045U: Invalid relocation <rel_number> in <objname>(<secname>). Type <type> is reserved for the GNU tool chain.`

`L6046U: Recursive via file inclusion depth of <limit> reached`

`L6047U: The code in this image is <actual_size> bytes - this version of the linker will not create images that large`

`L6175E: EMPTY region <regname> cannot have any section selectors.`

`L6176E: A negative max_size cannot be used for region <regname> without the EMPTY attribute.`

Only regions with the EMPTY attribute are allowed to have a negative max-size.

`L6177E: A negative max_size cannot be used for region <regname> which uses the +offset form of base address.`

Regions using the +offset form of base address are not allowed to have a negative max-size.

`L6188E: Special section <sec1> multiply defined by <obj1> and <obj2>.`

A “special” section is one that can only be used once, such as “Veneer$$Code”.

`L6195E: Cannot specify both '<attr1>' and '<attr2>' for region <regname>`

`L6199E: Number string '<number>' contains invalid character(s) '<badchar>'.`

Number must not contain characters that are not valid digits for the base.

`L6200E: Symbol <symbol> multiply defined (by <object1> and <object2>).`

There is one common example where this occurs:

1) `Symbol __stdout multiply defined (by retarget.o and stdio.o).`

   This means that there are two conflicting definitions of `__stdout` present – one in `retarget.o`, the other in `stdio.o`. The one in `retarget.o` is your own definition. The one in `stdio.o` is the default implementation, which was probably linked-in inadvertently.

   `stdio.o` contains a number symbol definitions and implementations of file functions like `fopen`, `fclose`, `fflush`, etc. `stdio.o` is being linked-in because it satisfies some unresolved references.

   To identify why stdio.o is being linked-in, you must link with the linker's "verbose" switch, e.g.:

   ```
   armlink [... your normal options...] --verbose --errors err.txt
   ```

   Then study `err.txt`, to see exactly what the linker is linking-in, from where, and why.

   You may have to either:
   - Eliminate the calls like `fopen`, `fclose`, `fflush`, etc, or
   - Re-implement the `_sys_xxxx` family of functions.

   See the RVCT 3.0 Compilers and Libraries Guide, section 5.11, "Tailoring the input/output functions".

Copyright © 2006-2008 ARM Limited. All rights reserved.

`L6201E: Object <objname> contains multiple entry sections.`

`L6202E: <objname>(<secname>) cannot be assigned to non-root region '<regionname>'`

A root region is a region which has an execution address the same as its load address, and so the region does not need to be moved/copied by the scatter load initialisation code.

Certain sections must be placed in root region in the image. __main.o and the two linker-generated tables (Region$$Table and ZISection$$Table) must be in a root region. If not, the linker will report, for example:
`L6202E: Region$$Table cannot be assigned to a non-root region.`

In RVCT 2.1, a new region tables format was introduced to support the new compression mechanisms. This new format no longer contains ZISection$$Table. Furthermore, new scatter-loading (__scatter*.o) and decompressor (__dc*.o) objects from the library must be placed in a root region. These can all be placed together using InRoot$$Sections, e.g:

```
ROM_LOAD 0x0000 0x4000
{
    ROM_EXEC 0x0000 0x4000          ; root region
    {
        vectors.o (Vect, +FIRST)    ; Vector table
        * (InRoot$$Sections)        ; All library sections
                                    ; that must be in a root region
                                    ; for example, __main.o, __scatter*.o,
                                    ; dc*.o and * Region$$Table
    }
    RAM 0x10000 0x8000
    {
        * (+RO, +RW, +ZI)         ; all other sections
    }
}
```

Please see http://www.arm.com/support/rvds3_faq.html for more information.

`L6203E: Entry point (<address>) lies within non-root region <regionname>.`
`L6204E: Entry point (<address>) does not point to an instruction.`
`L6205E: Entry point (<address>) must be word aligned for ARM instructions.`
`L6206E: Entry point (<address>) lies outside the image.`

The image entry point must correspond to a valid instruction in the root-region of the image.

`L6208E: Invalid argument for --entry command`

`L6209E: Invalid offset constant specified for --entry (<arg>)`

`L6210E: Image cannot have multiple entry points. (<address1>,<address2>)`

An ELF image can have only one unique entry point. Specify the unique entry point with --entry.

`L6211E: Ambiguous section selection. Object <objname> contains more than one section.`

This can occur when using the linker option `--keep` on an assembler object that contains more than one `AREA`. The linker needs to know which `AREA` you would like to keep.
To solve this, specify the names of the AREAs that you wish to keep, using more than one `--keep` option, for example: `--keep boot.o(vectors) --keep boot.o(resethandler)...`
Note that using assembler files with more than one `AREA` may give other problems elsewhere, so this is best avoided.

`L6212E: <symbolname> multiply defined (by <object1> and <object2>) at different offsets in a COMMON section.`

Copyright © 2006-2008 ARM Limited. All rights reserved.

See L6200E.

L6213E: Multiple First section <object2>(<section2>) not allowed.
<object1>(<section1>) already exists.

Only one FIRST section is allowed.

L6214E: Multiple Last section <object2>(<section2>) not allowed.
<object1>(<section1>) already exists.

Only one LAST section is allowed.

L6215E: Ambiguous symbol selection for --first/--last. Symbol <symbol> has more than one definition.

L6216E: Cannot use base/limit symbols for non-contiguous section <sectionname>

Certain sections must be placed contiguously within the same region, for their base/limit symbols to be accessible.

For example:

```
LOAD_ROM 0x00000000
{
      ER1 0x00000000
      {
            file1.o (+RO) ; from a C++ source
            * (+RO)
      }
      ER2 0x01000000
      {
            file2.o (+RO) ; from a C++ source
      }
      ER3 +0
      {
            * (+RW, +ZI)
      }
}
```

will produce this error because the base and limit symbols for file1.o and file2.o are in separate regions:

```
L6216E: Cannot use base/limit symbols for non-contiguous section .init_array
```

The following code shows the corrected example:

```
LOAD_ROM 0x00000000
{
      ER1 0x00000000
      {
            file1.o (+RO) ; from a C++ source
            * (.init_array)
            * (+RO)
      }
      ER2 0x01000000
      {
            file2.o (+RO) ; from a C++ source
      }
      ER3 +0
      {
            * (+RW, +ZI)
      }
}
```


Copyright © 2006-2008 ARM Limited. All rights reserved.

Now the base and limit symbols are contained in `.init_array` in a single region.

`L6217E: Section <objname>(<secname>) contains R_ARM_SBREL32 relocation (#NN) wrt imported symbol <sym>`

`L6218E: Undefined symbol <symbol> (referred from <objname>).`

<objname> refers to <symbol>, but <symbol> is not defined anywhere. You must either provide a definition of <symbol> or remove the reference to <symbol>.

There are three common examples where this occurs:

1) `Undefined symbol __ARM_switch8` or `__ARM_ll_<xxxx>` functions

These functions have been moved and are now contained in the h_... libraries (h indicates that these are compiler helper libraries, rather than standard C library code). Please ensure that these libraries can be found by the linker.

2) `Undefined symbol __rt_embeddedalloc_init (referred from entry.o)`

The function __rt_embeddedalloc_init() was used in SDT embedded projects to set up a heap. This is no longer needed in RVCT projects, so the call to it must be removed. You should also remove your implementation of __rt_heapdescriptor() (if there is one).

3) This error may occur when attempting to refer to a function/entity in C from a function/entity in C++. This is caused by C++ name mangling, and can be avoided by marking C functions ‘extern "C"’

`L6219E: <type> section <object1>(<section1>) attributes {<attributes>} incompatible with neighbouring section <object2>(<section2>).`

This error occurs when the linker’s default ordering rules of RO followed by RW followed by ZI are violated. This typically happens when one uses +FIRST or +LAST, e.g. in a scatter file, attempting to force RW before RO.

`L6220E: Load/Execution region <regionname> size (<size> bytes) exceeds limit (<limit> bytes).`

Example:

`L6220E: Execution region ROM_EXEC size (4208184 bytes) exceeds limit (4194304 bytes).`

This can occur where a region has been given an (optional) maximum length in the scatter-file, but this size of the code/data being placed in that region has exceeded the given limit. This error is suppressible with "`--diag_suppress 6220`".

`L6221E: <type1> region <regionname1> overlaps with <type2> region <regionname2>.`

`L6222E: Partial object cannot have multiple ENTRY sections`

Where objects are being linked together into a partially-linked object, only one of the sections in the objects may have an entry point. Note: It is not possible here to use the linker option `--entry` to select one of the entry points.

`L6223E: Ambiguous selectors found for <objname>(<secname>) from Exec regions <region1> and <region2>.`

This will occur if the scatter-file specifies `<objname>(<secname>)` to be placed in more than one execution region. This can occur accidentally when using wildcards ('*'). The solution is to make the selections more specific in the scatter-file.

`L6224E: Could not place <objname>(<secname>) in any Execution region.`

Copyright © 2006-2008 ARM Limited. All rights reserved.

L6225E: Number <str...> is too long.

L6226E: Missing base address for region <regname>.

L6227E: Using --reloc with --rw-base without --split is not allowed.

L6228E: Expected '<str1>', found '<str2>'.

L6229E: Scatter description <filename> is empty.

L6230E: Multiple execution regions (<region1>,<region2>) cannot select <secname>.

L6231E: Missing module selector.

L6232E: Missing section selector.

L6233E: Unknown section selector '+<selector>'.

L6234E: <str> must follow a single selector.

e.g. in a scatter file:

```
:
* (+FIRST, +RO)
:
```

+FIRST means "place this (single) section first", therefore selectors which can match multiple sections (e.g. +RO, +ENTRY, etc) are not allowed to be used with +FIRST (or +LAST), hence the error message.

L6235E: More than one section matches selector - cannot all be FIRST/LAST.

L6236E: No section matches selector - no section to be FIRST/LAST.

The scatter-file specifies a section to be +FIRST or +LAST, but that section does not exist, or has been removed by the linker because it believes it to be unused. Use the linker option "--info unused" to reveal which objects are removed from your project. Example:

```
ROM_LOAD 0x00000000 0x4000
{
    ROM_EXEC 0x00000000
    {
        vectors.o (Vect, +First)   << error here
        * (+RO)
    }
    RAM_EXEC 0x40000000
    {
        * (+RW, +ZI)
    }
}
```

Some possible solutions are:

a) ensure vectors.o is specified on the linker command-line.
b) link with "--keep vectors.o" to force the linker not to remove this, or switch off this optimization entirely, with --noremove [not recommended]
c) [Recommended] Add the ENTRY directive to vectors.s, to tell the linker that it is a possible entry point of your application, e.g.:

```
AREA Vect, CODE
ENTRY    ; define this as an entry point
```


Copyright © 2006-2008 ARM Limited. All rights reserved.

```
Vector_table
        ...
```
and then link with "`--entry 0x0`" to define the real start of your code.

`L6237E: <objname>(<secname>) contains relocation(s) to unaligned data.`

`L6238E: <objname>(<secname>) contains invalid call from '<attr1>' function to '<attr2>' function <sym>.`

This linker error is given where a stack alignment conflict is detected in object code. The "ABI for the ARM Architecture" demands that code maintains 8-byte stack alignment at its interfaces. This allows efficient use of LDRD and STRD instructions (in ARM Architecture 5TE and later) to access 8-byte-aligned "double" and "long long" data types.

Symbols like '~PRES8' and 'REQ8' are "Build Attributes" of the objects. PRES8 means the object PREServes 8-byte alignment of the stack. ~PRES8 means the object does NOT preserve 8-byte alignment of the stack (~ meaning NOT). REQ8 means the object REQuires 8-byte alignment of the stack.

This link error typically occurs in two cases:

1) where assembler code (that does not preserve 8-byte stack alignment) calls compiled C/C++ code (that requires 8-byte stack alignment).

2) when attempting to link legacy SDT/ADS objects with RVCT 3.x objects. Legacy SDT/ADS objects that do not have these attributes are treated as '~PRES8', even if they do actually happen to preserve 8-byte alignment.

For example:
```
Error: L6238E: foo.o(.text) contains invalid call from '~PRES8' function to
'REQ8' function foobar
```

This means that there is a function in the object foo.o (in the section named .text) that does not preserve 8-byte stack alignment, but which is trying to call function foobar that requires 8-byte stack alignment.

A similar warning that may be encountered is:
```
Warning: L6306W: '~PRES8' section foo.o(.text) should not use the address
of 'REQ8' function foobar
```
where the address of an external symbol is being referred to.

There are two possible solutions to work-around this issue:

1) Rebuild all your objects/libraries using RVCT 3.x.
If you have any assembler files, you will need to check that all instructions preserve 8-byte stack alignment, and if necessary, correct them.
e.g. change:
```
        STMFD sp!, {r0-r3, lr}  ; push an odd number of registers
```
to
```
        STMFD sp!, {r0-r3, r12, lr}  ; push an even number of registers
```
The assembler will automatically mark the object with the PRES8 attribute if all instructions preserve 8-byte stack alignment, so it is no longer necessary to add the PRESERVE8 directive to the top of each assembler file.

Copyright © 2006-2008 ARM Limited. All rights reserved.

2) If you have any legacy objects/libraries that cannot be rebuilt, either because you do not have the source code, or because the old objects must not be rebuilt (e.g. for qualification/certification reasons), then you must inspect the legacy objects to check whether they preserve 8-byte alignment or not. Use "fromelf -c" to disassemble the object code. C/C++ code compiled with ADS 1.1 or later will normally preserve 8-byte alignment, but assembled code will not.

If your objects do indeed preserve 8-byte alignment, then the linker error L6238E can be suppressed with the use of "--diag_suppress 6238" on the linker command line. By using this, you are effectively saying "I guarantee that these objects are PRES8". The linker warning L6306W is suppressible with "--diag_suppress 6306".

More information about linking with legacy objects/libraries and the "--apcs /adsabi" is given at: http://www.arm.com/support/faqdev/1242.html

`L6239E: Cannot call non-interworking <ARM/THUMB> symbol '<sym>' in <obj> object from <ARM/THUMB> code in <obj1>(<sec1>)`

Example:

```
L6239E: Cannot call non-interworking ARM symbol 'ArmFunc' in object foo.o from
THUMB code in bar.o(.text)
```

This problem may be caused by foo.c not being compiled with the option "`--apcs /interwork`", to enable ARM code to call Thumb code (and vice-versa) via Linker-generated interworking veneers.

`L6241E: <objname>(<secname>) cannot use the address of '<attr1>' function <sym> as the image contains '<attr2>' functions.`

When linking with '--strict', the linker reports conditions that might fail as errors, for example:

```
Error: L6241E: foo.o(.text) cannot use the address of '~IW' function main as
the image contains 'IW' functions.
```

'IW' means "interworking", '~IW' means "non-interworking"

`L6242E: Cannot link object <objname> as its attributes are incompatible with the image attributes.`

There are three common reasons for this error message:

1.

```
Error: L6242E: Cannot link object foo.o as its attributes are incompatible
with the image attributes.
... require 4-byte alignment of 8-byte datatypes clashes with require
8-byte alignment of 8-byte datatypes.
```

This can occur when linking RVCT (or later), objects with legacy objects built with SDT, ADS or RVCT 1.2. The Application Binary Interface (ABI) was changed between ADS and RVCT 2.0.

In SDT, ADS and RVCT 1.2, "double" and "long long" data types were 4-byte aligned (unless -Oldrd or __align were used). In RVCT 2.0 onwards, "double" and "long long" data types are now 8-byte aligned, according to the new EABI. This allows efficient use of LDRD and STRD instructions in ARM Architecture 5TE and later.

These changes mean that legacy SDT/ADS/RVCT1.2 objects/libraries using "double" or "long long" data types are not directly compatible with RVCT 3.x objects/libraries, and so the linker will report an attribute clash.

Some compatibility is made possible, with some restrictions, by way of the "--apcs /adsabi" switch in RVCT 3.x. To allow RVCT 3.x C objects to be used with legacy SDT/ADS C objects, compile the RVCT 3.x C code with "--apcs /adsabi".

Copyright © 2006-2008 ARM Limited. All rights reserved.

2. `Error: L6242E: Cannot link object foo.o as its attributes are incompatible with the image attributes.`
   `... pure-endian double clashes with mixed-endian double.`

   This can occur when linking RVCT 3.x or ADS objects with legacy objects built with SDT. The byte order of 'double' and 'long long' types changed between SDT and ADS.

   In SDT, the formats of little-endian 'double' and big-endian 'long long' are nonstandard. The ADS/RVCT compilers and assembler support industry-standard 'double' and 'long long' types in both little-endian and big-endian formats.

   If you try to link an ADS/RVCT object with an SDT object, all built with the normal defaults, the linker will report an attribute clash.

   Again, the recommended solution is to rebuild your entire project with RVCT. If you do not have the source code for an object or library, then try recompiling your RVCT code with '--fpu softfpa'.

3. `Error: L6242E: Cannot link object foo.o as its attributes are incompatible with the image attributes.`
   `... FPA clashes with VFP.`

   This error typically occurs when attempting to link objects built with different --fpu options. The recommended solution is to rebuild your entire project with RVCT, with the same --fpu options.

`L6243E: Selector only matches removed unused sections - no section to be FIRST/LAST.`
All sections matching this selector have been removed from the image because they were unused. For more information, use `--info unused`.

`L6244E: Load/Execution region <regionname> address (<addr>) not aligned on a <align> byte boundary.`

`L6245E: Failed to create requested ZI section '<name>'.`

`L6246E: Invalid memory access attributes '<attr>' specified for Execution region <region>`

`L6247E: Memory attributes of <objname>(<secname>) incompatible with those of parent Execution region <regname>.`

`L6248E: <objname>(<secname>) in <attr1> region '<r1>' cannot have <rtype> relocation to <symname> in <attr2> region '<r2>'.`
Example:
`L6248E: foo.o(areaname) in ABSOLUTE region 'ER_RO' cannot have address/offset type relocation to symbol in PI region 'ER_ZI'.`
See Compiler `#1359`. See also FAQ "What does "Error: L6248E: cannot have address type relocation" mean?" at: http://www.arm.com/support/rvds3_faq.html

`L6249E: Entry point (<address>) lies within multiple sections.`

`L6250E: Object <objname> contains illegal definition of special symbol <symbol>.`
`L6251E: Object <objname> contains illegal reference to special symbol <symbol>.`

`L6252E: Invalid argument for --xreffrom/--xrefto command: <arg>`

`L6253E: Invalid SYMDEF address: <number>.`
`L6254E: Invalid SYMDEF type : <type>.`

Copyright © 2006-2008 ARM Limited. All rights reserved.

The content of the symdefs file is invalid.

`L6255E: Could not delete file <filename>: <reason>.`

An I/O error occurred while trying to delete the specified file. The file was either read-only, or was not found.

`L6256E: Could not rename file <oldname> to <newname>: <reason>`

An I/O error occurred while trying to rename the specified file. File specified by `newname` may already exist.

```
L6257E: <object>(<secname>) cannot be assigned to overlaid Execution region.
L6258E: Entry point (<address>) lies in an overlaid Execution region.
L6259E: Reserved Word '<name>' cannot be used as a Load/Execution region name.
L6260E: Multiple load regions with the same name (<regionname>) are not allowed.
L6261E: Multiple execution regions with the same name (<regionname>) are not allowed.
```

The above 5 error messages all relate to a problem with the scatter file.

```
L6262E: Cannot relocate wrt symbol <symbol> (defined at non-zero offset in COMMON
section <objname>(<secname>)).
```

Relocations to a `COMMON` section are permitted only through a section relative symbol with zero offset. This error may indicate a compiler fault – please contact your supplier.

```
L6263E: <addr> address of <regionname> cannot be addressed from <pi_or_abs> Region
Table in <regtabregionname>
```

where <addr> is a string. It can take the value of:
`Load, Relocatable Load, Execution, Relocatable Execution`

```
Obsolete from 2.2 onwards; L6264E: Cannot express ZISection Table entry for
<regionname> as either address or offset.
```

```
L6265E: Non-RWPI Section <obj>(<sec>) cannot be assigned to PI Exec region <er>.
L6266E: RWPI Section <obj>(<sec>) cannot be assigned to non-PI Exec region <er>.
```

This may be caused by explicitly specifying the (wrong) ARM-supplied library on the linker command-line. You should not normally need to specify any ARM libraries explicitly e.g. `(c_t__ue.b)` on the link-line.

`L6268E: Non-word aligned address <addr> specified for region <regname>.`

`L6269E: Missing expected <ch>.`

```
L6271E: Two or more mutually exclusive attributes specified for Load region
L6272E: Two or more mutually exclusive attributes specified for Execution region
L6273E: Section <object2>(<section2>) has mutually exclusive attributes (READONLY and
ZI)
L6274E: Ignoring unknown <attr> attribute <subattr> specified for region <regname>.
```

The above 4 error messages all relate to a problem with the scatter file.

```
L6275E: COMMON section <obj1>(<sec1>) does not define <sym> (defined in
<obj2>(<sec2>))
```

Given a set of `COMMON` sections with the same name, the linker selects one of them to be added to the image and discards all others. The selected `COMMON` section must define all the symbols defined by any rejected `COMMON` section, otherwise, a symbol which was defined by the rejected section now becomes undefined again. The linker will generate an error if the selected copy does not define a symbol that a rejected copy does. This error would normally be caused by a compiler fault – please contact your supplier.

Copyright © 2006-2008 ARM Limited. All rights reserved.

L6276E: Address <addr> marked both as <s1>(from <sp1>(<obj1>) via <src1>) and <s2>(from <sp2>(<obj2>) via <src2>).

The image cannot contain contradictory mapping symbols for a given address, because the contents of each word in the image are uniquely typed as ARM ($a) or THUMB ($t) code, DATA ($d), or NUMBER. It is not possible for a word to be both ARM code and DATA. This may indicate a compiler fault – please contact your supplier.

L6277E: Unknown command '<cmd>'.

L6278E: Missing expected <str>.

L6279E: Ambiguous selectors found for <sym> ('<sel1>' and '<sel2>').

L6280E: Cannot rename <sym> using the given patterns.
L6281E: Cannot rename both <sym1> and <sym2> to <newname>.
L6282E: Cannot rename <sym1> to <newname> as a global symbol of that name exists (defined) in <obj>).

The RENAME command in the steering file is invalid.

L6283E: Object <objname> contains illegal local reference to symbol <symbolname>.

An object cannot contain a reference to a local symbol, since local symbols are always defined within the object itself.

L6284E: Cannot have multiple definitions of macro <macro_name>

Each macro can be defined only once. Multiple definitions of a macro (even using same value) are not permitted.

L6285E: Non-relocatable Load region <lr_name> contains R-Type dynamic relocations. First R-Type dynamic relocation found in <object>(<secname>) at offset 0x<offset>.

L6286E: Value(<val>) out of range(<range>) for relocation #<rel_number> (<rtype>, wrt symbol <symname>) in <objname>(<secname>)

This can typically occur in handwritten assembler code, where the limited number of bits for a field within the instruction opcode is too small to refer to a symbol so far away. For example, for an LDR or STR where the offset is too large for the instruction (+/-4095 for ARM state LDR/STR instruction). In other cases, please make sure you have the latest patch installed from: http://www.arm.com/support/downloads. For more information about this please see http://www.arm.com/support/faqdev/1239.html

L6287E: Illegal alignment constraint (<align>) specified for <objname>(<secname>).

An illegal alignment was specified for an ELF object.

L6291E: Base address <addr> lies in the previous exec region or before the start of the load region
L6292E: Ignoring unknown attribute '<attr>' specified for region <regname>.
L6293E: FIXED is incompatible with relative base <offset> for region <regname>.
L6294E: Load/Execution region <regionname> spans beyond 32 bit address space (base <base>, size <size> bytes).

The above 4 error messages all relate to a problem with the scatter file.

L6295E: SB Relative relocation (in section <object>(<secname>) at offset 0x<offset> wrt symbol <symname>) requires image to be RWPI

L6296E: Definition of special symbol <sym1> is illegal as symbol <sym2> is absolute.

L6297E: Definition of special symbol <sym1> is illegal as symbol <sym2> has synonyms (defined by <obj1>, <obj2>).

Copyright © 2006-2008 ARM Limited. All rights reserved.

`L6298E: Invalid definition of macro <macro_name>`

A macro definition is invalid if the macro name or value is missing.

`L6299E: Undefined macro <macro_name>`

A macro needs to be defined before it can be used. No definition of the specified macro was found.

`L6300W: Common section <object1>(<section1>) is larger than its definition <object2>(<section2>).`

This may indicate a compiler fault; please contact your supplier.

`L6301W: Could not find file <filename>: <reason>.`

The specified file was not found in the default directories.

`L6302W: Ignoring multiple SHLNAME entry.`

There can be only one `SHLNAME` entry in an edit file. Only the first such entry is accepted by the linker. All subsequent `SHLNAME` entries are ignored.

`L6303W: Symbol <symbol> multiply defined (by <object1> and <object2>).`

See `L6200E`.

`L6304W: Duplicate input file <filename> ignored.`

The specified filename occurred more than once in the list of input files.

`L6305W: Image does not have an entry point. (Not specified or not set due to multiple choices.)`

The entry point for the ELF image was either not specified, or was not set because there was more than one section with an entry point linked-in. You must specify the single, unique entry point with the linker option `--entry`, e.g. `--entry 0x0` or `--entry <label>` is typical for an embedded system.

`L6306W: '<attr1>' section <objname>(<secname>) should not use the address of '<attr2>' function <sym>.`

See `L6238E`.

`L6307W: <objname>(<secname>) contains branch to unaligned destination.`

`L6308W: Could not find any object matching <membername> in library <libraryname>.`

The name of an object in a library is specified on the link-line, but the library does not contain an object with that name.

`L6309W: Library <libraryname> does not contain any members.`

A library is specified on the link-line, but the library does not contain any members.

`L6310W: Unable to find ARM libraries.`

This is most often caused by a missing or invalid value of the environment variable RVCT3xLIB or by incorrect arguments to --libpath. For example RVCT30LIB needs to be set when RVDS 3.0 is installed. Make sure this matches with the tools you are using.

Alternatively, try specifying the path explicitly using --libpath switch. The default for a normal Windows installation will be: "`C:\Program Files\ARM\RVCT\Data\3.`*`x`*`\`*`build`*`\lib`". Make sure this path does not include "\armlib", "\cpplib" or any trailing slashes ("\") at the end as these will be added by the linker automatically. Use "`--verbose`" to display where the linker is attempting to get the libraries from.

`L6311W: Undefined symbol <symbol> (referred from <objname>).`

Copyright © 2006-2008 ARM Limited. All rights reserved.

See L6218E.

L6312W: Empty Load/Execution region description for region <region>

L6313W: Using <oldname> as an section selector is obsolete. Please use <newname> instead.

For example, use of "IWV$$Code" within the scatterfile is now obsolete, so should be replaced with "Veneer$$Code".

L6314W: No section matches pattern <module>(<section>).

Example:

```
No section matches pattern foo.*o(ZI).
```

This can occur for two possible reasons:

1) The file foo.o is mentioned in your scatter-file, but it is not listed on the linker command-line. To resolve this, add foo.o to the link-line.

2) You are trying to place the ZI data of foo.o using a scatter-file, but foo.o does not contain any ZI data. To resolve this, remove the "+ZI" attribute from the foo.o line in your scatter-file.

L6315W: Ignoring multiple Build Attribute symbols in Object <objname>.

An object can contain at most one absolute BuildAttribute$$... symbol. Only the first such symbol from the object symbol table is accepted by the linker. All subsequent ones are ignored.

L6316W: Ignoring multiple Build Attribute symbols in Object <objname> for section <secno>

An object can contain at most one BuildAttribute$$... symbol applicable to a given section. Only the first such symbol from the object symbol table is accepted by the linker. All subsequent ones are ignored.

L6317W: <objname>(<secname>) should not use the address of '<attr1>' function <sym> as the image contains '<attr2>' functions.

L6318W: <objname>(<secname>) contains branch to a non-code symbol <sym>.

This warning means that in the (usually assembler) file, there is a branch to a non-code symbol (in another AREA) in the same file. This is most likely a branch to a label or address where there is data, not code. For example:

```
AREA foo, CODE
B  bar
AREA bar, DATA
DCD 0
END
```

gives:

```
init.o(foo) contains branch to a non-code symbol bar.
```

If the destination has no name, e.g:

```
BL 0x200 ; Branch with link to 0x200 bytes ahead of PC
```

you will see, e.g:

```
bootsys.o(BOOTSYS_IVT) contains branch to a non-code symbol <Anonymous Symbol>.
```

This warning may also appear when linking objects generated by GCC. GCC uses linker relocations for references internal to each object. The targets of these relocations may not have appropriate mapping symbols that allow the linker to determine whether the target is code or data, so a warning will be generated. By contrast, armcc resolves all such references at compile-time.

L6319W: Ignoring <cmd> command. Cannot find section <objname>(<secname>).

Copyright © 2006-2008 ARM Limited. All rights reserved.

For example, when building a Linux application, you may have e.g. "--keep *(.init_array)" on the linker command-line in your makefile, but this section may not be present, e.g. when building with no C++, in which case this warning is reported:
`L6319W: Ignoring --keep command. Cannot find section *(.init_array)`
You can often ignore this warning, or suppress it with --diag_suppress 6319

`L6320W: Ignoring <cmd> command. Cannot find argument '<argname>'.`
`L6321W: Ignoring <cmd>. Cannot be used without <prereq_cmd>.`

`L6322W: <n_cycles> cyclic references found while sorting <sec> sections.`

`L6323W: Multiple variants of <sym> exist. Using the <type> variant to resolve relocation #NN in <objname>(<secname>)`

`L6324W: Ignoring <attr> attribute specified for Load region <regname>.`
This attribute is applicable to execution regions only. If specified for a `Load` region, the linker ignores it.

`L6325W: Ignoring <attr> attribute for region <regname> which uses the +offset form of base address.`
This attribute is not applicable to regions using the `+offset` form of base address. If specified for a region, which uses the +offset form, the linker ignores it. A region, which uses the +offset form of base address, inherits the `PI/RELOC/OVERLAY` attributes from the previous region in the description, or the parent load region if it is the first execution region in the load region.

`L6326W: Ignoring ZEROPAD attribute for non-root execution region <ername>.`
`ZEROPAD` only applies to root execution regions. A root region is a region whose execution address is the same as its load address, and so does not need to be moved/copied at run-time.

`L6329W: Pattern <module>(<section>) only matches removed unused sections.`
All sections matching this pattern have been removed from the image because they were unused. For more information, use "`--info unused`". See RVCT 3.0 Linker and Utilities guide, section 3.3.3, "Unused section elimination"

`L6330W: Undefined symbol <symbol> (referred from <objname>). Unused section has been removed.`
See RVCT FAQ at http://www.arm.com/support/faqdev/5672.html

`L6331W: No eligible global symbol matches pattern <pattern>.`

`L6332W: Undefined symbol <sym1> (referred from <obj1>). Resolved to symbol <sym2>.`

`L6333W: Undefined symbol <symbol> (referred from <objname>). To be resolved during dynamic linking.`
This warning is produced when a symbol is undefined but the user has marked the symbol to be placed in the Dynamic symbol table. The message is only informational in content and may be ignored. This warning is suppressed by default.

`L6334W: Illegal alignment constraint (<align>) for <objname>(<secname>) ignored. Using 4 byte alignment.`

`L6335W: ARM interworking code in <objname>(<secname>) may contain invalid tailcalls to ARM non-interworking code.`
The compiler is able to perform tailcall optimisation for improved code size and performance. However, there is a problematic sequence for Architecture 4T code where a Thumb IW function calls (via a veneer) an ARM IW function, which tailcalls an ARM not-IW function. The return

Copyright © 2006-2008 ARM Limited. All rights reserved.

from the ARM not-IW function may pop the return address off the stack into the PC instead of using the correct BX instruction. The linker can warn of this situation and will report the above warning.

Thumb IW tailcalls to Thumb not-IW do not occur because Thumb tailcalls with B are so short ranged that they can only be generated to functions in the same ELF section which must also be Thumb.

The warning is pessimistic in that an object _might_ contain invalid tailcalls, but the linker cannot be sure because it only looks at the attributes of the objects, not at the contents of their sections.

To avoid the warning, either recompile your entire code base, including any user libraries, with `--apcs /interwork`, or manually inspect the ARM IW function to check for tailcalls (i.e. where function calls are made using an ordinary branch B instruction), to check whether this is a real problem. This warning can be suppressed with `--diag_suppress L6335W`.

`L6337W: Common code sections <o1>(<s1>) and <o2>(<s2>) have incompatible floating-point linkage`

`L6338W: Load/Execution region <regionname> at <offset> aligned to next <align> byte boundary.`

`L6339W: Ignoring RELOC attribute for execution region <regname>.`

Execution regions cannot explicitly be given RELOC attribute. They can only gain this attribute by inheriting from the parent load region or the previous execution region if using the +offset form of addresssing.

`L6340W: options first and last are ignored for link type of <linktype>`

The `--first` and `--last` options are meaningless when creating a partially-linked object

`L6341E: Address of <objname>(<secname>) (<base addr>) does not match the required address <reqd_base addr>`

`L6564I: Not enough information to list the image debug input and output sizes.`

`L6565I: Not eliminating unused sections as image is unsuitable for such optimization. Unused section elimination cannot be performed on this image.`

Instead of using "`--entry <address>`" use "`--entry <label>`" because this makes it easier for the linker to follow the call tree.

`L6566I: Loading symbol definitions from SYMDEFS file <fname>.`

`L6567I: Not enough information to produce a SYMDEFs file.`

The `--symdefs` option could not create a symdefs file because, e.g, linking failed to complete.

`L6568I: Not enough information to list image symbols.`

The `--symbols` option could not complete because, e.g, linking failed to complete.

`L6569I: Not enough information to list the image map.`

The `--map` option could not complete because, e.g, linking failed to complete.

`L6570I: Not enough information to list the image sizes and/or totals.`

The `--info sizes` or `totals` option could not complete because, e.g, linking failed to complete.

`L6602W: Unmatched literal pool end symbol <symname> ignored in file <filename>.`

Copyright © 2006-2008 ARM Limited. All rights reserved.

L6603W: Literal pool begin symbol <symname> found within Literal pool in file <filename>.
L6604E: Literal pool end symbol <symname1> is in different area from literal pool begin symbol <symname2> in file <filename>

These three relate to AOF to ELF object conversion. The AOF object may be invalid. Try recompiling the source file with RVCT to create a new ELF object. Support for AOF is deprecated in RVCT 2.1 and obsolete in RVCT 2.2.

L6627U: Bad error message list <list> for command <command>
L6629E: Unmatched parentheses expecting) but found <character> at position <col> on line <line>
L6630E: Invalid token start expected number or (but found <character> at position <col> on line <line>
L6631E: Division by zero on line <line>
L6632W: Subtraction underflow on line <line>
L6633E: Could not open intermediate file <filename> to send to pre-processor: <reason>
L6634E: Pre-processor command in '<filename>' too long, maximum length of <max_size>
L6635E: Could not open intermediate file '<filename>' produced by pre-processor: <reason>
L6636E: Pre-processor step failed for '<filename>'

The above 9 messages indicate a problem with pre-processing the scatter file.

L6637W: No input objects specified. At least one input object or library(object) must be specified.

At least one input object or library(object) must be specified.

L6638U: Object <objname> has a link order dependency cycle, check sections with SHF_LINK_ORDER

L6640E: PDTTable section not least static data address, least static data section is <secname>

L6641E: Cannot find base of consolidated output section for input sections <secname> as sections are not contiguous

L6642W: Unused virtual function elimination might not work correctly, because <obj_name> has not been compiled with –vfe

L6643E: The virtual function elimination information in section <sectionname> refers to the wrong section.
L6644E: Unexpectedly reached the end of the buffer when reading the virtual function elimination information in section <oepname>(<xxxx>).
L6645E: The virtual function elimination information in section <oepname>(<xxxx>) is incorrect: there should be a relocation at offset <offset>
L6646W: The virtual function elimination information in section <oepname>(<xxxx>) contains garbage from offset <offset> onwards.
L6647E: The virtual function elimination information for section <section> (object <object>) incorrectly indicates that section <section> (object <object>), offset <offset> is a relocation (to a virtual function or RTTI), but there is no relocation at that offset.

Any of the 5 above messages may indicate a compiler fault; please contact your supplier.

L6648U: Object <objname> built with <producer> does not match <toolkit>.

L6649E: EMPTY region <regname> must have a maximum size.

L6650E: Object <objname> Group section <sectionidx> contains invalid symbol index <symidx>.

L6651E: Section <secname> from object <objname> has SHF_GROUP flag but is not member of any group.

L6652E: Cannot reverse Byte Order of Data Sections, input objects are <inputendian> requested data byte order is <dataendian>.

L6653W: Dynamic relocations are not partitioned into code and data targets.

Copyright © 2006-2008 ARM Limited. All rights reserved.

Warning for RVCT2.0, full v6 byte invariant addressing is not available in this version. It is however supported in 2.1 onwards.

`L6654E: Local symbol <symname> from <secname> in <objname> is referred to from a non group section <nongrpname>`

If the one of the three Errors above is reported this may indicate a compiler fault; please contact your supplier.

`L6655I: Removing COMDAT group <group> from the image.`

`L6656E: Internal error: the vfe section list contains a non-vfe section called <oepname>(<secname>).`

This may indicate a compiler fault; please contact your supplier.

`L6657E: Resolve is not permitted for Inline Veneer Symbol <symname>`

`L6661U: Cannot split caller request`

This error can occur in rare cases where the linker needs to add a veneer, but within a narrow range of insertion. For example, where a function is called from a BL at a low address and a high address (but both still being in range), if the linker tries to insert a veneer within this range, but it finds out that the range lies within the boundary of an existing section, the linker must then duplicate the veneer and insert before and after the section so that both callers can reach the veneer. If this process fails, then the linker will give this error message. It may be possible for you to work around this by rearranging your scatter file. Later patch builds of the RVCT 2.1 tools and all RVCT 2.2 & 3.0 tools fix this problem.

`L6662E: Cannot add common section <secname> from <objname> to non-comdat group.`

`L6664W: Relocation #<rel_number> in <objname>(<secname>) is wrt a symbol(#<idx> before last Map Symbol #<last>).`

`L6665W: Neither Lib$$Request$$armlib Lib$$Request$$cpplib defined, not searching ARM libraries.`

This reproduces the warning:

```
  AREA Block, CODE, READONLY
  EXPORT func1
  ;IMPORT || Lib$$Request$$armlib||
  IMPORT printf
func1
  LDR r0,=string
  BL printf
  BX lr

  AREA BlockData, DATA
string  DCB "mystring"
  END
```

The linker has not been told to look in the libraries and hence cannot find the symbol "printf". This causes an error also: `L6218E: Undefined symbol printf (referred from L6665W.o).`

To fix both the error and the warning uncomment the line: "IMPORT || Lib$$Request$$armlib||".

`L6666I: Not enough information to produce a FEEDBACK file.`

`L6667I: Creating FEEDBACK file <filename>.`

`L6670W: --nodebug overrides --bestdebug, all debug sections will be removed.`

Copyright © 2006-2008 ARM Limited. All rights reserved.

L6676W: The intermediate decompressor for images containing overlapping data was not initialised correctly.

L6679W: Data in output ELF section #<sec> '<secname>' was not suitable for compression (<data_size> bytes to <compressed_size> bytes).

L6681E: Region table updated for compressed data sections was not written into the file.

L6682E: Merge Section <spname> from object <oepname> is a code section

L6683E: Merge Section <spname> from object <oepname> has an element size of zero

L6684E: Section <spname> from object <oepname> has SHF_STRINGS flag but not SHF_MERGE flag

L6685E: Section <spname> from object <oepname> has a branch reloc <rel_idx> to a SHF_MERGE section

L6686E: Section <spname> from object <oepname> has a SWI reloc <rel_idx> to a SHF_MERGE section

L6687E: Section <spname> from object <oepname> has a reloc <rel_idx> with an unsupported type to a SHF_MERGE section

L6688U: Section <spname> from object <oepname> has a relocation <rel_idx> that references a negative element

L6689U: Section <spname> from object <oepname> has a relocation <rel_idx> to the middle of a multibyte character

L6690U: Merge Section <spname> from object <oepname> has no local symbols

L6695U: Bad --diag_style argument <style>

L6703W: Section <er> implicitly marked as non-compressible

L6707E: Padding value not specified with PADVALUE attribute for execution region <regionname>

L6708E: Could not process debug frame from <secname> from object <oepname>

L6709E: Could not associate fde from <secname> from object <oepname>

L6713W: Function at offset <offset> in Section <secname> in Object <oepname> has no symbol.

L6714W: Exception index table section .ARM.exidx from object <oepname> has no data.

L6719U: Exception table generation failure <text>.

L6720U: Exception table <spname> from object <oepname> present in image, --noexceptions specified.

L6721E: Section #<secnum> '<secname>' in <oepname> is not recognized and cannot be processed generically.

L6722E: Linker defined symbol <namedsym> shadows <nummedsym>. All SHT linker defined symbol references to a user section must be to the same linker defined symbol.

L6724W: Support for <feature> shall be removed in a future version of the linker. You should <alternative>

Copyright © 2006-2008 ARM Limited. All rights reserved.

L6725W: Unused virtual function elimination might not work correctly, because there are dynamic relocations.

L6726W: Unknown Diagnostic number (<num>)

L6727W: The contents of '<er2>' may be corrupted during scatterloading, if placed behind '<er1>'.

L6728U: Link order dependency on invalid section number <to> from section number <from>.

L6730W: ABI type <type> differs from legacy behaviour <legacy_type> for target symbol <name> of relocation <index> from section <secname> from object <objname>.

A change in the linker behaviour gives warnings about strict compliance with the ABI. Example:

```
        AREA foo, CODE, READONLY

        CODE32
        ENTRY

   func proc
        nop
        endp

        dcd foo

        keep
        end
```

The warning is related to how the assembler marks sections for interworking. Previously, the section symbol foo would be marked as ARM or Thumb code in the ELF file. The dcd foo above would therefore also be marked as subject to interworking.

However, the ABI specifies that only functions should be subject to interworking and marked as ARM or Thumb. The linker will therefore warn that it is expecting dcd <number>, which does not match the symbol type (ARM, or THUMB if you use CODE16) of the area section.

The simplest solution is to move the data into a separate data area in the assembly source file.

Alternatively, you can use --diag_suppress 6730 to suppress this warning.

L6731W: Unused virtual function elimination might not work correctly, because the section referred to from <secname> does not exist.

L6733W: <objname>(<secname>) contains offset relocation from <lr1name> to <lr2name>, load regions must be rigidly relative.

L6734W: Ambiguous VFE setting: the <option> and --vfemode options should not be used simultaneously.

L6738E: _GLOBAL_OFFSET_TABLE_ is undefined. Object <oepname> section '<secname>' relocation #<relocnum> makes a GOT-relative relocation to symbol <wrtsym>.

Some GNU produced images can refer to the symbol named _GLOBAL_OFFSET_TABLE_. If there are no GOT Slot generating relocations and the linker is unable to pick a suitable address for the GOT base the linker will issue this error message.

L6739E: Version '<vername>' has a dependency to undefined version '<depname>'

L6740W: Symbol '<symname>' versioned '<vername>' defined in '<symverscr>' but not found in any input object.

Copyright © 2006-2008 ARM Limited. All rights reserved.

L6741E: Versioned symbol binding should be 'local:' or 'global:'

L6742E: Symbol '<symname>' defined by '<oepname>'. Cannot not match to default version symbol '<defversym>'

L6743E: Internal consistency check: Relocation from <spname> from <oepname> index <index> to a symbol <symname> that has an alternate def

L6744E: Internal consistency check: Relocation from <spname> from <oepname> index <index> to undefined symbol <symname>

L6745E: Target CPU <cpuname> does not Support ARM, section <secname> from object <objname> contains ARM code

L6746W: RW data compression has been turned off: <reason>

L6747W: Raising target architecture from <oldversion> to <newversion>.

If the linker detects objects that specify ARMv3 (obsolete in RVCT 2.2 and later), it upgrades these to ARMv4 to be usable with ARM libraries.

L6748U: Missing dynamic array, symbol table or string table in file <oepname>.

L6761E: Cannot choose between <name> from objects <objname1> and <objname2>

L6762E: Cannot build '<type>' PLT entries when building a <imgtype>

L6763W: '<optname>' cannot be used when building a shared object or DLL. Switching it off.

L6764E: Cannot create a PLT entry for target architecture 4T that calls Thumb symbol <symname>

L6765W: Shared object entry points must be ARM-state when linking architecture 4T objects.

This may occur when linking with GNU C libraries. The GNU startup code crt1.o does not have any build attributes for the entry point, so the linker cannot determine which execution state (ARM or Thumb) the code will run in. As the GNU C library startup code is ARM code, you can safely ignore this warning, or suppress it with `--diag_suppress 6765`.

L6766W: PLT entries for architecture 4T do not support incremental linking.

L6769E: Object <oepname> section '<secname>' relocation #<relocnum> tries to relocate w.r.t non-existant GOTSLOT for symbol <wrtsym>.

L6770E: The size and content of the dynamic array changed too late to be fixed.

L6771W: Object <oepname> section '<secname>' contains one or more address-type relocations in RO data. Making section RW to be dynamically relocated at run-time.

L6772W: IMPORT <symname> command ignored when building –sysv.

L6773U: DWARF optimisation failure: <text>.

L6774W: The section '<secname>' in '<objname>' has debug frame entries of a bad length.

L6775W: The section '<secname>' in '<objname>' has FDEs which use CIEs which are not in this section.

L6776W: The debug frame section '<secname>' in '<objname>' does not describe an executable section.

Copyright © 2006-2008 ARM Limited. All rights reserved.

L6777W: The debug frame section '<secname>' in '<objname>' has <actual> relocations (expected <expected>)

L6778W: The debug frame section '<secname>' in '<objname>' uses 64-bit DWARF.

L6779E: Target cpu '<name> not recognized.

L6780W: <origvis> visibility removed from symbol '<symname>' through <impexp>

L6781E: Value(<val>) Cannot be represented by partition number <part> for relocation #<rel_number> (<rtype>, wrt symbol <symname>) in <objname>(<secname>)

L6782W: Relocation #<relnum> '<rtype>' in <oepname> may not access data correctly alongside <pltgot_type> PLT entries.

L6783E: Mapping symbol #<symnum> '<msym>' in section #<secnum> '<secname>' from <oepname> defined at the end of, or beyond, the section size (symbol offset=0x<moffset>, section size=0x<secsize>

This indicates that the address for a section points to a location at the end of or outside of the ELF section. This may be caused by an empty inlined data section and indicates there may be a problem with the object file. You can use "--diag_warning 6783" to suppress this error.

L6784E: Symbol #<symnum> '<symname>' in section #<secnum> '<secname>' from <oepname> with value 0x<moffset> has size 0x<size> that extends to outside the section.

The linker produces a downgradeable error (in RVCT 2.2 and earlier) whenever it sees a symbol with a size that extends outside of its containing section. Some earlier versions of RVCT and ADS can produce this error in the C-libraries. This message is only a warning by default in RVCT 2.2sp1 onwards. Use "--diag_warning 6784" to suppress this error in earlier versions.

L6785U: Symbol '<syname>' marked for import from '<libname>' already defined by '<oepname>'

L6786W: Mapping symbol #<symnum> '<msym>' in section #<secnum> '<secname>' from <oepname> defined at unaligned offset=0x<moffset>

L6787U: Region table handler '<handlername>' needed by entry for <regionname> was not be found.

L6788E: Scatter-loading of execution region <er1name> will cause the contents of execution region <er2name> to be corrupted at run-time.

This occurs when scatter-loading takes place and an execution region is put in a position where is overwrites partially or wholly another execution region; be it itself or another one.

E.g. This will work:

```
LOAD_ROM 0x0000 0x4000
{
    EXEC1 0x0000 0x4000
    {
        * (+RO)
    }
    EXEC2 0x4000 0x4000
    {
        * (+RW,+ZI)
    }
}
```

This will generate the error:

```
LOAD_ROM 0x0000 0x4000
```


Copyright © 2006-2008 ARM Limited. All rights reserved.

```
{
    EXEC1 0x4000 0x4000
    {
        * (+RW,+ZI)
    }
    EXEC2 0x0000 0x4000
    {
        * (+RO)
    }
}
Error: L6788E: Scatter-loading of execution region EXEC2 will cause the
contents of execution region EXEC2 to be corrupted at run-time.
```

Refer to the RVCT 3.0 Linker and Utilities Guide, Chapter 5, “Using Scatter-loading Description Files” for more information on scatter-loading.

```
L6789U: Library <library> member <filename> : Endianness mismatch
```

```
L6790E: May not IMPORT weak reference '<symname>' through GOT-generating relocation
#<relnum> in <objname>(<secname>)
```

```
L6791E: Unknown personality routine <pr> at 0x<offset> in section <secname> from
<oepname>.
```

```
L6792E: Descriptor at offset 0x<offset> in section <secname> from object <object> has
unknown type.
```

```
L6793E: Expecting Landing pad reference at offset 0x<offset> in cleanup descriptor in
section <secname> from object <object>.
```

```
L6794E: Expecting Landing pad reference at offset 0x<offset> in catch descriptor in
section <secname> from object <object>.
```

```
L6795E: Expecting RTTI reference at offset 0x<offset> in catch descriptor in section
<secname> from object <object>.
```

```
L6796E: Descriptor at offset 0x<offset> in section <secname> from object <oepname>
overruns end of section.
```

```
L6797E: Data at Offset 0x<offset> in exception table section <secname> from object
<oepname> overruns end of section
```

```
L6798E: Expecting RTTI reference at offset 0x<offset> in Function Specification
descriptor in section <secname> from object <object>.
```

```
L6799E: Expecting Landing Pad reference at offset 0x<offset> in Function
Specification descriptor in section <secname> from object <object>
```

A landing pad is code that cleans up after an exception has been raised. The exception table format was slightly different in RVCT 2.1. If a later linker detects old-format exception tables then it will automatically convert them to the new format. This message should never appear unless there was a fault in the compiler, in which case you should contact your supplier.

```
L6800W: Cannot convert generic model personality routine at 0x<offset> in section
<secname> from object <oepname>.
```

A personality routine is used to unwind the exception handling stack. The exception table format was slightly different in RVCT 2.1. If a later linker detects old-format exception tables then it will automatically convert them to the new format. This message should never appear unless there was a fault in the compiler, in which case you should contact your supplier.

Copyright © 2006-2008 ARM Limited. All rights reserved.

L6801E: <objname>(<secname>) containing <secarmthumb> code cannot use the address of '~IW' <funarmthumb> function <sym>.

The linker can diagnose where a non-interworking (~IW) function has its address taken by code in the other state. This was added in RVCT 2.2. This error is disabled by default, but can be enabled by linking with '--strict'. The error can be downgraded to just a warning with '--diag_warning 6801' and subsequently suppressed completely if required with '--diag_suppress 6801'.

E.g. where code in a.c uses the address of a non-interworking function in t.c:

```
armcc -c a.c
tcc -c t.c
armlink t.o a.o --strict
```

reports:

```
Error: L6801E: a.o(.text) containing ARM code cannot use the address of '~IW'
Thumb function foo.
```

L6802E: Thumb Branch Relocation <idx> in section <secname> from object <objname> refers to non-Thumb symbol <armsym> in section <armsecname> from object <armobjname>.

L6803W: Thumb Branch Relocation <idx> in section <secname> from object <objname> refers to <armsym> which is in different section <armsecname> from object <armobjname>, branch is unlikely to reach target.

L6804W: Handling symbols of type STT_LOPROC as STT_TFUNC, please upgrade your compiler to a more ABI compatible release

L6805E: Branch Relocation <idx> in section <secname> from object <objname> refers to Untyped Absolute <armsym> symbol from object <armobjname>, target state unknown

L6806W: Branch Relocation <idx> in section <secname> from object <objname> to Untyped symbol <othersym> which is in different section <othersecname> from object <otherobjname>, ABI requires external code symbols to be of type STT_FUNC.

L6807E: ARM Branch Relocation <idx> in section <secname> from object <objname> refers to Untyped symbol <othersym> in same section. State change is required.

L6809W: Relocation <i> in section <spname> from object <oepname> is of deprecated type <rtype>, please see ARMELF for ABI compliant alternative.

L6810E: Relocation <i> in section <spname> from object <oepname> is of obsolete type <rtype>

Relocation errors and warnings are most likely to occur if you are linking object files built with previous versions of the ARM tools.

To show relocation errors and warnings use the "`--strict_relocations`" switch. This option enables you to ensure ABI compliance of objects. It is off by default, and deprecated and obsolete relocations are handled silently by the linker.

L6810E: Relocation <idx> in section <spname> from object <oepname> is of obsolete type <rtype>

L6811U: Unknown internal SymbolState, please contact your supplier

L6812U: Unknown symbol action type, please contact your supplier

L6813U: Could not find Symbol <symname> to rename to <newname>

L6898E: ARM Branch Relocation <idx> in section <secname> from object <objname> refers to non-ARM/Thumb symbol <armsym> in section <armsecname> from object <armobjname>.

Copyright © 2006-2008 ARM Limited. All rights reserved.

L6899E: Existing SYMDEFS file '<filename> 'is read-only.

L6900E: Expected parentheses to specify priority between AND and OR operators.

L6901E: Expected symbol name.

L6902E: Expected a string.

L6903E: Cannot execute '<text>' in '<clause>' clause of script.

L6904E: Destination symbol of rename operation clashes with another rename.

L6905E: Source symbol of rename operation clashes with another rename.

L6906E: (This is the rename operation which it clashes with.)

L6907E: Expected an expression.

L6910E: Expected a phase name.

L6912W: Symbol <symname> at index <idx> in symbol table of Section <secname> of object <oepname>, has ABI symbol type <symtype> which is inconsistent with mapping symbol type <maptype>.

L6913E: Expected execution region name.

L6914W: option <spurious> ignored when using --<memoption>.

L6915E: Library reports error: <msg>

1) Error: L6915E: Library reports error: scatter-load file declares no heap or stack regions and __user_initial_stackheap is not defined.

It is most likely that you have not re-implemented __user_initial_stackheap(). Ensure that you have properly defined ARM_LIB_STACK and/or ARM_LIB_HEAP in the respective Scatter file. The RVDS 3.0 \emb_sw_dev directory contains examples of how to re-implement __user_initial_stackheap() - see the file retarget.c.
Please see FAQ "Re-implement __user_initial_stackheap() when using Scatterloading" at:
http://www.arm.com/support/faqdev/1247.html
Please see also Chapter 2 in the RVCT 3.0 Developer Guide - 2.1.5 , and Chapter 5, in the RVCT 3.0 Compiler and Libraries Guide 5.1.1 to 5.1.3:__user_initial_stackheap()

2) Error: L6915E: Library reports error: __use_no_semihosting was requested but <function> was referenced.

Or

Error: L6915E: Library reports error:__use_no_semihosting_swi was requested, but <reason> was referenced

Where <function> represents `__user_initial_stackheap`, `_sys_exit`, `_sys_open`, `_sys_tmpnam`, `_ttywrch`, `system`, `remove`, `rename`, `_sys_command_string`, `time`, or `clock`

This error may appear when retargeting semihosting-using functions, in order to avoid any SWI/SVC/BKPT instructions being linked-in from the C libraries.

To ensure that no semihosting-using functions are linked in from the C library, import __use_no_semihosting:
```
#pragma import(__use_no_semihosting)
```


Copyright © 2006-2008 ARM Limited. All rights reserved.

See the RVCT 3.0 Compilers and Libraries Guide, section 5.3.3, "Building an application for a nonsemihosted environment" and RVCT 3.0 Developer Guide, Section 2.3.2, "Avoiding C library semihosting".

If there are still semihosting-using functions being linked in, the linker will report this error.

To resolve this, you must provide your own implementations of these C library functions.
The RVCT 3.0 \emb_sw_dev directory contains examples of how to re-implement some of the more common semihosting-using functions - see the file `retarget.c`.

RVCT 3.0 Compiler and Libraries Guide, Table 5-3 give a full list of semihosting-using C library functions.
Note: The linker will NOT report any semihosting-using functions (e.g. __semihost()) in your own application code.

To identify which semihosting-using functions are still being linked-in from the C libraries:
1. Link with '`armlink --verbose --errors err.txt`'
2. Search err.txt for occurrences of '__I_use_semihosting'
For example:

```
:
Loading member sys_exit.o from c_a__un.l.
              reference :  __I_use_semihosting
              definition:  _sys_exit
:
```

This shows that the semihosting-using function _sys_exit is being linked-in from the C library. To prevent this, you will need to provide your own implementation of this function.

`3) Error: L6915E: Library reports error:__use_no_heap was requested, but <reason> was referenced`
Where <reason> represents `malloc`, `free`, `__heapstats`, or `__heapvalid`, the use of `__use_no_heap` conflicts with usage of these functions.

`4) Error: L6915E: Library reports error:__use_no_heap_region was requested, but <reason> was referenced`
Where <reason> represents `malloc`, `free`, `__heapstats`, `__heapvalid`, or `__argv_alloc`, the use of `__use_no_heap_region` conflicts with usage of these functions.

`L6916E: R_ARM_CALL relocation for conditional BL at <offset> in Section <spname> from object <oepname>.`

`L6917E: R_ARM_JUMP24 relocation for BLX at 0x<offset> in Section <spname> from object <oepname>.`

`L6918W: Execution region <ername> placed at 0x<eraddr> needs padding to ensure alignment <spalign> of section <spname> from object <oepname>.`

`L6922E: Section <objname>(<secname>) has mutually exclusive attributes (READONLY and TLS)`

`L6923E: TLS Relocation <type> at <idx|%d} in Section <secname> from <objname> targets non STT_TLS symbol <symname> from section <symsecname> from <symobjname>.`

`L6924E: Non-TLS Relocation <type> at <idx> in Section <secname> from <objname> targets STT_TLS symbol <symname> from section <symsecname> from <symobjname>.`

`L6925E: Ignoring <token> attribute for region <region>. MemAccess support has been removed.`

Copyright © 2006-2008 ARM Limited. All rights reserved.

L6926E: Incorrect relocation type <rtype> at offset <offset> instruction encoding 0x<bl> in Section <spname> from object <oepname>.

L6927E: Incorrect relocation type <rtype> at offset 0x<offset> instruction encoding 0x<bl1><bl2> in Section <spname> from object <oepname>.

L6935E: Debug Group contents are not identical, <name> with signature sym <sig> from objects (<new>) and (<old>)

L6938E: Invalid value for --ro-base

L6939E: Missing alignment for region <regname>

L6940E: Alignment {alignment|%d} for region {regname> must be at least 4 and a power of 2 or MAX

L6941W: chmod system call failed for file <filename> error <perr>

L6942E: <n> unused debug section(s) (total <bytes> bytes) removed from the image

L6966E: Alignment <alignment> for region <regname> cannot be negative

L6967E: Entry point <address> points to a THUMB instruction but is not a valid THUMB code pointer.

Copyright © 2006-2008 ARM Limited. All rights reserved.

5. ELF Format Converter (fromelf) Errors and Warnings

`Q0105E: Base and/or size too big for this format, max = 0x<maxval>.`

`Q0106E: Out of Memory.`

`Q0107E: Failed writing output file.`

`Q0108E: Could not create output file '<filename>': <reason>`

`Q0111E: Unrecognised option '<opt>'.`

`Q0112E: Missing output format before '<arg>'.`

`Q0113W: Ignoring unrecognised text information category '<cat>'.`

`Q0114W: Ignoring multiple input file '<filename>'.`

`Q0115W: Deprecated command syntax will not be supported in future versions. Use --output to specify the output file.`

This warning is intended to highlight that the old SDT 2.5x form of the `fromelf` command:

```
fromelf -bin image.elf image.bin
```

has now been changed to:

```
fromelf image.elf --bin -o image.bin
```

`Q0116E: No text information category specified.`

`Q0117E: Unrecognised file format '<arg>'.`

`Q0118E: Missing argument for option '<arg>'.`

`Q0119E: No output file specified.`
`Q0120E: No input file specified.`
`Q0122E: Could not open file '<filename>': <reason>.`

`Q0123E: Failed to read file. Invalid seek offset possible.`

`Q0127E: Cannot translate an ELF Relocatable file (object) into <format> format.`

Only executable files can be translated in this way.

`Q0128E: File i/o failure.`

`Q0129E: Not a 32 bit ELF file.`
`Q0130E: Not a 64 bit ELF file.`

`Q0131E: Invalid ELF identification number found.`

This error is given if you attempt to use fromelf on a file which is not in ELF format, or which is corrupted. In RVCT, object (.o) files and executable (.axf) files are in ELF format.

`Q0132E: Invalid ELF section index found <idx>.`
`Q0133E: Invalid ELF segment index found <idx>.`
`Q0134E: Invalid ELF string table index found <idx>.`
`Q0135E: Invalid ELF section entry size found.`
`Q0136E: ELF Header contains invalid file type.`
`Q0137E: ELF Header contains invalid machine name.`
`Q0138E: ELF Header contains invalid version number.`

See `Q0131E`.

Copyright © 2006-2008 ARM Limited. All rights reserved.

`Q0139E: ELF Image has insufficient information to effect this translation.`

Some fromelf operations require the ELF image to contain debug information. Rebuild your image with '-g'.

`Q0140E: ELF image requires an entry point to effect this translation.`

Some fromelf operations require the ELF image to have an entry point. Rebuild your image with '--entry'. This error can also occur with 3rd-party tools that do not set an ARM-specific flag (e_flags) in the ELF header. This flag is used by ARM tools to distinguish between an ELF image with no entry point, and an ELF image with an entry address of 0.

`Q0141E: Invalid debug offset found. Seek failure.`

`Q0142E: ELF Image does not have a ROOT Region.`

The image entry point must correspond to a valid instruction in the root-region in the image. Images that have been successfully created with the ARM linker will always have this.

`Q0143E: Failed to write High level debug information.`
`Q0144E: Failed to write Low level debug information.`
`Q0145E: Failed to write image string table.`

A file could not be written to - check that you have write access permissions.

`Q0147E: Failed to create Directory <dir>: <reason>.`
`Q0148E: Failed to change to directory <dir>: <reason>.`
`Q0149E: Failed to change to directory <dir>: <reason>.`

`Q0158W: Cannot use filename as argument '<filename>'.`

`Q0159W: Multiple output formats specified. Ignoring <format>.`

`Q0160E: Invalid ELF section offset found '<offset>'.`

See `Q0131E`.

`Q0161E: Section contents do not lie fully within the file '<offset>'.`

`Q0162E: Invalid ELF program segment offset found '<offset>'.`

See `Q0131E`.

`Q0163E: Program segment contents do not lie fully within the file. '<idx>'.`

`Q0164E: Invalid e_shstrndx value (<shstrndx>) found in ELF header (total sections <e_shnum>).`

`Q0165E: Symbol Table Section has not got type of SHT_SYMTAB or SHT_DYNSYM.`

The ELF section '`.symtab`', which contains the symbol table, must have type `SHT_SYMTAB`. If a given ELF file does not have this, this may be due to the ELF file being corrupt. Try re-linking it.

`Q0166E: Relocation Section has not got type of SHT_REL nor SHT_RELA.`

`Q0167E: Error occurred in section <idx>.`
`Q0168E: Error occurred in segment <idx>.`
`Q0170E: Section pointer is null`

`Q0171E: Invalid st_name index into string table <idx>.`

See `Q0131E`.

`Q0172E: Invalid index into symbol table <idx>.`

See `Q0131E`.

Copyright © 2006-2008 ARM Limited. All rights reserved.

Q0173E: Failed to close temporary file ‘<tmpname>’: <reason>

Q0174E: Failed to delete temporary file
Q0175E: Failed to rename temporary file

Q0178U: Internal error: bad section header pointer in section with index <idx>.

Q0179W: Multiple bank types specified. Ignoring <banks>.

Q0180W: Symbol Table entry size is 0.

Q0181W: Relocation entry size is 0.

Q0182E: Failed to open temporary file

Q0183W: <format> format is obsolete and will not be supported in future versions of the toolkit.

Q0184E: Section <name> (<number>) has File Offset <offset> which is not <required_align> byte aligned

Q0185E: Unable to make unique temporary file from <filename>

Q0186E: This option requires dwarf2 debugging information to be present

The `--fieldoffsets` option requires the image to be built with dwarf debug tables.

Q0187E: Cannot produce addresses for Relocatable Elf file

"`fromelf –a`", which prints data addresses, can only be used on executable image files, not object files.

Q0188E: Program segment <number> must be <required_align> aligned in file

Q0189U: Internal error: bad segment header pointer in section with index <idx>.

Q0190E: String Table Section <idx> has not got type of SHT_STRTAB.

The ELF section '`.strtab`', which contains the string table, must have type `SHT_STRTAB`. If a given ELF file does not have this, this may be due to the ELF file being corrupt. Try re-linking it.

Q0191E: Option <old> has changed name and is now deprecated, please use <new> instead.

Q0193E: Could not save output file <filename>, removal of old output file failed: <reason>

Q0194E: Could not save output file <filename>, renaming of temporary file failed: <reason>

Q0195E: Cannot open <filename>, existing directory with same name

Q0419E: No SYMTAB_SHNDX section exists for section <sec_idx>

Q0420E: Out of range symbol idx <sym_idx>

Q0421E: No associated SHT_SYMTAB_SHNDX section for SHT_SYMTAB section <symtab_sec>

Q0422E: Bad error message list <list> for command <command>

Copyright © 2006-2008 ARM Limited. All rights reserved.

`Q0424E: More than one relocation section for <secname>`

More than one relocation section linked to a vfe or exceptions section in fromelf --text output.

`Q0425W: Incorrectly formed virtual function elimination header in file`

`Q0426E: Error reading vtable information from file`

`Q0427E: Error getting string for symbol in a vtable`

This may indicate a compiler fault, please contact your supplier.

`A0447W: Unknown Diagnostic number (<num>)`

`Q0448W: Read past the end of the compressed data while decompressing section '<secname>' #<secnum> in <filename>`
`Q0449W: Write past the end of the uncompressed data buffer of size <bufsize> while decompressing section '<secname>' #<secnum> in <filename>`

Any of the 2 above messages may indicate an internal fault; please contact your supplier.

`Q0450W: Section '<secname>' #<secnum> in file <filename> uses a mixture of legacy and current ABI relocation types.`

Copyright © 2006-2008 ARM Limited. All rights reserved.

6. ARM Librarian (armar) Errors and Warnings

```
L6800U: Out of memory

L6825E: Reading archive '<archive>' : <reason>

L6826E: '<archive>' not in archive format

L6827E: '<archive>': malformed symbol table
L6828E: '<archive>': malformed string table
L6829E: '<archive>': malformed archive (at offset <offset>)

L6830E: Writing archive '<archive>' : <reason>

L6831E: '<member>' not present in archive '<archive>'

L6832E: Archive '<archive>' not found : <reason>
L6833E: File '<filename>' does not exist
L6834E: Cannot open file '<filename>' : <reason>
L6835E: Reading file '<filename>' : <reason>

L6836E: '<filename>' already exists, so will not be extracted

L6837E: Unrecognized option '<option>'

L6838E: No archive specified

L6839E: One of [<actions>] must be specified
L6840E: Only one action option may be specified

L6841E: Position '<position>' not found

L6842E: Filename '<filename>' too long for file system

L6843E: Writing file '<filename>' : <reason>

L6844E: Missing argument to '<option>'

L6845E: Cannot delete '<member>' as  '<member>' is a variant of it
L6846E: Cannot insert variant '<member>' as there is no symbol-compatible base member
L6847E: Cannot insert '<filename>' as it has incompatible build attributes
L6848E: Cannot replace '<member>' as new version and old version are not symbol
compatible, and it has a variant member ('<variant_member>') dependant upon it

L6849E: Unrecognized long option '<option>'

L6850E: Archive ‘<archive>’ contains non ELF Object <name>

L6851E: Bad error message list <list> for command <command>

L6870W: Via file '<filename>' is empty

L6871W: Build attributes of archive members inconsistent with archive name

L6874W: Minor variants of archive member '<member>' include no base variant
```

It is possible to have minor variants of the same function within a library, by compiling each variant with different build options in separate (individually named) object files. If these objects are combined in a library, at link-time the linker will select the most appropriate version of the function according to the callers build attributes. Examples of minor variants are versions compiled for different architectures, ROPI/non-ROPI etc. Major variants must be placed in separate libraries, examples are versions compiled for different instruction sets (ARM/Thumb), endianness etc.

Copyright © 2006-2008 ARM Limited. All rights reserved.

A base variant is a library member that contains all the attributes in common to all the variants armar is warning as it is usually a mistake to define a set of variants without a base variant, as the linker may not be able to find a default acceptable member in the library. For the case of:

```
Warning: L6874W: Minor variants of archive member 'abc.o' include no base
variant
```

'abc.o' (probably unintentionally) contains a function which is also defined in another archived object, which was built with different options. You can view the symbol table of an archive using 'armar --zs' - variant symbols will be appended with their build attributes. For example, if an archive contained an architecture v3 function 'func' and an architecture v4 variant, the symbols table might show:

```
func                                from v3_func.o  at offset   120
func$$BuildAttributes$$ARM_ISAv4    from v4_func.o  at offset  1104
```

Assuming that you intended to have different variants of the function, you would need to add an object containing a base variant in order to fix the warning. Alternatively, you could safely ignore the warning, but at link-time there is a risk that the linker may not be able to find a suitable default member.

`L6875W: Adding non-ELF object <filename> to archive <name>`

`A6972W: Unknown Diagnostic number (<num>)`

`L6973E: Reading member '<member>' : <reason>`

Copyright © 2006-2008 ARM Limited. All rights reserved.

7. ARM Via file handling

These error messages can be produced by any of the tools. The x prefixing the message number within this documentation will be replaced with the appropriate letter relating to that application when it is displayed.

`x0000U: Unrecognized option '<dashes><option>'.`
: <option> is not recognized by the tool. This could be due to a spelling error, or due to the use of an unsupported abbreviation of an option.

`x0001U: Missing argument for option '<option>'.`

`x0002U: Recursive via file inclusion depth of <limit> reached in file '<file>'.`

`x0003U: Argument '<argument>' not permitted for option '<option>’.`
: Possible reasons include malformed integers or unknown arguments.

`x0004U: Could not open via file '<file>'`

`x0005U: Error when reading from via file.`

`x0006U: Malformed via file.`

`x0007U: Via file command too long for buffer`

`x0008U: Overflow: '<string>' will not fit in an integer.`

`x0010W: Old syntax, please use '<hyphens><option><separator><parameter>’`

`x0012W: Option '<option>' is deprecated`

`x0013W: Could not close via file`

`x3900U: Unrecognized option '<dashes><option>'`

`x3901U: Missing argument for option '<option>'`

`x3902U: Recursive via file inclusion depth of <limit> reached in file '<file>'`

`x3903U: Argument '<argument>' not permitted for option '<option>'`

`x3904U: Could not open via file '<file>'`

`x3905U: Error when reading from via file`

`x3906U: Malformed via file`

`x3907U: Via file command too long for buffer`

`x3908U: Overflow: '<string>' will not fit in an integer`

`x3910W: Old syntax, please use '<hyphens><option><separator><parameter>'.`

`x3912W: Option '<option>' is deprecated.`

`x3913W: Could not close via file.`

`x3915W: Argument '<argument>' to option '<option>' is deprecated`

`x3916U: Unexpected argument for option '<dashes><option>'`

`x3917U: Concatenated options cannot have arguments: -<option> <arg>'`

Copyright © 2006-2008 ARM Limited. All rights reserved.