

ADS 1.2 Build Tools - Errors and Warnings

Release 1.2 - 2nd September 2003

Introduction

This document illustrates the errors and warning messages that are generated by the Build Tools of ARM Developer Suite (ADS) version 1.2 and the RealView Compilation Tools (RVCT) version 1.2. Many of these also apply to earlier versions of ADS (ADS 1.1 and ADS 1.0.1). If you have a later version of the RVCT tools please refer to the RVCT Errors and Warnings document instead.

This document is divided into the following sections:

1. CodeWarrior
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)

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 ADS 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 installed with ADS. For information on this, please see the ADS License Management FAQ at <http://www.arm.com/support/licensemanagement>

This document is intended to complement, not replace, the ADS 1.2 documentation. It should be read in conjunction with the ADS 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 ADS FAQ at http://www.arm.com/support/ads_faq 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>

1. CodeWarrior

The CodeWarrior IDE supplied with ADS 1.2 does not often report error messages itself. Typically an error reported by CodeWarrior is caused not by CodeWarrior itself, but by an unexpected return condition by the core tools (assembler, compiler, linker, etc), and hence CodeWarrior is unable to correctly process the task.

Windows errors, such as:

```
Unhandled Exception: c0000005
At address xxxxxxxx
```

are typical of such an event. If this occurs, please try performing the same build step using the core tools from the command-line instead, to see whether the same error occurs. Please ensure that the same build options are used as would be used with CodeWarrior. If the same error does occur when building from the command-line, please refer to the appropriate compiler, assembler, linker, etc., section in this document for details of that error.

If the error does not occur when building from the command-line, but only occurs when building with CodeWarrior, then CodeWarrior's configuration may be corrupted. You can "reset" ADS 1.2 CodeWarrior back to its default configuration by simply deleting (or renaming) the file "CodeWarrior IDE 4.2 Prefs" from the Metrowerks subdirectory in your Windows profiles directory. If you do this, no important data will be lost, just the list of most recently opened projects, etc. For ADS 1.1, the file was called "CodeWarrior Pro 5 IDE Prefs".

If the error occurs within a single project, then it is likely that the .mcp project file may be corrupted. This is a very rare occurrence. There is no easy way to fix this, so the best course of action is to create a new project file.

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, (-O1, -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 **ADS 1.2 Compiler and Libraries Guide Sections 2.3.10 and 2.3.12**. 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 "-fx" switch can be used to enable all those warnings that are suppressed by default (with the exception of the other -f checks - see section 2.3.11, "Specifying additional checks").

List of Errors and Warnings Messages

C2003E: SWI number 0x12345678 too large

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

C2004E: tcc cannot handle __irq functions

Modules containing __irq interrupt handler C functions can only be compiled with the ARM compilers, not the Thumb compilers, because it is architecturally defined that exception entry and exit must be executed in ARM state.

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.

C2007U: switch statement too large for Thumb compiler

The switch statement is too large. Try splitting it up into smaller functions, or compiling it for ARM instead of Thumb.

C2012U: Too many symbols in object file

In ADS 1.1, the compiler limit for the number of symbols in an object is 65535. In ADS 1.2, this limit is raised to 2^{24} .

C2015E: illegal constant (ignored): <unknown>

C2016E: illegal constant expression (ignored): non constant

C2017E: illegal constant expression (ignored)

C2018E: illegal shift specifier

C2019E: shift specifier expected

Illegal inline assembly instructions – consult the ARM ARM for the allowed forms.

C2020E: illegal instruction opcode

Example:

Error: (Serious) C2020E: illegal instruction opcode: STMFD

occurs due to an instruction like:

```
STMFD SP!, {R0, R2, R3}
```

You must never try to change the Stack Pointer (sp=r13) using the inline assembler, because the compiler requires full control of the stack for its operation, so writing to the SP is not allowed. The compiler will stack/restore any working registers as required automatically.

These, and other similar errors, are typically generated if you compile code with the Thumb compiler tcc, containing inline assembler `__asm` code, which was previously used with the ARM Compiler. The Thumb Instruction Set does not feature all ARM instructions, and so code containing these instructions will not compile with tcc. The use of the Thumb Inline Assembler is now deprecated in ADS 1.2, and will be removed in a future release.

C2021E: physical register name expected, but found foo instead

C2022E: illegal PSR field

C2023E: PSR field expected

Illegal inline assembly instructions – consult the ARM ARM for the allowed forms.

C2024E: shift value <Bits> out of range

Shift distances must be in the range 0 to 32 inclusive for LSR and ASR, 0 to 31 inclusive for all other shift operations.

C2025E: coprocessor identifier expected

This error is given by the inline assembler if the coprocessor number is accidentally omitted from an MCR or MRC instruction, e.g:

```
MRC      0, r0, c1, c0, 0
```

instead of:

```
MRC      p15, 0, r0, c1, c0, 0
```

C2026E: illegal coprocessor identifier

This error is given by the inline assembler if an invalid coprocessor number is specified in an MCR or MRC instruction, e.g:

```
MRC    p16, 0, r0, c1, c0, 0
```

instead of:

```
MRC    p15, 0, r0, c1, c0, 0
```

C2027E: preindexed addressing not available for LDRT/STRT

Illegal inline assembly instruction – consult the ARM ARM for the allowed forms.

C2028E: TEQP/TSTP/CMPP/CMNP are no longer supported

These obsolete 26-bit architecture instructions are no longer supported.

C2029E: long multiply is not available on this architecture

Long multiply instructions are only available on architecture 4 or later.

C2030E: halfword support is not available on this architecture

Halfword instructions are only available on architecture 4 or later.

C2031E: count leading zeros is not available on this architecture

CLZ instructions are only available on architecture 5T or later.

C2032E: dsp multiply is not available on this architecture

DSP multiply instructions are only available on architecture 5TE or later.

C2033E: R14 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.

C2034E: LDM/STM base with writeback in register list

C2035E: LDM/STM with empty register list

C2036E: LDM/STM with illegal register list

C2037E: LDR/STR base, [base] with writeback

C2038E: MUL/MLA with Rd = Rm

C2039E: MULL/MLAL with Rhi = Rlo, Rhi = Rm or Rlo = Rm

C2040E: SWP with Rd = Rn or Rm = Rn

C2041E: Thumb ADC/SBC cannot be used as 3-operand instructions

C2042E: illegal use of PC as operand

C2043E: illegal write to PC

C2044E: branching by writing PC is not supported

Illegal inline assembly instruction – consult the ARM ARM for the allowed forms.

C2045E: function identifier expected

C2046E: function <func> undefined

C2047E: illegal access to high register Rn

Illegal inline assembly instruction – consult the ARM ARM for the allowed forms.

C2048E: expression of type <int> expected

C2049E: expression of type <float> expected

- C2050E:** constant expression XXX out of range (max YYY)
- C2051E:** offset XXX out of range
- C2052E:** unaligned offset not allowed
- C2056W:** illegal unaligned load or store access - use `__packed` instead
- C2057E:** `-fpu xxx` is incompatible with selected `-cpu` option
 Example: `armcc -cpu arm10200 -fpu fpa`
 will fail because the arm10200 contains a vfp, not fpa.
- C2059E:** `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
- C2060E:** 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
- 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 ADS compiler documentation for more information about `"-zas"` and `"-wy"`.
- C2068E:** PSR corrupted but possibly reused later. This code may not work correctly
 See **C2033E**
- C2070W:** Memory access attributes below the minimum requirement for ARM/Thumb
 An invalid `-memaccess` option has been specified
- C2072E:** preload and double word accesses are not available on this architecture
 PLD and double word instructions are only available on architecture 5TE or later.
- C2073E:** double word coprocessor transfers are not available on this architecture
 Double word coprocessor instructions are only available on architecture 5TE or later.
- C2075W:** 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.
- C2076E:** illegal write to SP
 Example:
`__asm{ mov sp, #0x4000; }`
- You must never try to change the Stack Pointer (`sp=r13`) using the inline assembler, because the compiler requires full control of the stack for its operation, therefore writing to the SP is not allowed. The compiler will stack/restore any working registers as required automatically.
- C2077E:** The preload instruction can not have a writeback
 Illegal PLD inline assembly instruction – consult the ARM ARM for the allowed forms.
- C2200W:** '`<Number>`' treated as '`<Number>ul`' in 32-bit implementation

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

C2201W: '<Number>' treated as '<Number>ll'

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 the warning:

C2201W: '2147483648' treated as '2147483648ll'

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 3.2.2 of the ADS 1.2 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);
}
```

This warning can be suppressed with "-W0".

C2202W: '<Number>' treated as '<Number>ull'

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

C2203W: non-portable - not 1 char in '...'

This warns about multiple-character char constants. For example:

```
int a = 'abcd';
```

-Wm suppresses this warning

C2204W: C++ keyword used as identifier:

This warns about future compatibility with C++. This warning is suppressed by default, but can be enabled by -W+u. Example:

```
int *new(void *p) { return p; }
```

because "new" is a keyword in C++.

C2205W: Functionality of C++ keyword is not fully implemented:

For example:

Functionality of C++ keyword is not fully implemented: 'reinterpret_cast'

Functionality of C++ keyword is not fully implemented: 'using'

Please refer to the ADS 1.2 Compilers and Libraries Guide, Appendix C, "Standard C++ Implementation Definition" for an overview of the C++ support.

The reason for the message in certain cases is that the wrong cast may be done.

Example:

```
struct A { /* ... */ };
struct B { /* ... */ };
struct D : A, B { };
B* f(D* d) { return reinterpret_cast<B*>(d); }
// does a static_cast instead of a reinterpret_cast
```

We also do not enforce some restrictions on `reinterpret_cast`. The standard says `reinterpret_cast` is not allowed to cast away `const`, but this is not checked.

C2206W: Undefined macro 'FOO' in `#if` - treated as 0
This warning usually occurs because of code like:

```
#if FOO
\\do something
#endif
```

but `FOO` is not defined.

There is no compiler switch to suppress this warning.

This can be easily fixed by either `#include` a header file containing the definition of `FOO`, or use the compiler command-line switch `-D`, e.g.
`armcc -c yourfile.c -DFOO=0`

C2207W: inventing 'extern int func();'
This is a common error 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");
}
```

gives:

C2207W: inventing 'extern int printf();'

For ANSI C, this warning can be suppressed with `-wf` - useful when compiling old-style C in ANSI C mode. For C++, this is an error which cannot be suppressed.

C2209W: spurious `{}` around scalar initialiser

C2210W: Dangling 'else' indicates possible error
Check that the 'else' matches an 'if' correctly.

C2211W: non-value return in non-void function

C2213W: formal parameter `<arg>` not declared - 'int' assumed

For example:

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

C2214W: Old-style 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;
}

```

C2215W: Deprecated declaration `func()` - 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 `"-wd"` option when porting old code. In C++, `void foo();` means `void foo(void);` and no warning is generated.

C2218W: implicit 'int' return type for `<Function>` - 'void' intended?

`<Function>` has been declared or defined with no return type. 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. For ANSI C, the `"-Wv"` option suppresses this warning. For C++, this always results in an error.

C2219W: `<class>` has no named member

C2220W: Superfluous `','` in 'enum' declaration

There is a comma following the last member in an `enum` declaration. This is illegal in ANSI C.

C2221W: padding inserted in 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 `"-W+s"`.

C2223W: access declaration with no effect

C2224W: storage-class without declarator is spurious

C2225W: declaration lacks type/storage-class (assuming 'int')

For C++ only, the `-Ei` option downgrades from error to warning the use of implicit `int` in constructs such as `const i;`.

C2226W: archaic C-style function parameter

For example:

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

C2227W: `'='`, `','` or unary `'&'` defined as non-member

C2229W: inserting `...` in `':(...)'` anachronism

C2231E: `<class>` has no default constructor/destructor

C2232W: `<name>` ignored for template

C2233E: Function try block not implemented yet, handler(s) ignored

C2234W: `'__packed'` ignored

For example:

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

`__packed` is ignored here because a void parameter cannot be `__packed`.

C2235W: `'__packed'` ignored for `x`

For example:

```
void foo( void )
{
    __packed int x; // local variable
}
```

`__packed` is ignored here because a local variable cannot be `__packed`.

C2236W: '`__packed`' ignored in return type

For example:

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

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

C2248W: delete of pointer to undefined `<class>`; no destructor will be called

C2249W: base class `<class>` is implicitly private

C2250E: exceptions are not supported, they will be ignored

C++ exceptions are not currently supported in ADS.

C2252W: base `<class>` will be initialized in declaration order instead of the order written

C2253W: member `<name>` will be initialized in declaration order instead of the order written

This can be avoided by re-writing the order of the member initializers.

This warning can be suppressed with `-Wq`.

C2254E: no arguments allowed for pseudo-destructor

C2255E: cannot yet handle this sort of template definition

C2256E: in pseudo destructor call, types `<type1>` and `<type2>` are not equal

C2257E: type of `<name1>` not found, assuming same type as `<name2>`

C2258E: no arguments allowed for an implicit-destructor

C2260W: omitting trailing `'\0'` for `char [5]`

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").

C2262W: Attempt to initialise non-aggregate

C2263E: Number xxx too large for 32-bit implementation

C2264E: Number xxx too large for 64-bit implementation

C2265E: Grossly over-long floating point number

C2266E: Digit required after exponent marker

C2267E: Grossly over-long hexadecimal constant

C2268E: Grossly over-long number

C2269E: Hex digit needed after `0x` or `0X`

C2270E: Missing hex digit(s) after `\x`

C2271E: `\<space>` and `\<tab>` are invalid string escapes

C2272E: Newline or end of file within string

Probably a closing string quote has been omitted. If you want to include a newline in a string, use the escape sequence.

C2273E: misplaced preprocessor character

A '#' character is in an incorrect position

C2274E: illegal character (0xXX = '<char>') in source

C2275E: illegal character (hex code 0xXX) in source

Example:

Error: (Serious) C2275E: illegal character (hex code 0x1a) in source foo.c line 200.

You have a non-standard carriage return at the end of your file that evaluates to 0x1a. ASCII character 0x1a is not part of the standard 'alphabet', and so the C compiler cannot accept it as input, hence the error. Simply delete the last line of the source file and apply a correct carriage-return to terminate the file and this error should disappear.

C2276E: (...) must have exactly 3 dots

The ellipses to denote variadic functions, e.g. printf(), must have 3 dots.

C2278E: illegal destructor

C2279E: bit size <Bitsize> illegal - 1 assumed

<Bitsize> is larger than the maximum permitted size for a bitfield.

C2280E: zero width named bit field - 1 assumed

Padding bitfields (declared with size 0) must be anonymous.

C2281E: Array size xxx illegal - 1 assumed

There is a limit of 256MB (8MB in ADS 1.1 and earlier) on the maximum size of arrays or structures, due to internal compiler implementation limits. If you need to define arrays larger than this, then you can use malloc() to create some storage space instead. Alternatively, you can define the array as a ZI area in an assembler file using SPACE, for example:

```
        AREA mymem, DATA, NOINIT, ALIGN=2
arr     SPACE 4*2111000
        END
```

and then define the array in a C header file:

```
extern int arr[];
```

The SPACE directive reserves a zeroed block of memory. % is a synonym for SPACE.

C2282E: expected ' ' - inserted before ' '

C2283E: expected ' ' - inserted before ' '

C2284E: expected ' ' after command - inserted before ' '

C2285E: expected ' ' or ' ' - inserted ' ' before ' ')

C2286E: expected ' ')

C2287E: expected ' ')

C2288E: expected ' ' after command

C2289E: expected ' ' or ' ')

These can occur during code development, where there is e.g. a missing semicolon or comma.

C2285E: expected ';' or ',' - inserted ';' before ' ')

C2290E: char and wide (L"...") strings do not concatenate

C2292E: typedef name <name> used in expression context

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

```
typedef int footype;
int x = footype;    // reports "typedef name 'footype' used in expression
context"
```

To fix this, you must 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;
```

C2295E: EOF not newline after #if ...

#if must be followed by a newline, not an End Of File

C2296E: Junk after #if <expression>

For example:

```
#if EMBEDDED foo
```

C2297E: too many initialisers in {} for aggregate

C2298E: {} must have 1 element to initialise scalar

C2299E: Initialiser list must contain at least one expression

C2300E: 'default' not in switch - ignored

C2301E: duplicate 'default' case ignored

C2302E: 'case' not in switch - ignored

C2303E: duplicated case constant:

The switch-case-default statements are - malformed correct your code.

C2291E: <expression> expected but found '...'

C2294E: Expected <member> but found '...'

C2304E: <command> expected but found '...'

C2305E: <statement> expected but found '...'

C2306E: 'while' expected after 'do' but found '...'

C2311E: '{' of function body expected - found '...'

C2307E: Misplaced 'else' ignored

C2308E: 'continue' not in loop - ignored

C2309E: 'break' not in loop or switch - ignored

C2310E: 'goto' not followed by label - ignored

C2312E: storage class <stgclass> not permitted in context <context> - ignored

C2313E: storage class <stgclass> incompatible with <type> - ignored

C2314E: storage class <stgclass> not permitted for an anonymous union that is a member - ignored

C2315E: storage class <stgclass> not permitted for an anonymous union that is a member - ignored

C2316E: storage class repeated

C2317E: only constructors can be ...

C2318E: only data members can be ...

C2319E: type <Type1> inconsistent with <Type2>

The type name or type qualifier <Type1> cannot be used in the same <declaration specifier> as the type name or type qualifier <Type2>. If <Type1> is the same as <Type2>, this means that the type name or type qualifier may not be repeated. For example:

```
typedef int int;
```

C2320E: type <type> inconsistent with ...

C2321E: '{' or <identifier> expected after xxx but found '...'

C2322E: Expecting <declarator> or <type> but found '...'

This can occur for a number of different reasons, for example:

1) **C2322E:** Expecting <declarator> or <type> but found '"C"'

is reported when attempting to compile some C++ header files (e.g. stdcomp.h) with the C compiler instead of the C++ compiler.

2) after an earlier error e.g. missing semicolon, or bracket.

C2323E: superfluous <id> in <abstract declarator> - ignored

C2324E: undefined <struct/class> member

C2325E: undefined <struct/class> object

C2326E: attempt to include <struct/class> member within itself

C2327E: attempt to include <struct/class> object within itself

C2328E: incomplete type

C2329E: illegal 'void' type

C2330E: illegal 'void' member

C2331E: illegal 'void' object

This error may be produced, when the compiler is in strict 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.

When the compiler is not in strict ANSI C mode, this is not even given a warning.

C2332E: duplicate type specification of formal parameter

C2333E: Non-formal <name> in parameter-type-specifier

C2334E: names may not be initialised

For example:

```
__global_reg(1) int x = 1;
```

reports:

C2334E: '__global_reg' names may not be initialised

C2335E: <identifier> expected but found '...' in 'enum' definition

C2336E: Undefined enum

Unlike structs, enum tags may not be declared without also declaring the members of the enumeration.

C2337E: Misplaced '{' at top level - ignoring block

C2338E: not a friend class

C2339E: undefined base

C2340E: missing base tag

C2341E: Incompatible arguments to <name> (overload missing?)

C2343E: ambiguous N-way overload for call

C2344E: ambiguous N-way overload for operator

C2345E: Missing definition for <name>

C2346E: ambiguous N-way choice of conversion from <class> to <type>

C2348E: ambiguous N-way user-defined conversion from <type1> to <type2>

C2349E: illegal `asm(...)` declaration (ignored)

C2350E: recursive application of operator `->()` to object of type <class>; return type intended?

C2351E: no suitable operator=() for <class>: overload missing?

C2354W: cast from ptr/ref to <name1> to ptr/ref to <name2>; one is undefined, assuming unrelated

C2355E: can't yet cast ptr-to-mem needing this offset

C2356E: requires placement new

C2357W: handler is unreachable

C2358W: pure virtual function <pvfn> called
 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 a semihosting SWI. See ADS 1.2 Compilers and Libraries Guide, Table 4-10, "Signals used by the C and C++ libraries".

C2359W: name mangling truncated

C2360E: wrong use of operator&, <name> possible non-static member function

C2361E: constructor forbids <name> = {...} initialiser

C2362E: <class> has no <member> member

C2363E: member <member> not found in <class>

C2364E: Missing class member function name

C2365E: class-name <class> not found

C2366E: member cannot be initialised
 A certain member such as a floating-point number must be initialised in the constructor and not in the class definition.

C2367E: default values are not allowed in this context

C2368E: <class> lacks base class for ':(...)' anachronism

C2369E: <name> is not a member of struct/class

C2370E: misplaced 'catch' ignored

C2371E: illegal <simple type>

C2372E: class member <name> cannot be defined here

C2373E: 'friend <type>;' needs elaborated-type-specifier

C2374E: expected <linkage-spec> '}' before <name>

C2375E: 'operator <op>' is illegal

C2376E: wrong number of arguments to overload <name>

C2377E: <name> cannot follow unary '::'

C2378E: <name> cannot follow binary '::'
C2379E: expected destructor name after ::~, found <name>

C2380E: missing top-level declaration

C2381E: no <foo> declaration at this type

C2382E: <type> lacks an N-argument constructor

C2383E: no nullary constructor

C2384E: requires pointer argument

C2385E: duplicate member initialisation

C2386E: member initialiser not in constructor

C2387E: expecting <identifier> in <member-initialiser>

C2388E: omitted 'catch' after 'try'

C2389E: legal only in member function

C2390E: 'template' not class nor function

C2392E: attempt to use <class> without parameters

C2393E: inline assembler not available in strict ANSI mode

This is given when the user is trying to use `__asm {}` with `-strict`.

C2394E: too large to be returned in registers : `__value_in_regs` ignored

C2396E: cannot have member of type <type>

C2397E: cannot have member of type unpacked

The ADS 1.2 Compiler Guide, section 3.1.3, 'Type qualifiers', says:

"All substructures of a packed structure must be declared using `__packed`."

This rule applies for all releases of ADS and the earlier SDT 2.5x.

All ADS compilers will fault a non-packed child structure contained in a packed parent structure.

A new test was added into the ADS 1.2 compilers to catch 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:

C2397E: cannot have member 'child' of type unpacked 'struct ChildStruct' in '`__packed`' 'struct ParentStruct'

However, there is a known problem in ADS 1.2 where typedef's can lose the `__packed` qualifier for types, resulting in this error being incorrectly reported. This is fixed in the ADS 1.2 patch (build 826 and later), which can be download from: <http://www.arm.com/support/downloads>

C2398E: can't convert between pointer to member and pointer to non-member

Assigning a pointer to a member to pointer-to-member-type variable requires a very precise syntax. Firstly, the member must be qualified, secondly the parentheses are not allowed around the member-name: See C++ Standard Chapter 5.3.1 section 2.

There is no way in C++ to convert a pointer-to-member to anything else. The reason is that, in general, pointers-to-members are wider than four bytes.

C2399E: no conversion to type <type> in <name>

C2400E: abstract member

C2401E: abstract object

C2402E: abstract type

C2403E: type deduction: overloaded function type

C2404E: floating point type not allowed

C2405E: attempt to take address of template

C2406E: type deduction fails: <type1> disagrees with <type2>

C2407W: type deduction fails: array size differs

C2408E: type deduction fails: function type N-way resolvable

C2410E: type deduction fails: un-recognizable type

C2411E: type deduction fails: free template type arg

C2412E: type deduction fails: free template non-type arg

C2413E: type deduction failed: typename <type> not found

C2414E: Ambiguous class/function templates

C2415E: extern linkage expected for template argument

C2416E: illegal non-type template arg

C2417E: template type arg <type> must have extern linkage

C2418E: call to <name> not dependent on template args to <class>

C2419E: template type arg expected for 'Allocator'

For example:

```
#include <fstream>
#include <vector>

using namespace std;

int main()
{
    vector<int> v(10);
    cout << "test" << endl;
}
```

The ADS compilers do not support the use of namespaces.

The 'template type arg' error is due to the way the standard template library is implemented. The compiler does not know what sort of allocator to use. To overcome this problem, don't use a namespace, but specify the allocator explicitly when the vector is declared, for example:

```
#include <fstream>
#include <vector>

//using namespace std;

int main()
{
    vector<int,allocator<int> > v(10);
```

```

    cout << "test" << endl;
    return 0;
}

```

C2420E: template type <type> requires a <foo> but given <type>

C2421E: template type expected, found ...

C2422E: template type inconsistent with <type>

Template types clash.

C2423E: temporary required for template argument

C2424E: handler for '...' must come last

C2425E: number of template formals for <name> differs from a previous declaration

C2426E: number of template formals for <name> differs from a previous declaration

C2427E: template formal type <type1> inconsistent with <type2>

C2429E: structure or array too large

The maximum permitted size of a structure or array is 256 MB.

C2430E: string initialiser longer than S[N]

The string used to initialise a fixed size character array may not be longer than the size of the array,

e.g:

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

The name array can hold up to 5 characters. "Hello There" is too long to fit.

C2431E: non-constant initialiser (contains non-static address <addr>)

C2432E: non-constant initialiser

C2433E: objects of type 'void' can not be initialised

C2434E: must be defined for (static) variable declaration

C2435E: uninitialised static [] arrays illegal

C2436E: invalid global register number; 1 to n allowed

An invalid register is being used in "__global_reg". For example, __global_reg(4) is disallowed by the Thumb compilers, because r7 is used by the Thumb compilers as a temporary register, so r7 must be kept free for use, so cannot be reserved by __global_reg to hold a variable. Note that the use of "__global_reg" is NOT generally recommended, because this places severe constraints on the compiler's register allocator, possibly resulting in lower-performance code.

C2437E: no global floating point registers allowed

C2438E: no global int registers allowed

C2439E: invalid type for global int register

C2440E: invalid type for global floating point register

C2443E: compiler confused: static const with dynamic initialisation

C2444E: unimplemented: local static with destructor

C2445E: digit 8 or 9 found in octal number

C2446E: number illegally followed by letter

C2447E: hex number cannot have exponent

C2448E: overlarge escape '\xNNN' treated as '\xNN'

C2449E: overlarge escape '\XX' treated as '\YY'

For example:

```
char foo[] = {"\xB BBB" };
```

gives:

C2448E: overlarge escape '\xbbbb' treated as '\xbb'

C2450E: illegal string escape '\XX' - treated as X

This error is commonly associated with the attempted use of non-ASCII character sets, such as 16-bit Unicode characters. ADS 1.2 features 16-bit wchar_t support, but does not support Unicode directly. 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
```

C2451E: L'...' needs exactly 1 wide character

C2452E: no chars in character constant ''

For example:

```
char a = '';
```

C2453E: more than 4 chars in character constant

For example:

```
char a='abcd';
```

C2454E: Return type may not be a void type other than 'void'

The ANSI C standard requires that a function returning no result be declared with a return type of <void>, not some other void type. This is only diagnosed in the compilers' strict ANSI C mode - otherwise, it is not even warned of.

C2455E: array [0] found

Zero-sized arrays are not allowed by default. For example:

```
char name[0] = "Hello";
```

-Ez option suppresses the errors caused by zero-length arrays.

C2456E: undeclared name, inventing 'extern int i'

Example:

```
int foo(void)
{
    int a = 4;
    a = i;
}
```

results in the error: "undeclared name, inventing 'extern int i'", because i has not been declared.

C2457E: undeclared name, inventing 'extern "C" int func(...);'

C2458E: parentheses (...) inserted around expression

C2459E: return <expr> illegal for void function

C2460E: return <expr> illegal for constructor

C2461E: return <expr> illegal for destructor

C2462E: type qualifier <type> re-specified in declaration using typedef

C2463E: Function illegally qualified with const or volatile

Only C++ non-static member functions can be qualified.

C2464E: Pointers to illegal const or volatile functions

Pointers to const or volatile functions can only be C++ pointers to non-static member functions

C2465E: members can not be ...

C2466E: missing type specification - 'int' assumed

C2467E: missing type specification for <name> - 'int' assumed

C2468E: ANSI C does not support 'long float'

C2469E: omitted <type> before formal declarator - 'int' assumed

C2470E: function prototype formal <name> needs type or class - 'int' assumed

For example, change:

```
int foo( bar );
```

to:

```
int foo( int bar );
```

C2471E: ellipsis (...) cannot be only parameter

The ellipses to denote variadic functions, e.g. printf(), must follow at least one parameter, e.g

change:

```
int foo( ... );
```

to:

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

C2472E: prototype and old-style parameters mixed

C2473E: illegal [] member

Example:

```
typedef struct {  
    unsigned char size;  
    char string[];  
} FOO;
```

By declaring an 'open' sized array in the structure, the compiler will not be able to allocate a size of the structure.

C2474E: illegal type (void &) treated as (int &)

C2475E: <type> of reference illegal -- '&' ignored

C2476E: function returning <type> illegal -- assuming pointer

C2477E: abstract <class> is illegal for function return type

C2478E: array of <type> illegal -- assuming pointer

C2479E: type may not be function -- assuming pointer

C2480E: member <name> may not be function -- assuming pointer

C2481E: object <name> may not be function -- assuming pointer

C2482E: function <name> may not be initialised - assuming function pointer

C2483E: Ancient form of initialisation, use '='

C2484E: illegal bit field type <type> - 'int' assumed

Bit fields must have integral type.

C2485W: ANSI C forbids bit field type

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

This warning message can be suppressed with the -Wb option.

C2486E: formal name missing in function definition

In a function definition, all parameters must be given names.

C2487E: declaration with no effect

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. For C++ with `-strict`, the warning is upgraded to an error.

C2488E: duplicate member

C2489E: duplicate member <Func>

C2490E: ',' (not ';') separates formal parameters

C2491E: <class> has no members

C2492E: <name> is not a base member of <class>

C2493E: access declarations only in public and protected parts

C2494E: base access rights cannot be altered

C2495E: illegal conversion <Func>

C2496E: <Func> must have zero parameters

C2497E: <Func> must be a non-static member function

C2498E: no return type allowed for <Func>

C2499E: <Func> must be a function

C2500E: Invalid declarator in operator context

C2501E: 'new' <array> initialiser ignored

C2502E: empty 'new' placement argument list

C2503E: jump past initialisation for 'y'

Example for C++:

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

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;
```

C2504E: <name> is ambiguously qualified

C2505E: Ambiguous definitions for destructor

C2509E: inserting { } around command

C2510E: Definition of <name> not <qualifier>: qualifier ignored

C2511E: ignored for non-function

C2512E: ignored for non-function

C2513E: global anonymous union must be static

C2515E: unknown linkage: extern

C2516E: illegal union member

C2517E: illegal non-public anonymous union member

C2518E: 'e' ignored in 'delete [e]' anachronism

C2519E: self-copying constructor

C2520E: duplicate base <class> ignored

C2523E: private overloaded base member ignored

C2524E: <name> has auto storage

C2525E: cannot be declared here

C2526E: cannot be declared here

C2527E: friend <class> shall not be defined

C2528E: expecting destructor for <name>

C2529E: unions may not have bases

C2530E: too few/many arguments to initialiser for simple member

C2531E: reference <foo> must be initialised

C2532E: references cannot be default initialised

C2533E: type cannot be a reference

C2534E: function types cannot be <type>

C2535E: constant must be initialised

C2536E: constant must be initialised

C2537E: can't take address of constructor or destructor

C2538E: ambiguous N-way choice of conversion from <name> in Boolean context

C2539E: cannot define within formals

C2540E: cannot have both negative and unsigned enumerators

C2541E: explicit instantiation can't be mixed with template declaration or specialization

Example:

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

C2542E: multiple explicit instantiation

Example:

```
template template void f(); // is illegal
```

C2543E: extra template actual(s) ignored

C2544E: typedef name <name> not allowed after <foo>

C2545E: templates are only allowed in namespace and class scopes

C2546E: <name> has been declared as <type> but not defined as <qualifier>

A qualifier is missing.

C2547E: illegal in #if <expression>: <unknown>

C2548E: illegal in #if <expression>: non constant

C2549E: illegal in #if <expression>:

C2550E: illegal in case expression (ignored): <unknown>

C2551E: illegal in case expression (ignored): non constant

C2552E: illegal in case expression (ignored):

C2559E: has multiple conversion to pointer

C2560W: identifier too long (over <CurrentMax> characters) -- truncated to <NewIdentifier>

The identifier exceeded the <CurrentMax> length for an identifier and was truncated to <NewIdentifier>

C2561E: repeated qualifier <TypeQualifier> ignored

The type qualifier <TypeQualifier> may not be repeated.

C2562E: can't balance types in conditional expression: <Type> <Type>

The two results of a conditional expression have differing types.

C2563E: Trying the declare a destructor outside a scope

C2564W: 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:

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

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

C2565E: Pointer to member points to an illegal type

C2566E: Bitfield <name> can not be a static member

C2567E: The <foo> can not be declared with this number of arguments

C2568E: __value_in_regs used with non-structure type

C2587E: non-POD object cannot be passed as '...' argument

Objects of class type with copy constructors or destructors cannot be passed as '...' arguments

C2588E: expected '_' after '0%c'

C2589E: floating-point hex literal expects exactly xxx digits

C2590E: <name> can not be explicitly instantiated because template <t> is not defined

C2591E: Pointer to member <member> must be qualified
and

C2592E: Pointer to member <member> can not be surrounded by parentheses

Assigning a pointer to a member to pointer-to-member-type variable requires a very precise syntax.
The member must be qualified and parentheses are not allowed around the membername. See C++
Standard Chapter 5.3.1 section 2.

C2593E: ANSI C disallows unwidened type <type>

In strict ANSI C mode, enum, char, short and float cannot be used as arguments to va_start or
va_arg.

C2594E: Array or function types cannot be used in va_arg

Array and function types cannot be used as arguments to va_arg.

C2596E: base <class> has no default constructor/destructor

C2597E: <name> cannot be dynamically initialized

The const object cannot be initialized dynamically because it has extern "C" linkage or does not
have extern "C++:read/write" linkage when compiled /ropi or /rwpi.

C2598E: <name> cannot be const because it contains a mutable member

The object cannot be const because it contains a mutable member and has extern "C" linkage or
does not have extern "C++:read/write" linkage when compiled /ropi or /rwpi.

C2599E: type <Type1> inconsistent with <Type2>, ignored

C2600E: floating point constant '123.4' is not permitted with -fpu none

C2601E: definition of floating point function <function> is not permitted with -fpu
none

C2602E: Missing or bad Template Formal

C2603E: __value_in_regs used with a structure type that needs construction or
destruction

It is not possible to use __value_in_regs with a structure type that needs construction or
destruction, since this is a kind of temporary introduction that is non-trivial.

C2604E: this template declaration cannot have extern "C" linkage
 Templates (including class template partial specializations) and template explicit specializations may not have extern "C" linkage.

C2605E: this declaration may have at most one declarator

C2606E: class template declarations may not have any declarators

C2607E: <class> cannot be have an initializer because it has no members and no constructor

C2608E: incompatible with function definition

C2609E: Cannot form pointer to (__swi) function

C2610E: Function type modifier duplicated through typedef

C2611E: floating-point formats in printf/scanf are not permitted with -fpu none

C2612E: unspecialized template base
 Base classes cannot be unspecialized templates.

C2613E: #ident is not allowed in strict ANSI mode

C2614E: initialization can't be done statically
 Static initializers involving pointers can't be done for types narrower than pointers.

C2615E: alignment not power of 2 : ignored
 In __align(n), n must be a power of two.

C2616E: __align not allowed for function declaration

C2617E: alignment for an auto object may not be larger than 8: 8 assumed
 For example:
 __align(16) int foo = 10;
 is not allowed for a local variable foo, so the warning is given.

C2618E: Error in _Pragma(string) syntax, ignored

C2619E: 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

C2620E: __align is not allowed on a non-static datamember

C2621W: double constant automatically converted to float
 This is given when the default type of unqualified floating-point constants is changed by the option "-auto_float_constants". This warning is enabled by default, but can be suppressed with "-Wk".

C2622E: 'typename' cannot be followed by xx, it must be followed by '::' or <identifier>

C2623W: Tentative open array is not ISO C++ compliant

C2624W: Thumb inline assembler will not be supported in future releases of the compiler

The Thumb inline assembler is supported in ADS 1.2, but will not be supported in the next release of the tools. ARM inline assembly will continue 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. Please contact your supplier if you need more information.

C2801W: unknown option '<option>': ignored

C2803W: can't open pre-include file 'filename' (ignored)

C2806W: undefined behaviour: <name> written and read without intervening sequence point

For example: `int f(int i) { int j = i + (i = 4); return j; }`

C2807W: undefined behaviour: <name> written twice without intervening sequence point

For example: `int f(int i) { int j = (i = 5) + (i = 4); return j; }`

C2808W: ANSI '...' trigraph for '...' found - was this intended?

For example: `int f() { return '??/n'; }`

C2809W: character sequence <string> inside comment

Comments contained within `/*` and `*/` do not nest: occurrence of the characters `/*` within a comment is likely to be because:

- an earlier `*/` sequence has been omitted
- an attempt has been made to comment out with `/*` and `*/` a block of code which contained comments. Either comment out each line individually using `//`, or use `#if 0` and `#endif` for this.

C2810W: (possible error): `>= xxx` lines of macro arguments

C2811W: repeated definition of `#define` macro <Macro>

<Macro> has been defined twice (with identical replacement strings).

C2812W: Non-ANSI `#include <myHeader.h>`

`-Wp` option suppresses this warning message.

The ANSI C standard requires that you use `#include <...>` for ANSI C headers only. However, it is useful to disable this warning when compiling code not conforming to this aspect of the standard. This warning is suppressed by default unless you specify the `-strict` option.

C2813W: Unrecognised `#pragma` or `_Pragma`

Pragmas non-specific to the ARM compilers will produce this warning. You can omit your `#pragmas` from ARM specific builds by use of conditional compilation, for example:

replace:

```
#pragma unknown_pragma
```

with:

```
#ifndef __arm                // If __arm not defined
#pragma unknown_pragma      // include this #pragma
#endif
```

'__arm' is automatically defined by armcc/tcc.

Similarly, you can choose to include #pragmas for a specific compiler using #ifdef. e.g:

```
#ifdef __arm          // If __arm defined
#pragma arm_pragma    // include this #pragma
#endif
```

C2816W: Unbalanced #if/#ifdef/#ifndef/#endif in file

C2817W: argument <A> of macro <M> expanded in "..."

In PCC mode, macro arguments are replaced even inside string constants.

Exploitation of this is often unintentional.

C2819W: 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:

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

This warning is off by default. It can be enabled with -W+g.

C2820W: File is guarded by 'foo_h' but does not #define it

C2821W: trailing '\' continues comment

Removal of line continuations (\ newline sequences) happens at an earlier stage in translation than comment removal, so a comment introduced by // will apply to the next physical line too if its last character is \.

C2822W: preprocessor directive ignored in macro argument list

C2823W: Unmatched quote (") in skipped line

C2826E: differing redefinition of #define macro <Macro>

<Macro> has been defined twice (with different replacement strings).

If you need to do this, undefine the macro (#undef) before the second definition.

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
```

```
int foo()
{
    return TEST;
}
```

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.

C2827E: duplicate macro formal parameter

C2828E: operand of # not macro formal parameter

Example:

```
#define MOV(A, B) __asm { mov (A), (B), asr #0; }
```

While performing macro expansion, the # character is treated as special by the preprocessor to indicate that the following parameter should be placed in quotes ("). When the preprocessor sees 'asr #0' it tries to find the parameter named 0 and place it in quotes. The only parameters defined for the macro are A and B, so it complains that it cannot find a macro parameter named 0.

Fortunately, it is not necessary to use the # for an immediate value to asr. In the example you do not even need the asr since it has no effect. The macro could be written like this:

```
#define MOV(A,B) __asm{ mov (A), (B)}
```

If you do want to specify a rotation value you can use either a decimal or hex value like this:

```
#define MOV(A,B) __asm{ mov (A), (B), asr 16} // a decimal rotation value
#define MOV(A,B) __asm{ mov (A), (B), asr 0x10} // a hex rotation value
```

C2829E: ## first or last token in #define body

C2830E: missing newline before EOF - inserted

A newline was expected, not an End Of File. A newline was inserted.

C2831E: unprintable char found - ignored

C2832E: illegal option -D

Incorrect use of -D on the compile line, for example, "-D##"

C2833E: spurious #else ignored

C2834E: spurious #elif ignored

C2835E: spurious #endif ignored

Unexpected preprocessor statements were found - please check and correct your source code.

C2836E: number missing in #line

A #line statement was found without a number. The correct format is e.g.:

```
#line 27
```

C2837E: #error encountered <string>

A warning is given because a deliberate #error trap was encountered, when compiling with "-Ep". The warning can be upgraded to the error C2860E by removing the "-Ep".

C2839E: junk at end of #endif line - ignored

This error may be suppressed with "-Ep".

For example:

```
#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 */
```

C2840E: EOF in comment

C2841E: EOF in string

The comment/string were not closed correctly before the End Of File.

C2842E: quote (") inserted before newline

For example:

```
char foo[] = {"\" };
```

C2843E: EOF in string escape

The string escape sequence was not closed correctly before the End Of File.

C2844E: Missing '>' in pre-processor command line

For example:

```
#include <stdio.h
```

C2845E: No identifier after #if defined

C2846E: No ')' after #if defined(...

Incorrectly formatted preprocessor statements - please check and correct your source code.

C2847E: Missing ')' after func(... on line xxx

C2848E: Too many arguments to macro macroName(... on line xxx

C2849E: Too few arguments to macro macroName(... on line xxx

C2850E: Missing identifier after #define

C2851E: Missing parameter name in #define

C2852E: Missing ',' or ')' after #define

C2853E: Missing identifier after #undef

C2854E: Missing identifier after #ifdef

Incorrectly formatted preprocessor statements - please check and correct your source code.

C2855E: Missing '<' or '"' after #include

For example:

```
#include stdio.h>
```

C2856E: Junk after #include

For example:

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

C2857E: #include file <file.h> wouldn't open

C2858E: Unknown directive: #baddirective

C2859E: Missing #endif at EOF

An open #if was still active, but was not closed with #endif before the End Of File.

C2860E: #error encountered <string>

Fatal error : Compilation was abandoned because a deliberate #error trap was encountered.

This error can be downgraded to the warning C2837E by compiling with the "-Ep" switch.

C2861E: Host file system read error

C2863W: unused earlier static declaration

This warning is activated by '-fh'.

C2864W: extern <name> not declared in header

C2865W: extern 'main' needs to be 'int' function: 'int' assumed

C2866W: label <label> was defined but not used

C2867W: typedef <typedef> declared but not used

C2868W: typename <type> declared but not used

C2869W: function <func> declared but not used

C2870W: variable <var> declared but not used

C2871W: static <static> declared but not used

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 "-Wx" option suppresses these warnings.

C2872W: implicit return in inlined non-void function

C2873W: implicit return in non-void function

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

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

C2874W: <name> may be used before being set

C2876W: ANSI surprise: 'long' <...> 'unsigned' yields 'long'

C2877W: actual type <type> of argument mismatches format

For example:

```
unsigned long foo = 0x1234;
printf("%0X", foo);
```

results in the warning:

C2877W: actual type 'unsigned long' of argument 2 mismatches format '%0X'

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 ints and longs being 32 bits wide on an ARM.

C2878W: Illegal format conversion

C2879W: Incomplete format string

C2880W: Format requires 0 parameters, but xxx given

C2881W: Format requires 1 parameter, but xxx given

C2882W: Format requires xxx parameters, but yyy given

C2885W: division by zero

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.

C2887W: implicit constructor

-wi suppresses the implicit constructor warning (C++ only). It is issued when the code requires a constructor to be invoked implicitly. This warning is suppressed by default. It can be enabled with -W+i.

Example:

```
struct X { X(int); };
X x = 10;    //actually means, X x = X(10);
            //See the Annotated C++ Reference Manual p.272
```

C2888W: explicit cast to same type

This warning is activated by -fp. This warning can normally be ignored.

C2891W: unsigned constant overflow

C2892E: signed constant overflow
C2893E: unsigned constant overflow
C2894E: signed constant overflow

C2895E: implementation defined shift (treated as 8 bit unsigned)

The shift amount is larger than the number of bits in the promoted type of its first operand, or the amount is negative. The shift amount is truncated to an unsigned 8-bit value.

C2896E: division by zero

A divide or remainder operator has a second operand with value 0. This is an error if it occurs in a constant expression, or if execution reaches the expression.

C2897W: odd unsigned comparison with 0

The expression `<expr><=0`, where `<expr>` is unsigned, has the same value as `<expr>==0`. However, its use may indicate that there is a misunderstanding as to the sign of `<expr>`.

C2898W: unsigned comparison with 0 is always false

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.

C2899W: unsigned comparison with 0 is always true

See C2898W.

C2900E: floating-point constant overflow

C2901E: signed constant overflow

This can occur with enums, for example, it is common to use enums to represent bit positions in a memory mapped register, like this:

```
typedef enum  
{  
    Bit0   = 0x00000001,  
    Bit1   = 0x00000002,  
    Bit2   = 0x00000004,  
    :  
    Bit30  = 0x40000000,  
    Bit31  = 0x80000000  
} Bits;
```

This is not allowed, because Bit31 exceeds 0x7FFFFFFF, the upper bound for a signed int, so the compiler reports: Warning: C2901E: signed constant overflow: 'enum'

An error is given with `-strict`. See ADS 1.2 Compilers and Libraries Guide, section 3.3.4, "Structures, unions, enumerations, and bitfields".

The workaround is to change Bit31 to:

```
    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. Another workaround is to compile with `"-wb"`.

C2905E: implicit cast (to `<type>`) overflow

C2906E: implicit cast of pointer loses `<qual>` qualifier

For example: "implicit cast of pointer loses '__packed' qualifier"

The compiler has performed an implicit cast of a packed pointer, and is reporting an error to avoid the __packed qualifier being cast away unintentionally. The simplest solution for this is to make all casts of packed pointers explicit in the source code.

C2907W: deprecated conversion of string literal to pointer to non-const

C2908E: too many arguments for overload resolution

C2909E: operand of <name> cannot be 'bool'

(C2910E is obsolete)

C2911W: deprecated use: operand of <name> is 'bool'

C2912W: cast to 'bool' (other than integer 0 or 1)

C2913W: floating to integral conversion overflow

C2914W: out-of-bound offset xxx in address

C2915W: lower precision in wider context

An expression is evaluated in one type <Type1>, and the result is then converted to a wider type <Type2>. Where <Type1> and <Type2> are both implemented as hardware types of the same size (for example, <Type1> is int and <Type2> long), this just warns of a portability hazard. However, where <Type2> is long long the result of evaluating the expression may differ from that of evaluating it in the wider type throughout.

Example:

```
long x; int y, z; x = y*z
```

where the multiplication yields an int result that is then widened to long. This warning indicates a potential problem when either the destination is long long or where the code has been ported to a system that uses 16-bit integers or 64-bit longs. This option is off by default. It can be enabled with -W+l.

C2916W: use of = in condition context

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 silence the warning.

This warning can be suppressed with -Wa.

C2917W: no side effect in void context

C2918W: argument and old-style parameter mismatch

C2919W: 'format' arg. to printf/scanf etc. is variable, so cannot be checked

This warning is being given when printf- or scanf-like format checking is enabled with:

```
#pragma check_printf_formats
```

or

```
#check_scanf_formats
```

but the format string in the code is variable, so it cannot be checked.

This is explained in the ADS 1.2 Compilers and Libraries Guide, section 3.1.1, "Pragmas".

This warning cannot be disabled, except by removing the `#pragma`, by adding `#pragma no_check_printf_formats` before that line, or by changing the code so that it passes the checks.

C2920W: implicit cast from (void *), C++ forbids

This is warning about future compatibility with C++. Example:

```
int *new(void *p) { return p; }
```

This warning is suppressed by default, but can be enabled by `-W+u`.

C2921W: implicit narrowing cast from <FromType> to <ToType>

A value is being implicitly cast in a way that loses information. This warning is suppressed by default, and may be enabled with `"-W+n"`. Using an explicit cast will prevent the warning.

Example:

```
int w, x, y;
unsigned short h, i, j, k;

int main(void)
{
    w = x * y;    // OK
    h = i * j;    // warning
    k = (unsigned short)(i * j); // no warning
}
```

when compiled with `-W+n`, results in the warning:

C2921W: '=': implicit narrowing cast from 'int' to 'unsigned short'

To illustrate the loss of information that may occur, the `MUL` instruction used to perform the calculation will give a 32-bit result, which is reduced by the implicit cast to 16-bits by the following `STRH` instruction, and so the implicit cast is important to report:

```
0x0000001c:    e0010192    ....    MUL        r1,r2,r1
0x00000020:    e1c010b0    ....    STRH        r1,[r0,#0]
```

C2922W: cast between function pointer and non-function object

C2923W: explicit cast of pointer to <type>

This warning is activated by `-fP`. This warning indicates potential portability problems. Casting explicitly between pointers and integers is not harmful on the ARM processor because both are 32-bit types. In general, special care should be taken with alignment.

C2924W: 'this' unused in non-static member function

This warning is issued when the implicit "this" argument is not used in a non-static member function. It is applicable to C++ only. This warning is suppressed by default. It can be enabled with `"-W+t"`. The warning can also be avoided by making the member function a static member function. Example:

```
struct T {
    int f() { return 42; }
};
```

To avoid the warning, use `static int f() ...`

C2925W: C++ scope may differ

C2926W: <name> invented in parameter list

C2927E: attempt to use <Func>

C2930E: duplicate definition of <tag>

C2931E: re-using <name> as <tag> tag

C2932E: incomplete tentative declaration of <name>

C2933E: type disagreement for <name>

C2934E: duplicate definition of <name>

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.

In ADS 1.0.1, tentative declarations were disallowed by default, unless "-strict" is specified.

In ADS 1.1 and later, tentative declarations are allowed by default

C2935E: duplicate definition of label <name> - ignored

C2936E: label <name> has not been set

C2937E: static function <name> not defined - treated as extern

C2938E: conflicting global register declarations

C2939E: storage class not allowed for friends -- ignored

C2940E: attempt to specialize after use

C2941E: attempt to instantiate undefined template

C2942E: no template found to specialize or instantiate

C2943E: ambiguous templates to specialize or instantiate

C2944E: re-declaration of template type parameter

C2945E: Overlarge floating-point value found

C2946E: Overlarge (single precision) floating-point value found

C2947E: Illegal types for operands

C2948E: size of <struct/class> needed but not yet defined

C2950E: Illegal in lvalue: function or array

C2951E: bit fields do not have addresses

C2952E: Illegal in l-value: 'enum' constant

C2953E: Illegal in the context of an l-value

C2957E: illegal in constant expression: <unknown>

C2958E: illegal in constant expression: non constant

C2959E: illegal in constant expression

C2960E: illegal in floating type initialiser: <unknown>

C2961E: illegal in floating type initialiser: non constant

C2962E: illegal in floating type initialiser:

C2963E: illegal in static integral type initialiser: <unknown>

C2964E: illegal in static integral type initialiser: non constant

C2965E: illegal in static integral type initialiser

These errors can occur when a non-constant expression is being assigned to a static object. ANSI C requires the initializer for a static object to be a constant expression.

C2966E: attempt to apply a non-function

C2967E: 'void' values may not be arguments

C2968E: illegal cast of <type> to pointer

C2969E: illegal cast to <type>

C2970E: cast to non-equal <type> illegal

C2971E: <struct/class> not yet defined - cannot be selected from

C2972E: <struct/class> has no <field> field

C2973E: no 'this' pointer to access member
C2974E: no 'this' pointer for member function

C2975E: duplicate declaration of default value for argument

C2976E: missing default value for argument

C2977E: non-call site '.*' or '->*' yielding function

C2978E: illegal left operand
 The left operand for a member selection does not have struct type.

C2979E: <name> is a global register variable - can't take its address

C2980E: construction of value of type <type> is recursively confused

C2981E: no constructor for <class> at this type signature

C2982E: non-lvalue cast to non-const reference

C2983E: template non-type <foo> has no storage
C2984E: template member expected
C2985E: template parameter <arg> re-defined

C2986E: array <name> too large
 See error message: C2496E.

C2987E: illegal use of member function - '' intended?

C2989E: <tag> cannot be declared here

C2991E: linkage disagreement for foo - treated as 'extern'
 -E1 option suppresses errors about linkage disagreements where functions are implicitly declared as extern and then later re-declared as static.

C2992E: f() has no linkage, but g() requires it to have
C2993E: linkage disagreement between f() and g()
C2994E: 'f()' was previously declared without "C" linkage

C2995E: duplicate typedef

C2996E: extern f() mismatches top-level declaration

C2997W: <Derived Class Function> inherits implicit virtual from <Base Class Function>
 This warning is issued when a non-virtual member function of a derived class hides a virtual member of a parent class. For C++, the "-wr" option suppresses the implicit virtual warning.
 For example:

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

 gives:
 Warning : C2997W: 'Derived::f()' inherits implicit virtual from 'Base::f()'

Adding the virtual keyword in the derived class prevents the warning.

C2998E: inherited virtual function type differs

C2999E: top qualifiers in covariant return type are not equal to those of the inherited return type

C3000E: covariant return type is not equally or less qualified than the inherited return type

C3001E: covariant return type of function <Func>: <type1> is not derived from <type2>

C3002E: covariant return type: can access base <type>; either type <type> is being defined or base is non public

C3003E: covariant return types in multiple inheritance is not supported

C3004E: attempt to overload non-function

C3005E: <name> is an ambiguous name in <class>

C3006E: <name> can't be overloaded

C3007E: <name> must be of type 'void (void*)' or 'void (void*, size_t)'

C3008E: '::operator delete' must be of type 'void (void*)'

C3009E: <name> must have a return type of 'void*' and have a first argument of type 'size_t'

C3010E: more than one <name> has "C" linkage

C3011E: <name> inherits both virtual and non-virtual attributes from bases of <class>

C3012E: iffy arithmetic shift

C3013E: small floating-point value converted to 0.0

C3014E: small (single precision) floating value converted to 0.0

C3015E: sizeof <bit field> illegal - sizeof(int) assumed

C3016E: size of 'void' required - treated as 1

C3017E: size of a [] array required, treated as [1]

C3018E: size of function required - treated as size of pointer

C3021E: assignment to 'const' object

C3022E: 'register' attribute for <Variable> ignored when address taken
Variables declared with storage class "register" may not have their address taken. Taking the address of <Variable> will ensure that it cannot be allocated to a register.

C3023E: objects that have been cast are not l-values

C3024E: comparison of pointer and int
Comparison of a pointer with an (un-cast) integer is permitted only when the integer is a literal 0 (a null pointer constant).

C3025E: differing pointer types
Comparison of two pointer values is permitted only if they have the same type.

C3026E: wrong number of parameters to <Function>
The number of arguments in a call to <Function> either
- (<Function> declared with a non-variadic prototype) does not match the declared number of parameters
- (<Function> declared with a variadic prototype) does not match the declared fixed number of parameters

- (<Function> declared without a prototype) does not match the number of parameters in an earlier definition of the function

For example:

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

C3027E: cast of <FromType> to differing <ToType>

Implicit cast of a value with type <FromType> to a different <ToType>.

This is illegal in C++, so also warned of in C as a C++ compatibility problem.

C3028E: implicit cast of pointer to non-equal pointer

C3029E: implicit cast of non-0 int to pointer

C3030E: implicit cast of pointer to 'int'

C3031E: implicit cast of <type> to 'int'

Try re-writing the code to avoid the implicit cast, e.g. add an explicit cast.

These errors can be suppressed with "-Ec".

C3032E: <Function> is a non-public member

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

Example:

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

gives:

Error: C3032E: 'A::f' is a non-public member

C3033E: differing pointer types

The two results of a conditional expression have differing pointer types.

C3034E: illegal indirection on (void *): '*'

C3035E: non-call site overload

C3036E: cast to derived type from virtual base

C3038E: too many arguments: <type> constructor

C3039E: I/O error on object stream

C3040E: no external declaration in translation unit

Typically this message is generated when an empty source file is compiled.

C3041U: I/O error writing

C3042U: load file created with a different version of the compiler

The ARM compilers do not support precompiled headers.

C3046U: out of store (for error buffer)

C3047U: Too many errors

C3048U: out of store [cc_alloc(N)] while compiling -g

C3049U: out of store [cc_alloc(N)]

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 broken into smaller pieces, may help.

C3050U: Compilation aborted.

C3051E: couldn't write file 'filename'

C3052E: couldn't read file 'filename'

C3056E: bad option

C3057E: bad option

For example, the switches "-apcs /softfp", "-apcs /narrow", "-apcs /wide" which were supported in SDT, are no longer supported in ADS, and so must be removed from the compiler command-line.

C3059E: Missing file argument for '<option>'

<option> requires a file parameter, e.g. -errors err.txt

C3060E: No argument to compiler option

C3061E, C3062E, C3063E: unknown option

Examples:

Error: C3061E: unknown option '-fz': ignored

Error: C3063E: unknown option '-zal': ignored

These compiler options were commonly used in build scripts for SDT, however these are no longer supported by ADS. You should remove these switches from any makefiles. See the Migrating Projects from SDT to ADS section of the ADS Getting Started Guide for further details.

C3064E: Overlong filename: filename

C3065E: type of '<variable>' unknown

C3067E: Couldn't write installation configuration

C3073E: options -E and -M conflict: -E assumed

C3074E: Can't open -via file filename

C3075E: Can't open filename for output

C3078E: stdin ('-') combined with other files

C3079E: command with no effect

C3396E: Source file-name 'filename' cannot be configured

C3399E: 'main' cannot be qualified with: <type>, (ignored)

C3400E: non-lvalue in unary operator

C3401E: illegal cast from <Type>

Expressions of type <Type> cannot be explicitly cast.

C3403E: __alloca_state not defined

C3404W: division overflow

Constant propagation shows that a divide or remainder operator divides the maximum negative number by -1. The result would be one greater than the maximum positive signed number.

C3410W: Option <option> has been obsoleted

For example:

armcc -dwarf1

Error: C3410W: Option -dwarf1 has been obsoleted
Use -dwarf2 instead

C3411U: Unable to recover from a previous error

C3412E: floating-point constant underflow

C3413E: invalid floating-point constant operation

C3414E: floating-point constant division by zero

C3415W: invalid floating to integral conversion

C3417E: floating-point op is not permitted with -fpu none

C3418E: call of floating-point function is not permitted with -fpu none

C3420E: cast to floating-point type is not permitted with -fpu none

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.

C3422E: __irq functions must take no arguments and return no result

C3423E: __irq functions must not be the target of a function call

C3424E: too many arguments for __swi or __swi_indirect function

C3425E: arguments for __swi or __swi_indirect function must be passed in integer registers

C3426E: __swi_indirect function must have arguments

C3427E: first argument for __swi_indirect function must have integral type

C3428E: result of __swi or __swi_indirect function must be returned in integer registers

The __swi or __swi_indirect function is malformed. Up to 4 integer arguments may be passed in.

C3429W: explicit cast of non-0 int to pointer

This warning is activated by -fp. This warning indicates potential portability problems. Casting explicitly between pointers and integers is not harmful on the ARM processor because both are 32-bit types. In general, special care should be taken with alignment.

C3430E: expected ')' after '...' in macro parameter list

C3431E: cannot load constraints file for feature set <filename>

C3433E: <option> selection is incompatible with restrictions in '<constraintfile>'

The feature-restricted toolkit that uses a 'constraints file' is not installed correctly. Try re-installing.

C3434E: <func> must be a function

C3435E: reference to FOO not allowed

C3447E: option -E and inputfile type conflict

C3448W: floating-point constant comparison is unordered

A comparison between two floating-point values is unordered if one or more of them is a NaN.

C3449W: static initialisation of <foo> using address of <bar> may cause link failure -rwp

C3450W: static initialisation of <foo> by string literal may cause link failure -ropi

These two new warnings were added in ADS 1.2 to 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 : C3450W: static initialisation of 'str' by 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 *foo = &bar; /* global pointer */
```

when compiled with `-apcs/rwpi` gives:

Warning : C3449W: static initialisation of '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 ADS FAQ "What does "Error: L6248E: cannot have address type relocation" mean?" at: http://www.arm.com/support/ads_faq

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 ADS 1.2 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.

A1022U: Error on stdin: exiting

An error occurred whilst reading source from the standard input.

This can be caused by termination of the command piping source to the assembler.

A1023E: File "<filename>" could not be opened

The given file either did not exist, could not be found, or could not be opened.

Check that the correct path for the file is specified.

A1024E: File "<filename>" could not all be loaded

An error occurred when trying to read the file.

A disk error or a file sharing conflict could cause this.

A1042E: Unrecognized APCS qualifier '<qualifier>'

There is an error in the argument given to the -apcs command line option.

Check the spelling of <qualifier>.

A1046E: Via file '<filename>' would not open

The file given in the -via <filename> command line option either did not exist, could not be found, or could not be opened.

Check that the correct path for the file is specified.

A1047E: Missing argument to '<option>'

No argument was given for the command line option <-option>.

A1048E: Bad architecture specified

The argument to the -arch or -fpu command line option was not recognized. Check the spelling of the argument.

A1051E: Cannot open -depend file '<filename>'

The file given in the -depend <filename> command line option either did not exist, could not be found, or could not be opened. Check that the correct path for the file is specified.

A1055E: Cannot open -errors file '<filename>'

The file given in the `-errors <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.

A1056E: Target '`<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.

A1057E: Target cpu missing

No argument was given for the `-cpu` (or `-proc`) command line option.

A1058E: Input file specified as '`<file1>`', but it has already been specified as '`<file2>`'

More than one input file has been specified on the command line. Misspelling a command line option can cause this. Only one input file may be given on the command line. To assemble multiple files, it is necessary to invoke the assembler multiple times.

A1063W: Listing file page length out of range, ignored

The argument to the `-length` command line option was either negative, or too large. The value of the argument should be 0 (which specifies an unpagged listing), or a positive value less than 256.

A1065E: Bad value for `-maxcache`

Check the argument to the `-maxcache` command line option.

The argument must be a positive integer, either decimal or hexadecimal. Hexadecimal values must be preceded by `0x` or `&`. Note that `&` is recognized by some command line interpreters as a special character, and so may need to be escaped.

A1066W: Negative value for `-maxcache`, ignored

Check the argument to the `-maxcache` command line option. The argument may not be negative.

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

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.

A1077W: Width out of range, ignored

The argument to the `-width` command line was either too large or too small.

The `-width` option is ignored, and the default page width is used instead. This warning may be produced because the argument to the `-width` command has been accidentally omitted from the command line, or is not a decimal number.

A1079E: Unrecognized command line option '<option>'

<option> is not a valid command line option. Check the spelling of <option>.

A1080E: Bad command line operand '<operand>'

The only operands that may be specified on the command line are the source file and the object file, in that order. Any subsequent operands will be flagged with this error.

This error could be caused by attempting to specify multiple source files on the command line (this is not allowed), or by misspelling a command line option or option argument.

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 ARM (2nd Ed), section 4.1.18, says:

"Banked registers: 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

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

A1108E: Multiply or incompatibly defined symbol

A1109E: Bad expression type

A1110E: Expected constant expression

A constant expression was expected after, e.g. `SETA`. See the ADS 1.2 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 ADS 1.2 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]

A1116E: String operands can only be specified for DCB

A1117E: Register symbol 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 ADS 1.2 Assembler Guide, section 2.9, "Using macros".

A1120E: Bad global name

A1121E: Global name already exists

A1122E: Locals not allowed outside macros

A1123E: Bad local name

A1124E: Local name already exists

A1125E: Unknown or wrong type of global/local symbol

A1126E: Bad alignment boundary

A1127E: Bad IMPORT/EXTERN name

A1128E: Common name already exists

A1129E: Imported name already exists

A1130E: Bad exported name

A1131E: Bad exported symbol type

A1133E: Bad required symbol name

A1134E: Bad required symbol type

A1135E: Area name missing

AREA names starting with any non-alphabetic character must be enclosed in bars,e .g:
change:

AREA l_DataArea, CODE, READONLY

to:

AREA |l_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 too short for operation

A1139E: String overflow

A1140E: Bad operand type

There is a known problem with ADS 1.1 armasm which can trigger this. See ADS FAQ "ADS 1.1 armasm: Error: A1140E: Bad operand type " at: http://www.arm.com/support/ads_faq

A1142E: Subtractive relocations not supported for ELF format output

There is a known problem with ADS 1.1 armasm that can trigger this. See ADS FAQ "ADS 1.1 armasm: Error: A1142E: Subtractive relocations not supported for ELF format output" at: http://www.arm.com/support/ads_faq

A patch for ADS 1.1 armasm is available from: <http://www.arm.com/support/downloads>

For ADS 1.2, 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 ADS ELF, so this error message can sometimes appear when migrating an SDT project to ADS.

To resolve this for ADS, 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**A1146E:** Unable to open output file**A1147E:** Bad shift name**A1148E:** Unknown shift 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
```

Please see the ARM Architecture Reference Manual 2nd edition, section 5.1.7, "Data-processing operands - Logical shift right by immediate"

A1150E: Bad symbol

This typically occurs in two cases:

1) when the current file requires another file to be INCLUDED 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 IMPORTED, 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 work. 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

A1155E: Unexpected unary operator

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 ADS 1.2 Assembler Guide, section 3.5.6, "Local labels."

A1161E: Syntax error following local label definition

A1162E: Incorrect routine name

A1163E: Unknown opcode

The most common reasons for this are:

- 1) 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
```

- 2) 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
```

- 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, section 4.2, "ARM instructions and architecture versions". 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

A1166E: Syntax error following label

A1167E: Invalid line start

A1168E: Translate not allowed in pre-indexed form

A1169E: Missing close square bracket

A1170E: Immediate 0xNNNN out of range for this operation
 This error is given if a MOV or MVN instruction is used with a constant that cannot be assembled.
 See ADS 1.2 Assembler Guide, section 2.6.1, "Direct loading with MOV and MVN".

A1171E: Missing close bracket

A1172E: Bad rotator

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 out of range

A1175E: Bad register range

A1176E: Branch offset out of range
 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 ADS 1.2 Assembler Guide, section 3.5.6, "Local labels".

A1179E: Bad hexadecimal number

A1180E: Missing close quote

A1181E: Bad operator

A1182E: Bad based 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 <mini> to <maxi>

A1189E: Missing '#'

A1190E: Unexpected '<character>'

A1191E: Floating point register number out of range

A1192E: Coprocessor register number out of range

A1193E: Coprocessor number out of range

A1194E: Bad floating-point number

A1195W: Small floating point value converted to 0.0

A1196E: Too late to ban floating point

A1197W: Precision specifier ignored for 'FIX'

Unlike some of the other FPA instructions, "FIX" has no precision specifier

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:
```

```
armasm init.s -g -PD "ROM_RAM_REMAP SETL {FALSE}"
```

See ADS 1.2 Assembler Guide, section 3.5.4, "Assembly time substitution of variables"

A1199E: Coprocessor operation out of range

A1200E: Structure mismatch

A1201E: Substituted line too long

A1202E: No pre-declaration of substituted symbol

See ADS 1.2 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 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 in ADS - simply remove them.

A1209E: ADRL can't be used with PC

A1210E: Non-zero data within uninitialized area

A1211E: Missing open square bracket

A1212E: Division by zero

A1213E: Attribute <Attr1> cannot be used with attribute <Attr2>

A1214E: Too late to define symbol as register list

A1215E: Bad register list symbol

A1216E: Bad string escape sequence

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 <address> must be unconditional

A1234E: Undefined or Unexported Weak symbol

A1237E: Invalid register or register combination for this operation

A1238E: Immediate value must be word aligned for this operation

A1239E: Shift count out of range

A1240E: Immediate value cannot be used with this operation

A1241E: Must have immediate value with this operation

A1242E: Offset must be word aligned with this operation

A1243E: Offset must be halfword aligned with this operation

A1244E: Missing '!'

A1245E: B or BL from 16 bit code to 32 bit code

A1246E: B or BL from 32 bit code to 16 bit 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.

A1247E: BLX from 32 bit code to 32 bit code, use BL

A1248E: BLX from 16 bit code to 16 bit 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.

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.

A1255E: Operand to LDRB does not fit in 8 bits

A1256E: DATA directive can only be used in CODE areas

A1259E: Invalid PSR field specifier

A1261E: MRS cannot select fields, use CPSR 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

A1267E: Bad GET or INCLUDE

A1268E: Unmatched conditional or macro

A1270E: File "<filename>" not found

A1271E: Line too long

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 ADS 1.2 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

A1291E: Macro definitions cannot be nested
The macro definition is invalid.

A1310W: Symbol attribute not recognized

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

A1318W: TSTP/TEQP/CMNP/CMPP inadvisable in 32-bit PC configurations
These obsolete 26-bit architecture instructions are no longer supported.

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

A1323E: Reserved instruction (Rm = Rn with post-indexing)

A1324E: Undefined effect (PC + writeback)

A1325E: Undefined effect (destination same as written-back base)

A1326E: Undefined effect (PC used in a non-word context)

A1327W: Non portable instruction (LDM with writeback and base in reg. list)
 See ARM Architecture Reference Manual (ARM ARM), section 4.1.17, "LDM", Operand restrictions:
 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)
 See ARM Architecture Reference Manual (ARM ARM), section 4.1.42, "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 xfer 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

A1355U: A Label was found which was in no AREA
 Example:
 A common case where this case occur is 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, section 4.2, "ARM instructions and architecture versions".

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
A1411E: Closing '>' missing from vector specifier
A1412E: Bad vector length
A1413E: Bad vector stride
A1414E: Vector wraps round over itself
A1415E: VFPASSERT must be followed by 'VECTOR' or 'SCALAR'
A1416E: Vector length does not match current vector length
A1417E: Vector stride does not match current vector stride
A1418E: Register has incorrect type for instruction
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 may not be in first bank
A1426E: Source vector may not be in first 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 ADS 1.2 Assembler Guide, section 6, "Vector Floating-point Programming"

A1430E: Unknown frame directive
A1431E: Frame directives are not accepted outside of PROCs/FUNCTIONs

Invalid FRAME directive. See ADS 1.2 Assembler Guide, 7.5, "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.

A1436E: Memory access attributes '<attr1>' and '<attr2>' conflict
 Use of incorrect -memaccess arguments

A1446E: Bad or unknown attribute 'INTERWORK'. Use -apcs /interwork instead

Example:

```
AREA test, CODE, READONLY, INTERWORK
```

This code may have originally been intended to work with SDT. The INTERWORK area attribute is obsolete in ADS. To eliminate the warning for ADS:

- 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)

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 FAQ "armasm: Use of MRS and MSR instructions ('Deprecated form of PSR field specifier')" at: <http://www.arm.com/support/faq>

A1454E: FRAME STATE RESTORE directive without a corresponding FRAME STATE REMEMBER Invalid FRAME directive. See ADS 1.2 Assembler Guide, 7.5, "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 obsolete in ADS. To eliminate the warning for ADS:

- remove the `", INTERWORK"` from the AREA line.
- 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 ADS.

Example:

```
AREA test1, CODE, READONLY
:
AREA test2, CODE, READONLY, INTERWORK
```

To eliminate the error for ADS:

- move the two AREAs into separate assembler file, e.g. `test1.s` and `test2.s`
- remove the `", INTERWORK"` from the AREA line in `test2.s`
- assemble `test1.s` with `'armasm -apcs /nointerwork'`
- 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.

A1460W: -arch will not be supported in future releases of the assembler: use -cpu
Simply change "-arch" to "-cpu"

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

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 new warning, added in ADS 1.1, 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 ADS 1.2 Assembler Guide, section 3.6.9, "Operator precedence".

A1467W: FRAME ADDRESS with negative 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 ADS 1.2 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
```

A1472E: Cannot load constraints file for feature set <filename>

A1473E: <option> selection is incompatible with restrictions in '<constraintfile>'

The feature-restricted toolkit that uses a 'constraints file' is not installed correctly. Try re-installing.

A1475W: At least one register must be transferred, otherwise result is UNPREDICTABLE.

A1476W: BX r15 at not word-aligned address is UNPREDICTABLE

A1477W: This register combination results in UNPREDICTABLE behaviour

A1479W: requested ALIGN is greater than area alignment, which has been increased
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 ADS 1.2 Assembler Guide, section 7.7.1, "ALIGN" and 7.7.2, "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>

A1483E: Obsolete command line option '<option>'
<option> is no longer a valid command line option.

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 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

A1492E: Immediate 0x<val> out of range for this operation. Permitted values are 0x<mini> to 0x<maxi>

A1495E: Target of branch is a data address

A1496E: absolute relocation of ROPI address 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.

A1497E: absolute relocation of RWPI address 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

A1504E: VFP rev1 is known to have problems storing this number of registers

4. ARM Linker (armlink) Errors and Warnings

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. In some cases, CodeWarrior may not free the used memory afterwards. Some possible solutions are to close down other applications, or increase the memory on your workstation, or close down/restart CodeWarrior.

L6001U: Could not read from file <filename>.

L6002U: Could not open file <filename>.

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>.

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 ELF and AOF formats, and library files in AR and ALF formats. The specified file is either corrupt, or is in a file format that the linker cannot recognize.

L6008U: Could not recognize the format of member <mem> from <lib>.

The linker can recognize library member objects written in ELF and AOF file formats. The specified library member is either corrupt, or is written in a file format that the linker cannot recognize.

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>.

An file I/O error occurred while reading /opening/writing to the specified file.

L6011U: Invalid integer constant : <number>.

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>.
The specified object (or library) does not have a valid symbol table. It may be corrupt - try rebuilding it.
- 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
- L6027U:** Relocation #NN in <objname>(<secname>) has invalid/unknown type.
This may occur in rare cases when linking legacy SDT AOF objects with the ADS linker. For backward compatibility, the ADS linker is able to accept most object files in the SDT AOF format and libraries in the SDT ALF format. These formats are obsolete and will not be supported in the future. However, not all AOF relocations are recognized in ADS. 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.
To resolve this, the object/library must be rebuilt with ADS.
- 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.
- L6030U:** AOF area <objname>(<areaname>) has unsupported attribute <attribute>.
The object file was built with SDT, probably in assembler, using the old (ARM proprietary) AOF format, which is now obsolete in ADS. The ADS linker is able to recognise most AOF relocations, but not all the old attributes. To resolve this, the object/library must be rebuilt with ADS.
- L6031U:** Could not open scatter description file <filename>.
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.
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>.
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
Relocations with respect to mapping symbols are not allowed. This may indicate a compiler fault – please contact your supplier.
- L6043U:** Relocation #NN in <objname>(<secname>) is wrt an out of range symbol
Relocations with respect to out of range symbols are not allowed. This may indicate a compiler fault – please contact your supplier.
- L6046U:** Recursive via file inclusion depth of <limit> reached
- L6047U:** Could not create the partial object as the number of ELF sections (<count>) exceeds the ELF maximum limit (<limit>).
The ELF file format has a limit on the number of sections, which is 2*16=65536 sections.
A test to check whether this limit is exceeded was added into ADS 1.2 armlink.
- L6200E:** Symbol <symbol> multiply defined (by <object1> and <object2>).
There are two common examples where this occurs:
- 1) Symbol __semlhosting_swi_guard multiply defined (by use_semi.o and use_no_semi.o).
This error is reported when functions that use semihosting SWIs are linked in from the C library, in the presence of the __use_no_semlhosting_swi guard. See the ADS 1.2 Compilers and Libraries Guide, section 4.2.2, "Building an application for a nonsemlhosted environment" and ADS 1.2 Developer Guide, Section 6.10.1, "Linker error __semlhosting_swi_guard".

To resolve this, you must provide your own implementations of these C library functions. The ADS 1.2 \Examples\embedded directory contains examples of how to re-implement some of the more common SWI-using functions - see the file retarget.c.

To identify which SWI-using functions are 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_swi'

For example:

```
:
Loading member sys_exit.o from c_a__un.l.
      reference : __I_use_semihosting_swi
      definition: _sys_exit
:
```

This shows that the SWI-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.

- 2) 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`, so see exactly what the linker is linking-in, from where, and why.

To move forward, the user may have to either:

- Eliminate the calls like `fopen`, `fclose`, `fflush`, etc, or
- Re-implement the `_sys_xxxx` family of functions.

See the ADS 1.2 Compilers and Libraries Guide, section 4.10, "Tailoring the input/output functions".

L6201E: Object <objname> contains multiple entry sections.

L6202E: Section <secname> cannot be assigned to a non-root region.

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:

L6202E: Section Region\$\$Table cannot be assigned to a non-root region.

L6202E: Section ZISection\$\$Table cannot be assigned to a non-root region.

To fix this, use a root region in the scatter-file containing, as a minimum, these three lines:

```
LOAD_ROM 0x0000 0x4000    ; start address and length
{
    EXEC_ROM 0x0000 0x4000 ; root (load = exec address)
    {
        __main.o (+RO)      ; copying code
        * (Region$$Table)   ; RO/RW addresses to copy
        * (ZISection$$Table) ; ZI addresses to zero
        :
    }
    :
}
```

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.

L6207E: Invalid argument for -keep/-first/-last command

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.

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 <secname>

Certain sections must be placed contiguously within the same region, for their base/limit symbols to be accessible.

For example, C\$\$pi_ctorvec, C\$\$pi_dtorvec and C\$\$ddtorvec are AREAs generated by the C++ compiler which must be placed in the same region with __cpp_initialise and __cpp_finalise. This can be done by specifying the sections in a scatter-loading description file:

```
LOAD_ROM 0x00000000
{
    EXEC_ROM 0x00000000
    {
        cpp_initialise.o(+RO)
        cpp_finalise.o(+RO)
        * (C$$pi_ctorvec)
        * (C$$pi_dtorvec)
        * (C$$ddtorvec)
        ....
    }

    RAM 0x01000000
}
```

```

    {
        * (+RW,+ZI)
    }
}

```

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 two common examples where this occurs:

- 1) Undefined symbol Image\$\$ZI\$\$Limit (referred from sys_stackheap.o).
 It is most likely that you have not re-implemented __user_initial_stackheap(). The ADS 1.2 \Examples\embedded directory contains examples of how to re-implement __user_initial_stackheap() - see the file retarget.c. Please see ADS FAQ "Re-implement __user_initial_stackheap() when using Scatterloading" at:
http://www.arm.com/support/ads_faq
 Please see also chapter 6 in the ADS 1.2 Developer Guide - Writing Code for ROM.
 Note that this condition failed silently in ADS 1.1, because the symbol Image\$\$ZI\$\$Limit took the value zero.
- 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 ADS projects, so the call to it must be removed. You should also remove your implementation of __rt_heapdescriptor() (if there is one).

L6219E: <type> section <object1>(<section1>) attributes {<attributes>} incompatible with neighbouring section <object2>(<section2>).

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.

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 may have an entry point. You will need to rebuild your objects to achieve this. 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.

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 <file> is empty.

L6230E: Multiple execution regions (<region1>,<region2>) cannot select veneer section <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:

- ensure `vectors.o` is specified on the linker command-line.
- link with "`-keep vectors.o`" to force the linker not to remove this, or switch off this optimization entirely, with `-noremove` [not recommended]
- [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
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>.

L6239E: Cannot call ARM/THUMB symbol '<sym>' in non-interworking object <obj2> from ARM/THUMB code in <obj1>(<sec1>)

Example:

L6239E: Cannot call ARM symbol 'ArmFunc' in non-interworking 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.

(**L6240E:** is no longer used in ADS.)

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.

This error typically occurs when attempting to link ADS objects/libraries with legacy objects/libraries built with the old Software Development Toolkit (SDT), or if you try to link objects built with different -fpu options. The recommended solution is to rebuild your entire project with ADS, with the same -fpu options. As a last resort, if you do not have the source code for an SDT object/library, then try recompiling your ADS code with '-fpu softfpa'.

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 C3449W and C3450W. See also ADS FAQ "What does "Error: L6248E: cannot have address type relocation" mean?" at: http://www.arm.com/support/ads_faq

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/-xref to command

L6253E: Invalid SYMDEF address: <number>.

L6254E: Invalid SYMDEF type : <type>.

The content of the symdefs file is invalid.

L6255E: Could not delete file <filename>.

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>

An I/O error occurred while trying to rename the specified file. File specified by newname may already exist.

L6257E: Section <secname> cannot be assigned to an 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.

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: Cannot express Region Table entry for <regionname> as either address or offset.

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 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.

If you are using ADS 1.0.1, this may be caused by a problem in armlink which is fixed in ADS 1.1 and later. When you specify '-ropi' to ADS 1.0.1 armlink without '-rw-base' it effectively specifies '-rwpi' too. To work-around this you should specify a base address for the RW region, e.g. '-ropi -rw-base 0x10000'.

L6268E: Non-word aligned address <addr> specified for region <regname>.

L6271E: Two or more mutually exclusive attributes specified for Load region <regname>

L6272E: Two or more mutually exclusive attributes specified for Execution region <regname>

L6273E: Section <object2>(<section2>) has mutually exclusive attributes (READONLY and ZI)

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 – contact your supplier.

L6276E: Address <addr> marked both as <s1>(from <obj1>) and <s2>(from <obj2>).

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: Execution region (<er_name>) cannot have explicit RELOC attribute. RELOC can only be inherited implicitly.

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 addressing.

L6285E: Non-relocatable Load region <lr_name> contains R-Type dynamic relocations.

L6286E: Value(<val>) out of range(<range>) for relocation #NN (wrt symbol <symname>) in <objname>(<secname>)

means that you have exceeded the limited number of bits for a field within the instruction opcode. For instance, it may be because of an LDR or STR where the offset is too large for the instruction (+/-4095 for ARM state LDR/STR instruction). It can also occur for a branch instruction where the ADS linker has been unable to insert a "long-branch veneer" at an appropriate place, e.g. because an execution region is too large (Thumb execution regions are currently limited to <4MB). For more information, see: http://arm.com/support/ads_faq

In ADS 1.1, the error message was slightly different:

L6286E: Relocation value out of range for relocation #NN in foo.o

- L6287E:** Illegal alignment constraint (<align>) specified for <objname>(<secname>).
An illegal alignment was specified for an ELF object (see L6334W).
- L6288E:** cannot load constraints file for feature set <filename>
- L6290E:** <option> selection is incompatible with restrictions in '<constraintfile>'
The feature-restricted toolkit that uses a 'constraints file' is not installed correctly. Try re-installing.
- 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 region has overflowed the 0xFFFFFFFF address limit - make the region smaller.
There is a known problem with the ADS 1.2 linker where this error is reported incorrectly for a region that fits exactly just below this upper limit. This problem is fixed in ADS 1.2 patch build 842 which can be downloaded from: <http://www.arm.com/support/downloads>
- L6295E:** SB Relative relocation requires image to be RWPI
- L6300W:** Common section <object1>(<section1>) is larger than its definition <object2>(<section2>).
- L6301W:** Could not find file <filename>.
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 is typical for an embedded system.
- L6306W:** '<attr1>' section <objname>(<secname>) should not use the address of '<attr2>' function <sym>.
- 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: Name <section> has been truncated.

In ADS 1.1 (and earlier), there is a 31 character maximum for referencing section names in a scatter-loading description file. Section names longer than 31 characters are shortened by the linker. You must either shorten the section name in the source code and scatter-loading description file, or where this is not possible, refer to the section from the scatter-loading description file using * to shorten the name. e.g: <long_section_name_in_scatter_file> could be changed to: <long_section_name_in_scatter_f*>. Note that the * uses one of the available 31 characters.

L6311W: Undefined symbol <symbol> (referred from <objname>).

See **L6218E**.

L6312W: Empty Load/Execution region description for region <region>

L6313W: Using <oldname> as an section selector is obsolete. Please use <newname> instead.

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.

See also "Use of +RW and +ZI in scatter-loading" in ADS 1.2 Getting Started guide, section 4.2.3, "Linking".

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>.

L6319W: Ignoring <cmd> command. Cannot find section <objname>(<secname>).

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.

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 ADS1.2 Linker and Utilities guide, section 3.3.3, "Unused section elimination"

L6330W: Undefined symbol <symbol> (referred from <objname>). Unused section has been removed.

L6331W: No eligible global symbol matches pattern <pat>.

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. The ADS 1.2 Linker patch build 841 (and later) has this warning suppressed.

L6334W: Illegal alignment constraint (<align>) for <objname>(<secname>) ignored. Using 4 byte alignment.

An illegal alignment was specified for an AOF object (see L6287E).

L6335W: ARM interworking code in <objname>(<secname>) may contain invalid tailcalls to ARM non-interworking code.

The documentation on interworking can be found in Chapter 3 of the Developer Guide. This makes good background reading, but unless you are coding assembler routines that will be interworked, all you really need to do is use -apcs /interwork.

L6336W: Veneer out of reach of chosen ER <user> using source ER <source>

In the warning message, "ER" means "Execution Region". "Veneer" means the code automatically generated by the linker to enable branches of more than 32MB.

This warning can occur if you have attempted to place the veneer code in a specific position by the use of "(Veneer\$\$Code)" in the scatterfile. armlink will place the veneers into the region specified region, but only if it is safe to do so. It might not be possible for the veneers to be assigned to the region because of address range problems or execution region size limitations. If

the veneer cannot be added to the specified region, it is added to the execution region containing the relocated input section that generated the veneer. The warning from the linker is simply informing you that it could not place the veneers as specified, and has chosen a better location itself. Normally, you can simply remove the `*(Veneer$$Code)` from the scatter file. See the ADS1.2 Linker and Utilities guide, section 5.3.1, "Selecting veneer input sections in scatter-loading descriptions" (Section 5.10.4 in ADS1.1).

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.

L6340W: options first and last are ignored for link type of `-partial`
The `-first` and `-last` options are meaningless when creating a partially-linked object

L6559I: Changing previous definition of symbol `<symname>` (from `<objname>`) to `<newsymname>`.

L6565I: Not eliminating unused sections as image is unsuitable for such optimization.
Unused section elimination cannot be performed on this image.

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>`.

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 SDT to create a new AOF object, or preferably, recompile the source file with ADS to create a new ELF object.

5. ELF Format Converter (fromelf) Errors and Warnings

Q0105E: Base and/or size too big for this format, max = 0x<maxval>.

This is most likely a misguided use of the "-ihf" switch, which has a limit of 0x3ffff. IHF format is deprecated and will be removed in a future release. "-ihf" is NOT Intel Hex Format - use "-i32" instead.

Q0106E: Out of Memory.

Q0107E: Failed writing output file.

Q0108E: Could not create output file '<filename>'.

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 in ADS 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>'.

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 ADS, object (.o) files and executable (.axf) files are in ELF format. If you are using CodeWarrior, you may have fromelf utility selected as a post-linker with an invalid file format as the input.

A known ADS 1.1 linker fault would sometimes result in corrupted ELF files. This fault was fixed in the ADS 1.1 linker patch, which can be downloaded from

<http://www.arm.com/support/downloads>

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.

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>.
Q0148E: Failed to change to directory <dir>.
Q0149E: Failed to change to directory <dir>.
The directory could not be accessed - check your access permissions.

Q0158W: Cannot use filename as argument '<file>'.

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.
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 section <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.

Q0173E: Failed to close temporary file

Q0174E: Failed to delete temporary file
See Q0175E.

This error message can also occur when fromelf output file is used directly as input for some other non-ADS application, e.g. textpad or an EPROM programmer, which locks the file open when reading/processing it. A workaround for this problem has been added into fromelf in the ADS 1.2 patch build 842, which you can download from:
<http://www.arm.com/support/Downloads>

Q0175E: Failed to rename temporary file

This can be given by fromelf as a result of e.g:
fromelf -bin -o out.bin in.axf

During its operation, fromelf opens a temporary file and writes to that, instead of writing directly to out.bin. If the translation fails, fromelf will delete the temporary file. If the translation succeeds, fromelf will delete the original file if it exists. If the file deletion fails, it will give the error message "Error trying to delete temporary file". If the file deletion succeeds, it will then rename the temporary file to the original <filename>. If the renaming step fails, it will give the message "Error trying to rename temporary file".

Check whether the temporary file is still in the directory. If it is, then fromelf has done the work - you should be able to simply rename/copy the temporary file to recover the completed output.

If you are using ADS 1.1 (or earlier), try using _no_ file extension for the output file, e.g.:
fromelf -bin -o out in.axf.

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 <file>

Q0186E: This option requires dwarf2 debugging information to be present

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.

Q0192E: Warning IHF format is deprecated and will be removed in a future release
See Q0105E

Q0193E: Could not save output file <filename>
Is given if a failure is detected when deleting a file

Q0194E: Could not save output file <filename>
Is given if a failure is detected when renaming a file

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

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 contains non ELF Object <name>

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.

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: Archive <name> is not an ELF Object Library

ADS 1.1 (and earlier) armar was not able to merge libraries into another library. Attempting to join, for example, lib1.a and lib2.a into mixedlibs.a resulted in a warning:

```
L6875W: Archive mixedlib.a is not an ELF Object Library
```

ADS 1.2 armar is now able to merge libraries into another library.