ARM® Compiler

Version 6.01

Errors and Warnings Reference Guide

ARM®

ARM DUI 0807B (ID121814)

ARM Compiler
Errors and Warnings Reference Guide

Release Information

Change history

Description	Issue	Confidentiality	Change
14 March 2014	A	Non-Confidential	ARM Compiler v6.00 Release
15 December 2014	B	Non-Confidential	ARM Compiler v6.01 Release

Proprietary Notice

Confidentiality Status

Product Status

The information in this document is final, that is for a developed product.

Web Address

`http://www.arm.com`

Contents

ARM Compiler Errors and Warnings Reference Guide

Chapter 1
Conventions and Feedback

The following describes the typographical conventions and how to give feedback:

Typographical conventions

The following typographical conventions are used:

`monospace` Denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.

`mono`space Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.

`monospace italic`

Denotes arguments to commands and functions where the argument is to be replaced by a specific value.

`monospace bold`

Denotes language keywords when used outside example code.

italic Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.

bold Highlights interface elements, such as menu names. Also used for emphasis in descriptive lists, where appropriate, and for ARM® processor signal names.

Feedback on this product

If you have any comments and suggestions about this product, contact your supplier and give:

- Your name and company.

- The serial number of the product.
- Details of the release you are using.
- Details of the platform you are using, such as the hardware platform, operating system type and version.
- A small standalone sample of code that reproduces the problem.
- A clear explanation of what you expected to happen, and what actually happened.
- The commands you used, including any command-line options.
- Sample output illustrating the problem.
- The version string of the tools, including the version number and build numbers.

Feedback on content

If you have comments on content then send an e-mail to `errata@arm.com`. Give:

- The title.
- The number, ARM DUI 0807B.
- If viewing online, the topic names to which your comments apply.
- If viewing a PDF version of a document, the page numbers to which your comments apply.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

ARM periodically provides updates and corrections to its documentation on the ARM Information Center, together with knowledge articles and *Frequently Asked Questions* (FAQs).

Other information

- ARM Information Center `http://infocenter.arm.com/help/index.jsp`.
- ARM Technical Support Knowledge Articles `http://infocenter.arm.com/help/topic/com.arm.doc.faqs/index.html`.
- ARM Support and Maintenance `http://www.arm.com/support/services/support-maintenance.php`.
- ARM Glossary `http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html`.

Chapter 2
armasm Errors and Warnings

The error and warning messages for the assembler, `armasm`, are listed in the following topic:

- *List of the armasm error and warning messages* on page 2-2.

2.1 List of the armasm error and warning messages

The error and warning messages for `armasm` are:

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 require the offset of a label called `<label>` within an area called `<areaname>`, use `<label> - <areaname>`.

See the following in the *armasm User Guide*:

- *Unary operators* on page 10-21.

A1020E `Bad predefine: <directive>`

The operand to the `--predefine` or `--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 might require escaping depending on the operating system and shell. For example:

```
--predefine "versionstr SETS \"5A\""
```

A1021U `No input file`

No input file was specified on the command line. This might be because there was no terminating quote on a quoted argument.

A1042E `Unrecognized APCS qualifier '<qualifier>'`

There is an error in the argument given to the `--apcs` command-line option. Check the spelling of `<qualifier>`.

A1056E `Target cpu '<cpu>' not recognized`

The name given in the `--cpu` command-line option is not a recognized processor name. Check the spelling of the argument.

Use `--cpu=list` to list the supported processors and architectures.

A1067E `Output file specified as '<filename1>', but it has already been specified as '<filename2>'`

More than one output file, `-o filename`, has been specified on the command line. Misspelling a command-line option can cause this.

A1071E `Cannot open listing file '<filename>': <reason>`

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 might 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 might be because the object filename was accidentally omitted from the command line.

A1074E `The specified depend file '<filename>' must not be a source file`

The filename argument to the `--depend` command-line option has an extension that indicates it is a source (`.s`) file. This might be because the filename argument was accidentally omitted from the command line. Check that the correct arguments are given.

A1075E `The specified errors file '<filename>' must not be a source file`

The filename argument to the `--errors` command-line option has an extension that indicates it is a source (`.s`) file. This might be because the filename argument was accidentally omitted from the command line. Check that the correct arguments are given.

A1085W `Forced user-mode LDM/STM must not be followed by use of banked R8-R14`

The ARM architecture does not permit you to access the banked registers in the instruction following a User registers `LDM` or `STM`. Adding a `NOP` immediately after the `LDM` or `STM` is one way to avoid this.

For 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 reported when no `AREA` directive is specified (see A1105E).

A1105E `Area directive missing`

This is reported when no `AREA` directive is specified (see also A1088W).

A1110E `Expected constant expression`

A constant expression was expected after, for example, `SETA`.

See the following in the *armasm User Guide*:

- *Numeric expressions* on page 10-16.

A1113E `Expected string expression`

A string expression was expected after, for example, `SETS`.

See the following in the *armasm User Guide*:

- *String expressions* on page 10-14.

A1119E `MEND not allowed within conditionals`

`MEND` means *END of Macro* (not the English word *mend*).

See the following in the *armasm User Guide*:

- *Use of macros* on page 7-31.

A1135E `Area name missing`

`AREA` names starting with any non-alphabetic character must be enclosed in bars, for example change:

```
AREA 1_DataArea, CODE, READONLY
```

to:

```
AREA |1_DataArea|, CODE, READONLY
```

A1137E `Unexpected characters at end of line`

This is given when extra characters that are not part of an instruction are found on an instruction line.

For example:

```
ADD r0, r0, r1 comment
```

You can change this to:

```
ADD r0, r0, r1 ; comment
```

A1142E `Subtractive relocations not supported for <entity> format output`

This can occur when subtracting symbols that are in different areas, for example:

```
IMPORT sym1
IMPORT sym2
DCD (sym2 - sym1)
```

A1150E `Bad symbol, not defined or external`

This typically occurs in the following cases:

- When the current file requires an `INCLUDE` of another file 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, for example:

  ```
  INCLUDE targets/eb40.inc
  ```

- When the current file requires `IMPORT` for some symbols, 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, for example:

  ```
  IMPORT ||Image$$RAM$$ZI$$Limit||
  ```

A1151E `Bad register name symbol`

Example:

```
MCR p14, 3, R0, Cr1, Cr2
```

The coprocessor registers `CR` must be labeled as a lowercase `c` for the code to build. The ARM register can be `r` or `R`:

```
MCR p14, 3, r0, c1, c2
```

or

```
MCR p14, 3, R0, c1, c2
```

A1158E `Illegal line start, should be blank`

Some directives, for example, `ENTRY`, `IMPORT`, `EXPORT`, and `GET` must be on a line without a label at the start of the line. This error is given if a label is present.

A1159E `Label missing from line start`

Some directives, for example, `FUNCTION` or `SETS`, require a label at the start of the line, for example:

```
my_func FUNCTION
```

or

```
label SETS
```

This error is given if the label is missing.

A1160E `Bad local label number`

A numeric local label is a number in the range 0-99, optionally followed by a name.

See the following in the *armasm User Guide*:

- *Numeric local labels* on page 10-12.

A1163E `Unknown opcode <name> , expecting opcode or Macro`

The most common reasons for this are:

- 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
  ```
- Mistyping the opcode:
  ```
   ADDD
  ```
  instead of
  ```
   ADD
  ```

A1164E `Opcode not supported on selected processor`

The processor selected on the `armasm` command line does not support this instruction. See the *ARM Architecture Reference Manuals* http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/index.html#reference.

A1170E `Immediate 0x<adr> out of range for this operation, must be below (0x<adr>)`

This error is given when `DCB`, `DCW` or `DCWU` directives are used with immediates that are too large.

See the following in the *armasm Reference Guide*:

- *DCB* on page 10-23.
- *DCW and DCWU* on page 10-31.

A1173E `ADR/L cannot be used on external symbols`

The `ADR` and `ADRL` pseudo-instructions can only be used with labels within the same code area. To load an out-of-area address into a register, use `LDR` instead.

A1176E `Branch offset 0x<val> out of range.  Permitted values are 0x<mini> to 0x<maxi>`

Branches are PC-relative, and have a limited range. If you are using numeric local labels, you can use the `ROUT` directive to limit their scope. This helps to avoid referring to the wrong label by accident.

See the following in the *armasm User Guide*:

- *Numeric local labels* on page 10-12.

A1186E `Code generated in data area`

An instruction has been assembled into a data area. This can happen if you have omitted the `CODE` attribute on the `AREA` directive.

See the following in the *armasm Reference Guide*:

- *AREA* on page 10-13.

A1198E `Unknown operand`

This can occur when an operand is accidentally mistyped.

For example:

```
armasm --cpu=8-A.64 init.s -g -PD "ROM_RAM_REMAP SETL {FALS}"
```

must be:

```
armasm --cpu=8-A.64 init.s -g -PD "ROM_RAM_REMAP SETL {FALSE}"
```

See the following in the *armasm User Guide*:

- *Assembly time substitution of variables* on page 10-6.

A1202E `No pre-declaration of substituted symbol '<name>'`

See the following in the *armasm User Guide*:

- *Assembly time substitution of variables* on page 10-6.

A1206E `Registers should be listed in increasing register number order`

This warning is given if registers in, for example, `LDM` or `STM` instructions are not specified in increasing order and the `--checkreglist` option is used.

A1207E `Bad or unknown attribute`

This error is given when an invalid attribute is given in the `AREA` directive. For example:

```
AREA test,CODE,READONLY,HALFWORD
```

`HALFWORD` is invalid, so remove it.

See the following in the *armasm Reference Guide*:

- *AREA* on page 10-13.

A1219E `Bad APSR, CPSR or SPSR designator`

For example:

```
 MRS r0, PSR
```

You must specify whether to use the CPSR or SPSR, for example:

```
 MRS r0, CPSR
```

A1247E `BLX from ARM code to ARM code, use BL`

This occurs when there is a `BLX` *`label`* branch from A32 code to A32 code within this assembly file. This is not permitted because `BLX` *`label`* always results in a change of instruction set state. The usual solution is to use `BL` instead.

A1248E `BLX from Thumb code to Thumb code, use BL`

This occurs when there is a `BLX` *`label`* branch from T32 code to T32 code within this assembly file. This is not permitted because `BLX` *`label`* always results in a change of instruction set state. The usual solution is to use `BL` instead.

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.

A1261E `MRS cannot select fields, use APSR, CPSR or SPSR directly`

This is caused by an attempt to use fields for CPSR or SPSR with an `MRS` instruction, such as:

```
MRS r0, CPSR_c
```

A1283E `Literal pool too distant, use LTORG to assemble it within 1KB`

For T32 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 A32 code, the literal pool must be within 4KB of the LDR instruction to access it. To solve this, add an LTORG directive into your assembly source file at a convenient place.

See the following in the *armasm User Guide*:

- *Load addresses to a register using LDR Rd, =label* on page 7-18.

See the following in the *armasm Reference Guide*:

- *LTORG* on page 10-65.

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.

A1322E `Unaligned transfer of PC, destination address must be 4 byte aligned, otherwise result is UNPREDICTABLE`

This error is reported when you try to use an LDR instruction to load the PC from a non word-aligned address. This gives an UNPREDICTABLE result. For example:

```
AREA Example, CODE
LDR pc, [pc, #6] ; Error - offset must be a multiple of 4
END
```

A1327E `Non portable instruction (LDM with writeback and base in register list, final value of base unpredictable)`

In the LDM instruction, if the base register <Rn> is specified in <registers>, and base register writeback is specified, the final value of <Rn> is UNKNOWN.

A1328E `Non portable instruction (STM with writeback and base not first in register list, stored value of base unpredictable)`

In the STM instruction, if <Rn> is specified in <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 UNKNOWN.

A1329E `Unpredictable instruction (forced user mode transfer with write-back to base)`

This is caused by an instruction such as PUSH {r0}^ where the ^ indicates access to user registers. Writeback to the base register is not available with this instruction.

Instead, the base register must be updated separately. For example:

```
SUB sp, sp,#4
STMID sp, {r0}^
```

Another example is replacing STMFD R0!, {r13, r14}^ with:

```
SUB r0, r0,#8
STM r0, {r13, r14}^
```

See also A1085W.

A1355U `A Label was found which was in no AREA`

This can occur if no white space precedes an assembler directive.

Assembler directives must be indented. For example use:

```
    IF :DEF: FOO
    ; code
    ENDIF
```

instead of:

```
IF :DEF: FOO
 ; code
ENDIF
```

Symbols in the left-hand column are assumed to be labels.

A1356E `Instruction not supported on targeted CPU`

This occurs if you try to use an instruction that is not supported by the specified architecture or processor for `armasm`.

See the *ARM Architecture Reference Manual* http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/index.html#reference.

A1429E `Expected register list`

The assembler reports this when `FRAME SAVE` and `FRAME RESTORE` directives are not given register lists.

See the following in the *armasm Reference Guide*:

- *FRAME RESTORE* on page 10-45.
- *FRAME SAVE* on page 10-47.

A1431E `Frame directives are not accepted outside of PROCs/FUNCTIONs`

See the following in the *armasm User Guide*:

- *Frame directives* on page 7-38.

A1433E `Only the writeback form of this instruction exists`

The addressing mode specified for the instruction did not include the writeback specifier (that is, 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 required behavior.

A1435E `{PCSTOREOFFSET} is not defined when assembling for an architecture`

`{PCSTOREOFFSET}` is only defined when assembling for a processor, not for an architecture.

A1437E `{ARCHITECTURE} is undefined`

`{ARCHITECTURE}` is only defined when assembling for an architecture, not for a processor.

A1446E `Bad or unknown attribute '<attr>'. Use --apcs /interwork instead`

Example:

```
 AREA test1, CODE, READONLY
 AREA test, CODE, READONLY, INTERWORK
```

This code might have originally been intended to work with the legacy ARM Software Development Toolkit (SDT). The `INTERWORK` area attribute is obsolete. To eliminate the warning:

- Remove the `", INTERWORK"` from the `AREA` line.
- Assemble with `--apcs /interwork` instead

A1447W `Missing END directive at end of file, but found a label named END`

This is caused by the END directive not being indented.

A1450E `Deprecated form of PSR field specifier used (use _cxsf for future compatibility)`

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.

Earlier releases of the assembler permitted 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.

Similar control and flag settings apply for SPSR.

These forms are deprecated and must not be used. If your legacy code contains them, the assembler reports:

`Deprecated form of PSR field specifier used (use _cxsf)`

To avoid the warning, in most cases you can modify your code to use _c, _f, _cf or _cxsf instead.

See the following in the *armasm User Guide*:

- *Conditional execution in A32 code* on page 8-3.
- *Conditional execution in T32 code* on page 8-4.
- *General-purpose registers in AArch32 state* on page 4-6.
- *Access to the inline barrel shifter in AArch32 state* on page 4-18.

See also the FAQ *armasm: use of MRS and MSR instructions ('Deprecated form of PSR field specifier')*
http://infocenter.arm.com/help/topic/com.arm.doc.faqs/ka3724.html.

A1454E `FRAME STATE RESTORE directive without a corresponding FRAME STATE REMEMBER`

See the following in the *armasm User Guide*:

- *Frame directives* on page 7-38.

See the following in the *armasm Reference Guide*:

- *FRAME STATE REMEMBER* on page 10-48.
- *FRAME STATE RESTORE* on page 10-49.

A1456W `INTERWORK area directive is obsolete. Continuing as if --apcs /inter selected`

Example:

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

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

1. Remove the ", INTERWORK" from the AREA line.
2. Assemble with --apcs /interwork instead.

A1457E `Cannot mix INTERWORK and NOINTERWORK code areas in same file`

INTERWORK and (default) NOINTERWORK code areas cannot be mixed in the same file. This code might have originally been intended to work with SDT. The INTERWORK area attribute is obsolete in the ARM Compiler toolchain.

Example:

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

To eliminate the error:

1. Move the two AREAs into separate assembly files such as, for example, test1.s and test2.s.
2. Remove the ", INTERWORK" from the AREA line in test2.s.
3. Assemble test1.s with --apcs /nointerwork.
4. Assemble test2.s with --apcs /interwork.
5. At link time, the linker adds any necessary interworking veneers.

A1459E Cannot B or BL to a register

This form of the instruction is not permitted. See the *ARM Architecture Reference Manual* for the permitted forms.

A1461E Specified processor or architecture does not support Thumb instructions

It is likely that you are specifying an architecture or processor that does not support T32 code using the --cpu option and then incorporating T32 code in the AREA that is generating this error.

A1466W Operator precedence means that expression would evaluate differently in C

armasm has always evaluated certain expressions in a different order to C. This warning might help C programmers from being caught out when writing in assembly language.

To avoid the warning, either:

- Modify the code to make the evaluation order explicit (that is, add more brackets).
- Suppress the warning with --unsafe switch.

See the following in the *armasm User Guide*:

- *Operator precedence* on page 10-29.

A1469E FRAME STATE REMEMBER directive without a corresponding FRAME STATE RESTORE

See the following in the *armasm User Guide*:

- *Frame directives* on page 7-38.

See the following in the *armasm Reference Guide*:

- *FRAME STATE REMEMBER* on page 10-48.
- *FRAME STATE RESTORE* on page 10-49.

A1471W Directive <directive> may be in an executable position

This can occur with, for example, the LTORG directive (see A1283E and A1284E). LTORG instructs the assembler to dump literal pool DCD data at this position.

To prevent this warning from occurring, the data must be placed where the processor cannot execute them as instructions. 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
```

A1477E `This register combination results in UNPREDICTABLE behavior`

This error is generated when you are assembling an instruction that has UNPREDICTABLE results on execution. You must rewrite your code to avoid this UNPREDICTABLE behavior. For example, the following instructions all cause this error when assembling to T32, and the target architecture is ARMv6T2 or later:

```
ADD sp, r0, #100 ; error - UNPREDICTABLE use of SP
CMP pc, #1 ; error - UNPREDICTABLE use of PC
PUSH    {r0, pc} ; error - use of an UNPREDICTABLE register combination
```

A1479W `Requested alignment <alignreq> is greater than area alignment <align>, which has been increased`

This is warning about an ALIGN directive that has a coarser alignment boundary than its containing AREA. This is not permitted. 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 is on a 16-byte boundary, causing the error.

Note

The two alignment types are specified in different ways.

This warning can also occur when using `AREA ... ALIGN=0` to align a code section on a byte boundary. This is not possible. Code sections can only be aligned on:

- A four-byte boundary for A32 code, so use "ALIGN=2".
- A two-byte boundary for T32 code, so use "ALIGN=1".

See the following in the *armasm Reference Guide*:

- *ALIGN* on page 10-11.
- *AREA* on page 10-13.

A1484W `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 shift types:

- ROR.
- ASR.
- LSR.
- LSL.

An arithmetic (that is, signed) shift left is the same as a logical shift left, because the sign bit always gets shifted out.

If the name ASL is used, the assembler reports this warning and converts the ASL to LSL.

See the following in the *armasm Reference Guide*:

- *--unsafe* on page 2-67.

- *ASR, LSL, LSR, ROR, and RRX* on page 3-41.

A1486E `ADR/ADRL of a symbol in another AREA is not supported in ELF`

The `ADR` and `ADRL` pseudo-instructions can only be used with labels within the same code section. To load an out-of-area address into a register, use `LDR` instead.

A1487W `Obsolete instruction name 'ASL', use LSL instead`

This warning is given when the `ASL` instruction is used in pre-UAL T32 code, that is, when you assemble using the `--16` command-line option, or you use the `CODE16` directive. See the corresponding A32 `ASL` message `A1484W`.

A1490E `<CPU> is undefined`

{`CPU`} is only defined by assembling for a processor and not an architecture.

A1491W `Internal error: Found relocation at offset <offset> with incorrect alignment`

This might indicate an assembler fault. Contact your supplier.

A1495W `Target of branch is a data address`

`armasm` determines the type of a symbol and detects branches to data. Specify `--diag-suppress 1495` to suppress this warning.

A1496W `Absolute relocation of ROPI address with respect to symbol '<symbol>' at offset <offset> may cause link failure`

This warning relates to a feature that is unsupported in ARM Compiler 6.0.

A1497W `Absolute relocation of RWPI address with respect to symbol '<symbol>' at offset <offset> may cause link failure`

This warning relates to a feature that is unsupported in ARM Compiler 6.0.

A1498E `Unexpected characters following Thumb instruction`

For example:

```
ADD r0, r0, r1
```

is accepted as a valid instruction, for both A32 and T32, but:

```
ADD r0, r0, r1, ASR #1
```

is a valid instruction for A32, but not for T32. The unexpected characters are `", ASR #1"`.

A1546W `Stack pointer update potentially breaks 8 byte stack alignment`

Example:

```
PUSH {r0}
```

The stack must be eight-byte aligned on an external boundary so pushing an odd number of registers causes this warning. This warning is suppressed by default. To enable this warning use `--diag_warning 1546`.

See the following in the *armasm Reference Guide*:

- *--diag_warning* on page 2-24.

A1547W `PRESERVE8 directive has automatically been set`

Example:

```
PUSH {r0,r1}
```

This warning is given because you have not explicitly set the `PRESERVE8` directive, but the assembler has set it automatically. This warning is suppressed by default. To enable this warning use `--diag_warning 1547`.

See the following in the *armasm Reference Guide*:

- *--diag_warning* on page 2-24.
- *REQUIRE8 and PRESERVE8* on page 10-78.

A1548W `Code contains LDRD/STRD indexed/offset from SP but REQUIRE8 is not set`

Example:

```
PRESERVE8
STRD r0,[sp,#8]
```

This warning is given when the `REQUIRE8` directive is not set when required.

See the following in the *armasm Reference Guide*:

- *REQUIRE8 and PRESERVE8* on page 10-78.

A1549W `Setting of REQUIRE8 but not PRESERVE8 is unusual`

Example:

```
PRESERVE8 {FALSE}
REQUIRE8
STRD r0,[sp,#8]
```

A1563W `Instruction stalls CPU for <stalls> cycle(s)`

The assembler can give information about possible interlocks in your code caused by the pipeline of the processor chosen by the `--cpu` option. To do this assemble with `armasm --diag_warning 1563`

Note

If the `--cpu` option specifies a multi-issue processor such as Cortex®-A8, the interlock warnings are unreliable.

See also warning `A1746W`.

A1581W `Added <no_padbytes> bytes of padding at address <address>`

The assembler warns by default when padding bytes are added to the generated code. This occurs whenever an instruction or directive is used at an address that requires a higher alignment, for example, to ensure A32 instructions start on a four-byte boundary after some T32 instructions, or where there is a `DCB` followed by `DCD`.

For example:

```
 AREA Test, CODE, READONLY
 THUMB
T32Code
  MOVS r0, #1
  ADR r1, A32Prog
  BX r1
; ALIGN ; <<< add to avoid the first warning
 ARM
A32Prog
 ADD r0,r0,#1
 BX LR
 DCB 0xFF
 DCD 0x1234
 END
```

Results in the warnings:

```
A1581W: Added 2 bytes of padding at address 0x6
8 00000008 ARM
A1581W: Added 3 bytes of padding at address 0x11
```

```
13 00000014 DCD 0x1234
```

The warning can also occur when using ADR in T32-only code. The ADR T32 pseudo-instruction can only load addresses that are word aligned, but a label within T32 code might not be word aligned. Use ALIGN to ensure four-byte alignment of an address within T32 code.

See the following in the *armasm Reference Guide*:

- *ADR (PC-relative)* on page 3-32.
- *ADR (register-relative)* on page 3-34.
- *DCB* on page 10-23.
- *DCD and DCDU* on page 10-24.
- *ALIGN* on page 10-11.

A1609W `MOV <rd>,pc instruction does not set bit zero, so does not create a return address`

This warning is caused when the current value of the PC is copied into a register while executing in T32 state. An attempt to create a return address in this way fails because bit[0] is not set. Attempting a BX branch to this instruction causes a state change (to A32).

To create a return address, you can use:

```
MOV r0, pc
ADDS r0, #1
```

This warning can then be safely suppressed with:

```
--diag-suppress 1609
```

A1616E `Instruction, offset, immediate or register combination is not supported by the current instruction set`

This can be caused by attempting to use an invalid combination of operands. For example, in T32 code:

```
MOV r0, #1 ; /* Not permitted */
MOVS r0, #1 ; /* Ok */
```

See the following in the *armasm Reference Guide*:

- Chapter 3 *A32 and T32 Instructions*.

A1627W `BL from ARM code to Thumb code`

This occurs when there is a branch from A32 code to T32 code, or from T32 to A32 within this file. The usual solution is to move the T32 code into a separate assembly file. Then, at link-time, the linker adds any necessary interworking veneers.

A1630E `Specified processor or architecture does not support ARM instructions`

ARM M-profile processors, for example Cortex-M3 and Cortex-M1, implement only the T32 instruction set, not the A32 instruction set. It is likely that the assembly file contains some A32-specific instructions and is being built for one of these processors.

A1645W `Substituted <old> with <new>`

armasm can warn when it substitutes an instruction when assembling.

For example:

- ADD *negative_number* is the same as SUB *positive_number*.
- MOV *negative_number* is the same as MVN *positive_number*.
- CMP *negative_number* is the same as CMN *positive_number*.

For the T32 instruction set, UNPREDICTABLE single register LDMs are transformed into LDRs.

This warning is suppressed by default, but can be enabled with `--diag_warning 1645`

For example:

```
AREA foo, CODE
ADD r0, #-1
MOV r0, #-1
CMP r0, #-1
```

When assembled with:

```
armasm --diag_warning 1645
```

the assembler reports:

```
Warning: A1645W: Substituted ADD with SUB
3 00000000 ADD r0, #-1
Warning: A1645W: Substituted MOV with MVN
4 00000004 MOV r0, #-1
Warning: A1645W: Substituted CMP with CMN
5 00000008 CMP r0, #-1
```

and the resulting code generated is:

```
foo
0x00000000: e2400001 ..@. SUB r0,r0,#1
0x00000004: e3e00000 .... MVN r0,#0
0x00000008: e3700001 ..p. CMN r0,#1
```

A1647E `Bad register name symbol, expected Integer register`

An integer (core) register is expected at this point in the syntax.

A1648E `Bad register name symbol, expected Wireless MMX SIMD register`

This message relates to Wireless MMX.

A1649E `Bad register name symbol, expected Wireless MMX Status/Control or General Purpose register`

This message relates to Wireless MMX.

A1650E `Bad register name symbol, expected any Wireless MMX register`

This message relates to Wireless MMX.

A1651E `TANDC, TEXTRC and TORC instructions with destination register other than R15 is undefined`

This message relates to Wireless MMX.

A1658W `Support for <opt> is deprecated`

The option passed to `armasm` is deprecated. Use `armasm --help` to view a summary of the available options.

See the following in the *armasm Reference Guide*:

- *armasm command-line options* on page 2-3.

A1694E `Instruction cannot be conditional in the current instruction set`

Conditional instructions are not permitted in the specified instruction set. The instruction `MOVEQ`, for example, is permitted in A32 code, and in T32 code in architectures in which the `IT` instruction is available.

A1745W `This register combination is DEPRECATED and may not work in future architecture revisions`

This warning is generated when all of the following conditions are satisfied:

- You are using a deprecated register combination, for example:
  ```
  PUSH    {r0, pc}
  ```
- You are assembling for a target architecture that supports 32-bit T32 instructions, in other words ARMv6T2 or later.
- You are assembling to A32 code.

Note

- When assembling to T32, rather than A32 code, and the target architecture is ARMv6T2 or later, the assembler generates error A1477E instead.
- When assembling for an architecture or processor that does not support 32-bit T32 instructions, in other words ARM architectures before ARMv6T2, by default no diagnostic is emitted.

A1746W `Instruction stall diagnostics may be unreliable for this CPU`

This warning is shown when you enable message A1563W for a processor that is not modeled accurately by the assembler. It indicates that you cannot rely on the output of A1563W when improving your code.

See also warning `A1563W`.

A1762E `Branch offset 0x<val> out of range of 16-bit Thumb branch, but offset encodable in 32-bit Thumb branch`

This is caused when assembling for T32 if an offset to a branch instruction is too large to fit in a 16-bit branch. The `.W` suffix can be added to the instruction to instruct the assembler to generate a 32-bit branch.

A1763W `Inserted an IT block for this instruction`

This indicates that the assembler has inserted an IT block to permit a number of conditional instructions in T32 code. For example:

```
MOVEQ r0,r1
```

This warning is off by default. It can be enabled using `--diag_warning A1763`.

A1764W `<name> instructions are deprecated in architecture <arch> and above`

A1765E `Size of padding value on ALIGN must be 1, 2 or 4 bytes`

This is caused when the optional `padsize` attribute is used with an `ALIGN` directive, but has an incorrect size. It does not refer to the parameter to align to. The parameter can be any power of 2 from 2^0 to 2^31

A1786W `This instruction using SP is deprecated and may not work in future architecture revisions`

This warning is generated when all of the following conditions are satisfied:

- You explicitly use the SP in a deprecated way, for example:
  ```
  ADD sp, r0, #100
  ```
- You are assembling for a target architecture that supports 32-bit T32 instructions, in other words ARMv6T2 or later.
- You are assembling to A32 code.

ARM deprecates the explicit use of the SP in A32 instructions in any way that is not possible in the corresponding T32 instruction. Such deprecated register uses are still possible in A32 instructions for backwards compatibility and you can

suppress this warning by using the assembler command-line option `--diag_suppress=1786`. However, ARM recommends you modify your code, because it might not work in future architecture revisions.

You can replace the deprecated use of the SP shown in the example with a sequence like:

```
ADD r1, r0, #100
MOV sp, r1
```

Note

- When assembling to T32, rather than A32 code, and the target architecture is ARMv6T2 or later, the assembler generates error A1477E instead.
- When assembling for an architecture or processor that does not support 32-bit T32 instructions, in other words ARM architectures before ARMv6T2, by default no diagnostic is emitted.

A1788W `Explicit use of PC in this instruction is deprecated and may not work in future architecture revisions`

This warning is generated when all of the following conditions are met:

- You explicitly use the PC in a deprecated way, for example:
  `CMP pc, #1`
- You are assembling for a target architecture that supports 32-bit T32 instructions, in other words ARMv6T2 or later.
- You are assembling to A32 code.

ARM deprecates most explicit uses of the PC in A32 instructions, although they are still possible for backwards compatibility. You can suppress this warning by using the assembler command-line option `--diag_suppress=1788`. However, ARM recommends you modify your code, because it might not work in future architecture revisions.

Note

- When assembling to T32 rather than A32 code, and the target architecture is ARMv6T2 or later, the assembler generates error A1477E instead.
- When assembling for an architecture or processor that does not support 32-bit T32 instructions, in other words ARM architectures before ARMv6T2, by default no diagnostic is emitted.

A1790W `Writeback ignored in Thumb LDM loading the base register`

This is caused by incorrectly adding an exclamation mark to indicate base register writeback.

For example:

```
LDM r0!, {r0-r4}
```

is not a legal instruction because r0 is the base register and is also in the destination register list. In this case, the assembler ignores the writeback and generates:

```
LDM r0, {r0-r4}
```

A1809W `Instruction aligns PC before using it; section ought to be at least 4 byte aligned`

This warning is generated when all the following conditions apply:

- You are using a PC-relative offset in a T32 instruction that requires the PC to be word-aligned.
- The code section containing this instruction has less than 4-byte alignment.
- The instruction is not relocated at link time (because of a relocation emitted by the assembler).

If these conditions are all met, and the code section containing this instruction is not placed at a 4-byte aligned address when linking, the instruction might operate on or with the wrong address at runtime. This is because the instruction aligns the PC to a 4-byte address before using it.

The following example shows an `LDR` instruction in T32 that is diagnosed by this warning because the section has an alignment of 2 bytes:

```
AREA ||.text||, CODE, READONLY, ALIGN=1
THUMB
LDR r0, [pc, #8]  ; gives warning A1809W
```

A1847E `Expression requiring more than one relocation not allowed`

This can occur during the assembly of A32 instructions 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`.

Change your code to use the simpler, equivalent syntax:

```
LDR r0, label
```

This works if `label` is either in the same area or in a different area.

A1875E `Register Rn must be from R0 to R7 in this instruction`

Change the specified register to be in the range R0 to R7.

A1903E `Line not seen in first pass; cannot be assembled`

This occurs if an instruction or directive does not appear in pass 1 but appears in pass 2 of the assembler.

The following example shows when a line is not seen in pass 1:

```
    AREA x,CODE
    [ :DEF: foo
num EQU 42 ; assembler does not see this line during pass 1 because
           ; foo is not defined at this point during pass 1
    ]
foo DCD num
    END
```

A1907W `Test for this symbol has been seen and may cause failure in the second pass.`

This diagnostic is suppressed by default. Enable it to identify situations that might result in errors A1903E, A1909E, or A1908E.

A1908E `Label '<name>' value inconsistent: in pass 1 it was <val1>; in pass 2 it was <val2>`

The following example generates this error because in pass 1 the value of `x` is `0x0004+r9`, and in pass 2 the value of `x` is `0x0000+r0`:

```
    map 0, r0
    if :lnot: :def: sym
        map 0, r9
        field 4
    endif
x   field 4
sym LDR r0, x
```

A1909E `Line not seen in second pass; cannot be assembled`

This occurs if an instruction or directive appears in pass 1 but does not appear in pass 2 of the assembler.

The following example shows when a line is not seen in pass 2:

```
    AREA x,CODE
    [ :LNOT: :DEF: foo
    MOV r1, r2  ; assembler does not see this line during pass 2 because
                ; foo is already defined
    ]
foo MOV r3, r4
    END
```

A9511E `Product definition file was not found`

The assembler cannot find the required product license mapping (`.elmap`) files.

This might be because:

- The `.elmap` files are missing, for example if your installation is corrupt.
- The assembler is looking for the `.elmap` files in the wrong place because the `ARM_PRODUCT_PATH` environment variable is incorrectly set.
- The assembler is looking for a non-existant `.elmap` file because the `ARM_TOOL_VARIANT` environment variable is incorrectly set.

Chapter 3
Linker Errors and Warnings

The following topics describe the error and warning messages for the linker, `armlink`:

3.1 Suppressing armlink error and warning messages

All linker warnings are suppressible with `--diag_suppress`. For example:

```
--diag_suppress 6306
```

Some errors such as L6220E, L6238E, and L6784E can be downgraded to a warning by using:

```
--diag_warning
```

3.2 List of the armlink error and warning messages

The error and warning messages for `armlink` are:

L6000U `Out of memory.`

For details on why you might see this error, and possible solutions, see the description for error `L6815U`.

L6002U `Could not open file <filename>: <reason>`

This indicates that the linker was unable to open a file specified on the linker command line. This can indicate a problem accessing the file or a fault with the command line specified. Some common occurrences of this message are:

- `L6002U: Could not open file /armlib/{libname}: No such file or directory`

  Specify the library path with `--libpath`.

  See the following in the *armlink Reference Guide*:

  — *--libpath on page 2-81*.
- `Error : armlink : L6002: Could not open file errors=ver.txt`

  Caused by the double-dash (`--`) missing from in front of `errors=ver.txt`. If you do not prefix options with `--` or `-` the linker treats them as input files and fails the link step because it is unable to load all the specified files. The correct switch is `--errors=ver.txt`

L6003U `Could not write to file <filename>.`

A file I/O error occurred while reading, opening, or writing to the specified file.

L6004U `Incomplete library member list <list> for <library>.`

This can occur where there is whitespace in the list of library objects.

The following example fails:

```
armlink x.lib(foo.o, bar.o)
Fatal error: L6004U: Missing library member in member list for x.lib.
```

The following example succeeds:

```
armlink x.lib(foo.o,bar.o)
```

Another less common occurrence is caused by a corrupt library, or possibly a library in an unsupported format.

L6006U `Overalignment value not specified with OVERALIGN attribute for execution region <regionname>.`

See the following in the *armlink Reference Guide*:

- *Syntax of an input section description on page 4-24*

See the following in *armlink User Guide*:

- *Overalignment of execution regions and input sections on page 8-62*.

L6007U `Could not recognize the format of file <filename>.`

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

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

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

L6009U `File <filename> : Endianness mismatch.`

The endianness of the specified file or object did not match the endianness of the other input files. The linker can handle input of either big endian or little endian objects in a single link step, but not a mixed input of some big and some little endian objects.

L6010U `Could not reopen stderr to file <filename>: <reason>`

A file I/O error occurred while reading, opening, or 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 `&`, `0x`, or `0X`.

L6015U `Could not find any input files to link.`

The linker must be provided with at least one object file to link.

For example, if you try to link with:

```
armlink lib.a -o foo.axf
```

the linker reports this error.

You must instead use, for example:

```
armlink foo_1.o foo_2.o lib.a -o foo.axf
```

L6016U `Symbol table missing/corrupt in object/library <object>.`

This can occur when linking with libraries built with the GNU tools. This is because GNU `ar` can generate incompatible information.

The workaround is to replace `ar` with `armar` and use the same command line arguments. Alternatively, the error is recoverable by using `armar -s` to rebuild the symbol table.

L6017U `Library <library> symbol table contains an invalid entry, no member at offset 0x<offset>.`

The library might be corrupted. Try rebuilding it.

L6022U `Object <objname> has multiple <table>.`

The object file is faulty or corrupted. This might indicate a compiler fault. Contact your supplier.

L6024U `Library <library> contains an invalid member name.`

The file specified is not a valid library file, is faulty or corrupted. Try rebuilding it.

L6025U `Cannot extract members from a non-library file <library>.`

The file specified is not a valid library file, is faulty or corrupted. Try rebuilding it.

L6026U `ELF file <filename> has neither little or big endian encoding`

The ELF file is invalid. Try rebuilding it.

L6027U `Relocation #<rel_class>:<rel_number> in <objname>(<secname>) has invalid/unknown type.`

This might indicate a compiler fault. Contact your supplier.

L6028U `Relocation #<rel_class>:<rel_number> in <objname>(<secname>) has invalid offset.`

This might indicate a compiler fault. Contact your supplier.

L6029U `Relocation #<rel_class>:<rel_number> in <objname>(<secname>) is wrt invalid/missing symbol.`

The relocation is with respect to a symbol that is either:

- Invalid or missing from the object symbol table.
- A symbol that is not suited to be used by a relocation.

This might indicate a compiler fault. Contact your supplier.

L6030U `Overalignment <overalignment> for region <regname> must be at least 4 and a power of 2`

See the following in the *armlink Reference Guide*:

- *Execution region attributes on page 4-13*.
- *Syntax of an input section description on page 4-24*

See the following in *armlink User Guide*:

- *Overalignment of execution regions and input sections on page 8-62*.

L6031U `Could not open scatter description file <filename>: <reason>`

An I/O error occurred while trying to open the specified file. This could be because of an invalid filename.

L6034U `Symbol <symbolname> in <objname> has invalid value.`

This is most often caused by a section-relative symbol having a value that exceeds the section boundaries.

L6035U `Relocation #<rel_class>:<rel_number> in ZI Section <objname>(<secname>) has invalid type.`

ZI Sections cannot have relocations other than of type R_ARM_NONE.

L6036U `Could not close file <filename>: <reason>`

An I/O error occurred while closing the specified file.

L6037U `'<arg>' is not a valid argument for option '<option>'.`

The argument is not valid for this option. This could be because of a spelling error, or because of 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.

L6039W `Relocation from #<rel_class>:<rel_number> in <objname>(<secname>) with respect to <symname>. Skipping creation of R-type relocation. No corresponding R-type relocation exists for type <rel_type>.`

`--reloc` is used with objects containing relocations that do not have a corresponding R-type relocation.

L6041U `An internal error has occurred (<clue>).`

Contact your supplier.

L6042U `Relocation #<rel_class>:<rel_number> in <objname>(<secname>) is wrt a mapping symbol(#<idx>, Last Map Symbol = #<last>).`

Relocations with respect to mapping symbols are not permitted. This might indicate a compiler fault. Contact your supplier.

L6043U `Relocation #<rel_class>:<rel_number> in <objname>(<secname>) is with respect to an out of range symbol(#<val>, Range = 1-<max>).`

Relocations can only be made with respect to symbols in the range (1-n), where n is the number of symbols.

L6064E `ELF Executable file <filename> given as input on command line`

This might be because you specified an object file as output from the compiler without specifying the `-c` compiler option. For example:

```
armclang file.c -o file.o
```

```
armlink --force_scanlib file.o -o file.axf
```

L6065E `Load region <name> (size <size>) is larger than maximum writable contiguous block size of 0x80000000.`

The linker attempted to write a segment larger than 2GB. The size of a segment is limited to 2GB.

L6176E `A negative max_size cannot be used for region <regname> without the EMPTY attribute.`

Only regions with the EMPTY attribute are permitted to have a negative max-size.

L6177E `A negative max_size cannot be used for region <regname> which uses the +offset form of base address.`

Regions using the `+offset` form of base address are not permitted to have a negative max-size.

L6188E `Special section <sec1> multiply defined by <obj1> and <obj2>.`

A special section is one that can only be used once, such as `"Veneer$$Code"`.

L6195E `Cannot specify both '<attr1>' and '<attr2>' for region <regname>`

See the following in the *armlink Reference Guide*:

- *Load region attributes on page 4-8*.
- *Execution region attributes on page 4-13*.
- *Address attributes for load and execution regions on page 4-16*..
- *Inheritance rules for load region address attributes on page 4-20*
- *Inheritance rules for execution region address attributes on page 4-21*.
- *Inheritance rules for the RELOC address attribute on page 4-22*.

L6200E `Symbol <symbolname> multiply defined by <object1> and <object2>.`

A common example where this occurs:

```
Symbol __stdout multiply defined (by retarget.o and stdio.o).
```

means that there are two conflicting definitions of `__stdout` present in `retarget.o` and `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`, and `fflush`.

`stdio.o` is being linked-in because it satisfies some unresolved references.

To identify why `stdio.o` is being linked-in, you must use the `verbose` link option switch. For example:

```
armlink [... your normal options...] --verbose --list err.txt
```

Then study `err.txt` to see exactly what the linker is linking in, from where, and why.

You might have to either:

- Eliminate the calls like `fopen`, `fclose`, and `fflush`.

- Re-implement the _sys_xxxx family of functions.

See the following in the *ARM C and C++ Libraries and Floating-Point Support User Guide*:

- *Tailoring input/output functions in the C and C++ libraries* on page 2-88.

L6201E `Object <objname> contains multiple entry sections.`

The input object specifies more than one entry point. Use the `--entry` command-line option to select the entry point to use.

See the following in the *armlink Reference Guide*:

- *--entry on page 2-49*.

L6202E `<objname>(<secname>) cannot be assigned to non-root region '<regionname>'`

A root region is a region that has an execution address the same as its load address. The region does not therefore require moving or copying by the scatter-load initialization code.

Certain sections must be placed in a root region in the image, including:

- `__main.o`
- The linker-generated table (`Region$$Table`)
- Scatter-loading (`__scatter*.o`) objects from the library
- Decompressor (`__dc*.o`) objects from the library.

If a required section is not placed in a root region, the linker reports, for example:

```
anon$$obj.o(Region$$Table) cannot be assigned to a non-root region 'RAM'.
```

You can use `InRoot$$Sections` to include all required sections in a root region:

```
ROM_LOAD 0x0000 0x4000
{
  ROM_EXEC 0x0000 0x4000 ; root region
  {
    vectors.o (Vect, +FIRST) ; Vector table
    * (InRoot$$Sections) ; All library sections
                         ; that must be in a root region
                         ; for example, __main.o, __scatter*.o,
                         ; dc*.o and * Region$$Table
  }
  RAM 0x10000 0x8000
  {
    * (+RO, +RW, +ZI) ; all other sections
  }
}
```

L6203E `Entry point (<address>) lies within non-root region <regionname>.`

The image entry point must correspond to a valid instruction in a root-region of the image.

L6204E `Entry point (<address>) does not point to an instruction.`

The image entry point you specified with the `--entry` command-line option must correspond to a valid instruction in the root-region of the image.

See the following in the *armlink Reference Guide*:

- *--entry on page 2-49*.

L6205E `Entry point (<address>) must be word aligned for ARM instructions.`

This message is displayed because the image entry point you specified with the `--entry` command-line option is not word aligned. For example, you specified `--entry=0x8001` instead of `--entry=0x8000`.

See the following in the *armlink Reference Guide*:

- *--entry on page 2-49*.

L6206E `Entry point (<address>) lies outside the image.`

The image entry point you specified with the `--entry` command-line option is outside the image. For example, you might have specified an entry address of `0x80000` instead of `0x8000`, as follows:

```
armlink --entry=0x80000 test.o -o test.axf
```

See the following in the *armlink Reference Guide*:

- *--entry on page 2-49*.

L6208E `Invalid argument for --entry command: '<arg>'`

See the following in the *armlink Reference Guide*:

- *--entry on page 2-49*.

L6209E `Invalid offset constant specified for --entry (<arg>)`

See the following in the *armlink Reference Guide*:

- *--entry on page 2-49*.

L6210E `Image cannot have multiple entry points. (<address1>,<address2>)`

One or more input objects specifies more than one entry point for the image. Use the `--entry` command-line option to select the entry point to use.

See the following in the *armlink Reference Guide*:

- *--entry on page 2-49*.

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 must know which `AREA` you want to keep.

To solve this, use more than one `--keep` option to specify the names of the `AREA`s to keep, such as:

```
 --keep boot.o(vectors) --keep boot.o(resethandler) …
```

Note

Using assembler files with more than one `AREA` might give other problems elsewhere, so this is best avoided.

L6213E `Multiple First section <object2>(<section2>) not allowed. <object1>(<section1>) already exists.`

Only one `FIRST` section is permitted.

L6214E `Multiple Last section <object2>(<section2>) not allowed. <object1>(<section1>) already exists.`

Only one `LAST` section is permitted.

L6215E `Ambiguous symbol selection for --First/--Last. Symbol <symbol> has more than one definition.`

See the following in the *armlink Reference Guide*:

- *--first on page 2-56*.

- *--last on page 2-79*.

L6216E `Cannot use base/limit symbols for non-contiguous section <secname>`

The exception-handling index tables generated by the compiler are given the section name `.ARM.exidx`. For more information, see *Exception Handling ABI for the ARM Architecture* `http://infocenter.arm.com/help/topic/com.arm.doc.ihi0038-/index.html`.

At link time these tables must be placed in the same execution region and be contiguous. If you explicitly place these sections non-contiguously using specific selector patterns in your scatter file, then this error message is likely to occur. For example:

```
LOAD_ROM 0x00000000
{
  ER1 0x00000000
  {
    file1.o (+RO) ; from a C++ source
    * (+RO)
  }
  ER2 0x01000000
  {
    file2.o (+RO) ; from a C++ source
  }
  ER3 +0
  {
    * (+RW, +ZI)
  }
}
```

This might produce the following error if exception-handling index tables are in both `file1.o` and `file2.o`, because the linker cannot place them in separate regions:

```
Error: L6216E: Cannot use base/limit symbols for non-contiguous section
.ARM.exidx
```

Also, the `.init_array` sections must be placed contiguously within the same region for their base and limit symbols to be accessible.

The corrected example is:

```
LOAD_ROM 0x00000000
{
  ER1 0x00000000
  {
    file1.o (+RO) ; from a C++ source
    * (.ARM.exidx)  ; Section .ARM.exidx must be placed explicitly,
                    ; otherwise it is shared between two regions, and
                    ; the linker is unable to decide where to place it.
    *(.init_array)  ; Section .init_array must be placed explicitly,
                    ; otherwise it is shared between two regions, and
                    ; the linker is unable to decide where to place it.
    * (+RO)
  }
  ER2 0x01000000
  {
    file2.o (+RO) ; from a C++ source
  }
  ER3 +0
  {
    * (+RW, +ZI)
  }
}
```

In the corrected example, the base and limit symbols are contained in `.init_array` in a single region.

For more information, see the following in *ARM C and C++ Libraries and Floating-Point Support User Guide*:

- *How C and C++ programs use the library functions* on page 2-52.
- *C++ initialization, construction and destruction* on page 2-54.

L6217E `Relocation #<rel_class>:<rel_number> in <objname>(<secname>) with respect to <symbol>. R_ARM_SBREL32 relocation to imported symbol`

L6218E `Undefined symbol <symbol> (referred from <objname>).`

Some common examples where this can occur are:

- User Error. Somebody has referenced a symbol and has either forgotten to define it or has incorrectly defined it.
- `Undefined symbol __ARM_switch8 or __ARM_ll_<xxxx> functions`

  The helper functions are automatically generated into the object file by the compiler.

  Note

  An undefined reference error can, however, still be generated if linking objects from legacy projects where the helper functions are in the `h_`*xxx* libraries (`h` indicates that these are compiler helper libraries, rather than standard C library code).

  Re-compile the object or ensure that these libraries can be found by the linker.

- When attempting to refer to a function/entity in C from a function/entity in C++. This is caused by C++ name mangling, and can be avoided by marking C functions `extern "C"`.
- `Undefined symbol thunk{v:0,-44} to Foo_i::~Foo_i() (referred from Bar_i.o)`

  The symbol `thunk{v:0,-44} to Foo_i::~Foo_i()` is a wrapper function round the regular `Foo_i::~Foo_i()`.

  `Foo_i` is a derived class of some other base class, therefore:

  — It has a base-class vtable for when it is referred to by a pointer to that base class.

  — The base-class vtable has an entry for the thunk.

  — The destructor thunk is output when the actual (derived class) destructor is output.

  Therefore, to avoid the error, ensure this destructor is defined.

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

This error occurs when the default ordering rules used by the linker (RO followed by RW followed by ZI) are violated. This typically happens when one uses `+FIRST` or `+LAST`, for example in a scatter file, attempting to force RW before RO.

L6220E `<type> region <regionname> size (<size> bytes) exceeds limit (<limit> bytes).`

Example:

```
Execution region ROM_EXEC size (4208184 bytes) exceeds limit (4194304 bytes).
```

This can occur where a region has been given an (optional) maximum length in the scatter file, but this size of the code/data being placed in that region has exceeded the given limit. This error is suppressible with `--diag_suppress 6220`.

For example, this might occur when using `.ANY`*num* selectors with the `ALIGN` directive in a scatter file to force the linker to insert padding. You might be able to fix this using the `--any_contingency` option.

See the following in the *armlink User Guide*:

- *Placement of unassigned sections with the .ANY module selector on page 8-26*.

See the following in the *armlink Reference Guide*:

- *--any_contingency on page 2-5*.
- *--diag_suppress on page 2-39*.

L6221E `<type1> region <regionname1> with <addrtype1> range [<base1>,<limit1>) overlaps with <type2> region <regionname2> with <addrtype2> range [<base2>,<limit2>).`

This error can occur even though information in the scatter-loading description file and map information generated by the linker indicates that the execution regions do not overlap.

Example `test.s` file

```
    AREA area1, CODE
    ENTRY
    BX lr

    AREA area2, READWRITE, NOINIT
    SPACE 10

    AREA area3, READWRITE
    DCD 10
    END
```

Example `scatter.txt` file

```
LR1 0x8000
{
  ER1 +0
  {
    *(+ro)
  }
  ER2 +0
  {
    *(+zi)
  }
  ER3 +0
  {
    *(+rw)
  }
}
```

Built with:

```
armasm --cpu=8-A.32 test.s
armlink -o test.axf --scatter scatter.txt --no_remove test.o
```

Generates:

```
Warning: L6221E: Execution region ER2 with Execution range
[0x00008004,0x00008010) overlaps with Execution region ER3 with Load range
[0x00008004,0x00008008).
```

The linker might emit warning message L6221E when an execution region base address overlaps with the load address of another region. This could be because of an incorrect scatter file. The memory map of the image has a load view and an execution view, described by the scatter-loading file. A non-ZI section must have a unique load address and in most cases must have a unique execution address. The linker does not assign space to ZI execution regions. Therefore this warning might be because a load region LR2 with a relative base address immediately follows a ZI execution region in a load region LR1.

Because the overlapping part might not have real code or data inside, the warning might be harmless.

You can use the following linker options to find out the addresses of each region, to identify which regions overlap with a load region:

```
--load_addr_map_info --map --list=map.txt
```

You can do one of the following:

- Ignore the warning, only if after analysis it is possible to determine that the execution region is not going to corrupt the load region that has not yet been copied to its execution region address. Also, debug the application to confirm that it initializes and executes correctly.
- Adjust the base addresses of the execution regions.
- Use the FIXED scatter-loading attribute to make the load regions and execution regions have the same base addresses. The *armlink User Guide* and the *armlink Reference Guide* provide more information about the FIXED attribute.

See the following in the *armlink User Guide*:

- *Using the FIXED attribute to create root regions on page 8-17*.

See the following in the *armlink Reference Guide*:

- *Execution region attributes on page 4-13*.
- *Scatter files containing relative base address load regions and a ZI execution region on page 4-38*.

L6222E `Partial object cannot have multiple ENTRY sections, <e_oname>(<e_sname>) and <oname>(<sname>).`

Where objects are being linked together into a partially-linked object, only one of the sections in the objects can have an entry point.

Note

It is not possible in this case 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 occurs 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.`

This occurs if the linker can not match an input section to any of the selectors in your scatter file. You must correct your scatter file by adding an appropriate selector.

See the following in *armlink User Guide*:

- *Section placement with the linker on page 4-18*.

L6227E `Using --reloc with --rw-base without --split is not allowed.`

See the following in the *armlink Reference Guide*:

- *--reloc on page 2-116*.
- *--rw_base on page 2-121*.
- *--split on page 2-131*.

L6234E `<ss> must follow a single selector.`

For example, in a scatter file:

```
:
* (+FIRST, +RO)
:
```

`+FIRST` means place this (single) section first. Selectors that can match multiple sections (for example, `+RO` or `+ENTRY`) are not permitted to be used with `+FIRST` (or `+LAST`). If used together, the error message is generated.

L6235E `More than one section matches selector - cannot all be FIRST/LAST.`

See the following in the *armlink User Guide*:

- *Placing sections with FIRST and LAST attributes on page 4-20*.

See the following in the *armlink Reference Guide*:

- *Syntax of an input section description on page 4-24*.

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 `--no_remove`. ARM does not recommend this.
- ARM recommends that you add the `ENTRY` directive to `vectors.s`, to tell the linker that it is a possible entry point of your application such as, for example:

  ```
  AREA Vect, CODE
  ENTRY ; define this as an entry point
  Vector_table
  ...
  ```

  Then link with `--entry Vector_table` to define the real start of your code.

See the following in the *armlink User Guide*:

- *Placing sections with FIRST and LAST attributes on page 4-20*.

See the following in the *armlink Reference Guide*:

- *--entry on page 2-49*.
- *--info on page 2-65*.
- *--keep on page 2-74*.
- *--remove, --no_remove on page 2-118*.
- *Syntax of an input section description on page 4-24*.

See the following in the *armasm Reference Guide*:

- *ENTRY* on page 10-34.

L6238E `<objname>(<secname>) contains invalid call from '<attr1>' function to '<attr2>' function <sym>.`

This linker error is given where a stack alignment conflict is detected in object code. The *ABI for the ARM Architecture* suggests that code maintains eight-byte stack alignment at its interfaces. This permits efficient use of `LDRD` and `STRD` instructions (in ARM Architecture 5TE and later) to access eight-byte aligned `double` and `long long` data types.

Symbols such as `~PRES8` and `REQ8` are *Build Attributes* of the objects:

- `PRES8` means the object PREServes eight-byte alignment of the stack.
- `~PRES8` means the object does NOT preserve eight-byte alignment of the stack (~ meaning NOT).
- `REQ8` means the object REQuires eight-byte alignment of the stack.

This link error typically occurs in two cases:

- Where assembler code (that does not preserve eight-byte stack alignment) calls compiled C/C++ code (that requires eight-byte stack alignment).
- Where attempting to link legacy objects that were compiled with older tools with objects compiled with recent tools. Legacy objects that do not have these attributes are treated as `~PRES8`, even if they do actually happen to preserve eight-byte alignment.

For example:

```
Error: L6238E: foo.o(.text) contains invalid call from '~PRES8' function to 'REQ8' function foobar
```

This means that there is a function in the object `foo.o` (in the section named `.text`) that does not preserve eight-byte stack alignment, but which is trying to call function `foobar` that requires eight-byte stack alignment.

A similar warning that might be encountered is:

```
Warning: L6306W: '~PRES8' section foo.o(.text) should not use the address of 'REQ8' function foobar
```

where the address of an external symbol is being referred to.

There are two possible solutions to work-around this issue:

- Rebuild all your objects/libraries.

  If you have any assembler files, you must check that all instructions preserve eight-byte stack alignment, and if necessary, correct them.

  For example, change:

  ```
  STMFD sp!, {r0-r3, lr} ; push an odd number of registers
  ```

  to

  ```
  STMFD sp!, {r0-r3, r12, lr} ; push even number of registers
  ```

The assembler automatically marks the object with the PRES8 attribute if all instructions preserve eight-byte stack alignment, so it is no longer necessary to add the PRESERVE8 directive to the top of each assembler file.

- If you have any legacy objects/libraries that cannot be rebuilt, either because you do not have the source code, or because the old objects must not be rebuilt (for example, for qualification/certification reasons), then you must inspect the legacy objects to check whether they preserve eight-byte alignment or not.

  Use `fromelf -c` to disassemble the object code. C/C++ code compiled with ADS 1.1 or later normally preserves eight-byte alignment, but assembled code does not.

  If your objects do indeed preserve eight-byte alignment, then the linker error L6238E can be suppressed with the use of `--diag_suppress 6238` on the linker command line.

  By using this, you are effectively guaranteeing that these objects are PRES8.

  The linker warning L6306W is suppressible with `--diag_suppress 6306`.

See also *8 Byte Stack Alignment* http://infocenter.arm.com/help/topic/com.arm.doc.faqs/ka4127.html.

L6239E `Cannot call non-interworking <t2> symbol '<sym>' in <obj2> from <t1> code in <obj1>(<sec1>)`

Example:

```
Cannot call non-interworking ARM symbol 'ArmFunc' in object foo.o from
THUMB code in bar.o(.text)
```

This problem can be caused by `foo.c` not being compiled with the option `--apcs /interwork`, to enable A32 code to call T32 code (and T32 to A32) by linker-generated interworking veneers.

L6241E `<objname>(<secname>) cannot use the address of '<attr1>' function <sym> as the image contains '<attr2>' functions.`

When linking with `'--strict'`, the linker reports conditions that might fail as errors, for example:

```
Error: L6241E: foo.o(.text) cannot use the address of '~IW' function main
as the image contains 'IW' functions.
```

`IW` means *interworking*, and `~IW` means *non-interworking*.

L6242E `Cannot link object <objname> as its attributes are incompatible with the image attributes.`

Each object file generated by the compilation tools includes a set of attributes that indicates the options that it was built with. The linker checks the attributes of each object file it processes. If it finds attributes that are incompatible with those of object files it has loaded previously, it generates this error.

There are three common reasons for this error, each of which produces a different message:

- `Error: L6242E: Cannot link object foo.o as its attributes are incompatible with the image attributes.`
  `require four-byte alignment of eight-byte datatypes clashes with require eight-byte alignment of eight-byte data types.`

  This can occur when you try to link objects built using ARM Compiler toolchain with objects built using ADS or RVCT 1.2. In ADS and RVCT 1.2, `double` and `long long` data types were 4-byte aligned (unless you used the `-Oldrd` compiler option or the `__align` keyword). Subsequently, the ABI changed, so that `double` and `long long` data types are 8-byte aligned.

This change means that ADS and RVCT 1.2 objects and libraries using `double` or `long long` data types are not directly compatible with objects and libraries built using later toolchain versions, and so the linker reports an attribute clash.

- `Error: L6242E: Cannot link object foo.o as its attributes are incompatible with the image attributes.`
  `... pure-endian double clashes with mixed-endian double.`

  This can occur when you are linking objects built using the ARM Compiler toolchain with legacy SDT objects or objects built using either of the legacy compiler options `--fpu softfpa` or `--fpu fpa`. (These options are only supported in RVCT 2.1 and earlier). SDT used a non-standard format for little-endian `double` and big-endian `long long`. However ARM Compiler toolchain uses industry-standard `double` and `long long` types. If you attempt to link object files that use the different formats for little-endian `double` and big-endian `long long` then the linker reports this error.

  ARM recommends you rebuild your entire project using the ARM Compiler toolchain.
- `Error: L6242E: Cannot link object foo.o as its attributes are incompatible with the image attributes.`
  `... FPA clashes with VFP.`

  This error typically occurs when you attempt to link objects built with different `--fpu` options. ARM recommends you rebuild your entire project using the same `--fpu` options.

See *Are legacy objects and libraries compatible with my project?* `http://infocenter.arm.com/help/topic/com.arm.doc.faqs/ka3639.html`.

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`.

L6248E `<objname>(<secname>) in <attr1> region '<r1>' cannot have <rtype> relocation to <symname> in <attr2> region '<r2>'.`

This error relates to a feature that is unsupported in ARM Compiler 6.0.

L6254E `Invalid SYMDEF type : <type>.`

The content of the symdefs file is invalid.

See the following in *armlink User Guide*:

- *Symdefs file format on page 7-21*.

L6255E `Could not delete file <filename>: <reason>`

An I/O error occurred while trying to delete the specified file. The file was either read-only, or was not found.

L6257E `<object>(<secname>) cannot be assigned to overlaid Execution region '<ername>'.`

This message indicates a problem with the scatter file.

See the following in the *armlink Reference Guide*:

- Chapter 4 *Scatter File Syntax*.

L6258E `Entry point (<address>) lies in an overlaid Execution region.`

This message indicates a problem with the scatter file.

See the following in the *armlink Reference Guide*:

- Chapter 4 *Scatter File Syntax*.

L6259E `Reserved Word '<name>' cannot be used as a <type> region name.`

`<name>` is a reserved word, so choose a different name for your region.

L6260E `Multiple load regions with the same name (<regionname>) are not allowed.`

This message indicates a problem with the scatter file.

See the following in the *armlink Reference Guide*:

- Chapter 4 *Scatter File Syntax*.

L6261E `Multiple execution regions with the same name (<regionname>) are not allowed.`

This message indicates a problem with the scatter file.

See the following in the *armlink Reference Guide*:

- Chapter 4 *Scatter File Syntax*.

L6263E `<addr> address of <regionname> cannot be addressed from <pi_or_abs> Region Table in <regtabregionname>`

The Region Table contains information used by the C-library initialization code to copy, decompress, or create ZI. This error message is given when the scatter file specifies an image structure that cannot be described by the Region Table.

The error message is most common when PI and non-PI Load Regions are mixed in the same image.

L6265E `Non-PI Section <obj>(<sec>) cannot be assigned to PI Exec region <er>.`

This might be caused by explicitly specifying the wrong ARM library on the linker command-line. Either:

- Remove the explicit specification of the ARM library.
- Replace the library, for example, `c_t.l`, with the correct library.

L6266E `RWPI Section <obj>(<sec>) cannot be assigned to non-PI Exec region <er>.`

This error relates to a feature that is unsupported in ARM Compiler 6.0.

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

This message indicates a problem with the scatter file.

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

This message indicates a problem with the scatter file.

L6273E `Section <objname>(<secname>) has mutually exclusive attributes (READONLY and ZI)`

This message indicates a problem with the object file.

L6275E `COMMON section <obj1>(<sec1>) does not define <sym> (defined in <obj2>(<sec2>))`

Given a set of `COMMON` sections with the same name, the linker selects one of them to be added to the image and discards all others. The selected `COMMON` section must define all the symbols defined by any rejected `COMMON` section, otherwise a symbol that was defined by a rejected section would become undefined again. The linker generates an error if the selected copy does not define a symbol that a rejected copy does. This error is normally caused by a compiler fault. Contact your supplier.

L6276E `Address <addr> marked both as <s1>(from <sp1>(<obj1>) via <src1>) and <s2>(from <sp2>(<obj2>) via <src2>).`

The image cannot contain contradictory mapping symbols for a given address, because the contents of each word in the image are uniquely typed as A32 (`$a`) or T32 (`$t`) code, DATA (`$d`), or NUMBER. It is not possible for a word to be both A32 code and DATA. This might indicate a compiler fault. Contact your supplier.

L6280E `Cannot rename <sym> using the given patterns.`

See the following in the *armlink Reference Guide*:

- *RENAME on page 3-5*.

L6281E `Cannot rename both <sym1> and <sym2> to <newname>.`

See the following in the *armlink Reference Guide*:

- *RENAME on page 3-5*.

L6282E `Cannot rename <sym> to <newname> as a global symbol of that name exists (defined) in <obj>).`

See the following in the *armlink Reference Guide*:

- *RENAME on page 3-5*.

L6283E `Object <objname> contains illegal local reference to symbol <symbolname>.`

An object cannot contain a reference to a local symbol, because local symbols are always defined within the object itself.

L6285E `Non-relocatable Load region <lr_name> contains R-Type dynamic relocations. First R-Type dynamic relocation found in <object>(<secname>) at offset 0x<offset>.`

This error occurs where there is a PI reference between two separate segments, if the two segments can be moved apart at runtime. When the linker sees that the two sections can be moved apart at runtime it generates a relocation (an R-Type relocation) that can be resolved if the sections are moved from their statically linked address. However the linker faults this relocation (giving error L6285E) because PI regions must not have relocations with respect to other sections as this invalidates the criteria for being position independent.

L6286E `Relocation #<rel_class>:<rel_number> in <objname>(<secname>) with respect to {symname|%s}. Value(<val>) out of range(<range>) for (<rtype>)`

This error typically occurs in the following situations:

- In handwritten assembly code, where there are not enough bits within the instruction opcode to hold the offset to a symbol.

  For example, the offset range is +/-4095 for an A32 state `LDR`/`STR` instruction.

- When the linker is having difficulty placing veneers around a large code section in your image.

  When the linker places a veneer near a very large section it must decide whether to place the veneer before or after the section. When the linker has placed the veneer it might have to place more veneers, which could be placed between the original veneer and its target. This would increase the distance between the veneer and its target.

  The linker automatically allows for modest increases in distances between veneers and their targets. However, a large number of veneers placed between the original veneer and its target might result in the target moving out of range. If this occurs the linker generates message L6286E.To work around this you can move large code sections away from areas where the

linker is placing many veneers. This can be done either by placing large sections in their own regions or by placing them first in the region they are located in using the +FIRST directive in the scatter-loading description file.

For example:

```
LOAD 0x0A000000 0x1000000
{
   ROM1 +0x0
   {
      *(+RO)
   }
}
```

Can be changed to:

```
LOAD 0x0A000000 0x1000000{
   ROM1 +0x0
   {
      *(+RO)
   }
   ROM1A +0x0
   {
      large.o (+RO)
   }
}
```

- When .ARM.exidx exception-handling index tables are placed in different execution regions, or too far from exception handling code.

  The .ARM.exidx exception-handling index tables must be located in a single execution region. Also, the distance from these tables to the C++ code that uses C++ exception handling must be within the range -0x40000000 to 0x3fffffff. Otherwise, the linker reports the following error:

  ```
  L6286: Value(0x9ff38980) out of range(-0x40000000 - 0x3fffffff) for
  relocation #0 (R_ARM_PREL31), wrt symbol xxx in XXXX.o(.ARM.exidx)
  ```

  This behavior is specified in the *ARM Exception Handling ABI* (EHABI). The EHABI states that the R_ARM_PREL31 relocation, which .ARM.exidx uses, does not use the highest bit (bit 31) for calculating the relocation.

  The most likely cause of this is that C++ code, that must access the .ARM.exidx sections, has been split and placed into separate execution regions, outside of the valid range (-0x40000000 to 0x3fffffff).

  To resolve this error, if you have memory between the separated execution regions, place the .ARM.exidx section there with the selector *(.ARM.exidx). For example:

  ```
  LOAD_ROM 0x00000000
  {
     ER1 0x00000000 ;the distance from ER2 to ER1 is out of range
     {
        file1.o (+RO) ; from a C++ source
        * (+RO)
     }
     ERx 0x30000000
     {
        *(.ARM.exidx) ; ARM.exidx to ER1 and ER2 both in range
     }
     ER2 0x60000000
     {
        file2.o (+RO) ; from a C++ source
     }
     ER3 +0
     {
        * (+RW, +ZI)
     }
  ```

Otherwise, try placing the code into an execution region close enough to the tables (within the range of -0x40000000 to 0x3fffffff).

In other cases, make sure you have the latest patch installed from *Downloads* https://silver.arm.com/browse.

For more information see *Value out of range for relocation* http://infocenter.arm.com/help/topic/com.arm.doc.faqs/ka3553.html.

L6287E `Illegal alignment constraint (<align>) specified for <objname>(<secname>).`

An illegal alignment was specified for an ELF object.

L6291E `Cannot assign Fixed Execution Region <ername> Load Address:<addr>. Load Address must be greater than or equal to next available Load Address:<load_addr>.`

See the following in the *armlink Reference Guide*:

- *Execution region attributes on page 4-13*.

L6292E `Ignoring unknown attribute '<attr>' specified for region <regname>.`

This error message is specific to execution regions with the `FIXED` attribute. `FIXED` means make the load address the same as the execution address. The linker can only do this if the execution address is greater than or equal to the next available load address within the load region.

See the following in the *armlink User Guide*:

- *Using the FIXED attribute to create root regions on page 8-17*.

See the following in the *armlink Reference Guide*:

- *Execution region attributes on page 4-13*.

L6294E `<type> region <regionname> spans beyond 32 bit address space (base <base>, size <size> bytes).`

This error message relates to a problem with the scatter file.

L6296E `Definition of special symbol <sym1> is illegal as symbol <sym2> is absolute.`

See L6188E.

L6300W `Common section <object1>(<section1>) is larger than its definition <object2>(<section2>).`

This might indicate a compiler fault. Contact your supplier.

L6301W `Could not find file <filename>: <reason>`

The specified file was not found in the default directories.

L6302W `Ignoring multiple SHLNAME entry.`

There can be only one `SHLNAME` entry in an edit file. Only the first such entry is accepted by the linker. All subsequent `SHLNAME` entries are ignored.

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 use linker option `--entry` to specify the single, unique entry, for example:

```
--entry 0x0
```

or

`--entry <label>`

The label form is typical for an embedded system.

L6306W `'<attr1>' section <objname>(<secname>) should not use the address of '<attr2>' function <sym>.`

See L6238E.

L6307W `Relocation #<rel_class>:<rel_num> in <objname>(<secname>) with respect to <sym>. 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 linker command-line, but the library does not contain any members.

L6310W `Unable to find ARM libraries.`

This is most often caused by incorrect arguments to `--libpath` or an invalid value for an environment variable.

Set the correct path with either the `--libpath` linker option or the environment variable. The default path for a Windows installation is:

`install_directory``\lib`

Ensure this path does not include:

- `\armlib`.
- `\cpplib`.
- Any trailing slashes (\) at the end. These are added by the linker automatically.

Use `--verbose` or `--info libraries` to display where the linker is attempting to locate the libraries.

See the following in the *armlink Reference Guide*:

- *--info on page 2-65*.
- *--libpath on page 2-81*.
- *--verbose on page 2-158*.

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

See L6218E.

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

For example, use of `IWV$$Code` within the scatter file is obsolete. Replace `IWV$$Code` with `Veneer$$Code`.

L6314W `No section matches pattern <module>(<section>).`

Example:

`No section matches pattern foo.*o(ZI).`

This can occur for the following reasons:

- 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.

- 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.
- You have used `__attribute__((section(address)))` in your source code to place code and data at a specific address. You have also specified `*(.ARM.__AT_address)` in a scatter file, but you have not specified the address with the same number of hexadecimal digits. For example, you might have specified `__attribute__((section(0x10000)))` in your source code, but you specified the section name as `*(.ARM.__AT_0x00010000)` in the scatter file.

See the following in the *armlink User Guide*:

- *Methods of placing functions and data at specific addresses on page 8-18*.
- *Placement of sections at a specific address with __attribute__((section(".ARM.__at_address"))) on page 8-41*.

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 <sec_no>`

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.

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
```

This results in the message:

`init.o(foo) contains branch to a non-code symbol bar.`

If the destination has no name:

`BL 0x200 ; Branch with link to 0x200 bytes ahead of PC`

the following message is displayed:

`bootsys.o(BOOTSYS_IVT) contains branch to a non-code symbol <Anonymous Symbol>.`

This warning can also appear when linking objects generated by GCC. GCC uses linker relocations for references internal to each object. The targets of these relocations might not have appropriate mapping symbols that permit the linker to determine whether the target is code or data, so a warning is generated. By contrast, `armclang` resolves all such references at compile-time.

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

For example, when building a Linux application, you might have:

`--keep *(.init_array)`

on the linker command-line in your makefile, but this section might not be present when building with no C++, in which case this warning is reported:

`Ignoring --keep command. Cannot find section *(.init_array)`

You can often ignore this warning, or suppress it with `--diag_suppress 6319`.

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 that uses the `+offset` form of base address inherits the `PI`, `RELOC`, or `OVERLAY` attributes from either:

- The previous region in the description.
- The parent load region if it is the first execution region in the load region.

See the following in the *armlink Reference Guide*:

- *Inheritance rules for load region address attributes on page 4-20*.
- *Inheritance rules for execution region address attributes on page 4-21*.
- *Inheritance rules for the RELOC address attribute on page 4-22*.

L6326W `Ignoring ZEROPAD attribute for non-root execution region <ername>.`

`ZEROPAD` only applies to root execution regions. A root region is a region whose execution address is the same as its load address, and so does not require moving or copying at run-time.

See the following in the *armlink Reference Guide*:

- *Execution region attributes on page 4-13*.

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 the following in the *armlink User Guide*:

- *Elimination of unused sections on page 5-4*.

See the following in the *armlink Reference Guide*:

- *--info on page 2-65*.

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

This means that an unused section has had its base and limit symbols referenced. For more information, use `--info unused`.

See the following in the *armlink User Guide*:

- *Elimination of unused sections on page 5-4*.

See the following in the *armlink Reference Guide*:

- *--info on page 2-65*.

L6334W `Overalignment <overalignment> for region <regname> cannot be negative.`

See the following in the *armlink User Guide*:

- *Overalignment of execution regions and input sections on page 8-62*.

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

The compiler is able to perform tailcall optimization for improved code size and performance. However, there is a problematic sequence for Architecture 4T code where a T32 IW function calls (by a veneer) an A32 IW function, which tailcalls an A32 not-IW function. The return from the A32 not-IW function can pop the return address off the stack into the PC instead of using the correct `BX` instruction. The linker can detect this situation and report this warning.

T32 IW tailcalls to T32 not-IW do not occur because T32 tailcalls with `B` are so short ranged that they can only be generated to functions in the same ELF section which must also be T32.

The warning is pessimistic in that an object *might* contain invalid tailcalls, but the linker cannot be sure because it only looks at the attributes of the objects, not at the contents of their sections.

To avoid the warning, either rebuild your objects with interworking enabled, or manually inspect the A32 IW function to check for tailcalls (that is, where function calls are made using an ordinary branch `B` instruction), to check whether this is a real problem. This warning can be suppressed with `--diag_suppress L6335W`.

L6339W `Ignoring RELOC attribute for execution region <er_name>.`

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.

See the following in the *armlink Reference Guide*:

- *Execution region attributes on page 4-13*.

L6340W `options first and last are ignored for link type of <linktype>`

The `--first` and `--last` options are meaningless when creating a partially-linked object.

L6370E `cpu <cpu> is not compatible with fpu <fpu>`

See the following in the *armlink Reference Guide*:

- *--cpu on page 2-31*.
- *--fpu on page 2-61*.

L6384E `No Load Execution Region of name <region> seen yet at line <line>.`

This might be because you have used the current base address in a limit calculation in a scatter file. For example:

```
ER_foo 0 ImageBase(ER_foo)
```

L6387E `Load Region Expressions can only be used in ScatterAssert expressions on line <line>`

See the following in the *armlink Reference Guide*:

- *ScatterAssert function and load address related functions on page 4-40*.

L6388E `ScatterAssert expression <expr> failed on line <line>`

See the following in the *armlink Reference Guide*:

- *ScatterAssert function and load address related functions on page 4-40*.

L6390E `Conditional operator (expr) ? (expr) : (expr) on line <line> has no : (expr).`

See the following in the *armlink Reference Guide*:

- *About Expression evaluation in scatter files on page 4-32*.
- *Expression rules in scatter files on page 4-34*.

L6404W `FILL value preferred to combination of EMPTY, ZEROPAD and PADVALUE for Execution Region <name>.`

See the following in the *armlink Reference Guide*:

- *Execution region attributes on page 4-13*.

L6405W `No .ANY selector matches Section <name>(<objname>).`

See the following in the *armlink User Guide*:

- *Placement of unassigned sections with the .ANY module selector on page 8-26*.

L6406W `No space in execution regions with .ANY selector matching Section <name>(<objname>).`

This occurs if there is not sufficient space in the scatter file regions containing `.ANY` to place the section listed. You must modify your scatter file to ensure there is sufficient space for the section.

See the following in the *armlink User Guide*:

- *Placement of unassigned sections with the .ANY module selector on page 8-26*.

L6407W `Sections of aggregate size 0x<size> bytes could not fit into .ANY selector(s).`

This warning identifies the total amount of image data that cannot be placed in any `.ANY` selectors.

For example, `.ANY(+ZI)` is placed in an execution region that is too small for the amount of `ZI` data:

```
ROM_LOAD 0x8000
{
    ROM_EXEC 0x8000
    {
        .ANY(+RO,+RW)
    }
    RAM +0 0x{...} <<< region max length is too small
    {
        .ANY(+ZI)
    }
}
```

See the following in the *armlink User Guide*:

- *Placement of unassigned sections with the .ANY module selector on page 8-26*.

L6411E `No compatible library exists with a definition of startup symbol <name>.`

The compiler does not generate `$$Lib$Request` symbols when building objects, so `armlink` does not automatically link with the ARM libraries, resulting in this message.

Invoke `armlink` with `--force_scanlib` to link with the ARM libraries. When compiling and linking in one step, the compiler automatically passes this option to `armlink`.

See the following in the *armlink Reference Guide*:

- *--force_scanlib on page 2-58*.

L6414E `--ropi used without --rwpi or --rw-base.`

This error relates to options that are unsupported in ARM Compiler 6.0.

L6415E `Could not find a unique set of libraries compatible with this image. Suggest using the --cpu option to select a specific library.`

See the following in the *armlink Reference Guide*:

- *--cpu on page 2-31*.

L6422U `PLT generation requires an architecture with ARM instruction support.`

For the linker to generate `PLT`, you must be using a target that supports the A32 instruction set. For example, the linker cannot generate `PLT` for a Cortex-M3 target.

L6429U `Attempt to set maximum number of open files to <val> failed with error code <error>.`

An attempt to increase the number of file handles armlink can keep open at any one time has failed.

L6443W `Data Compression for region <region> turned off. Region contains reference to symbol <symname> which depends on a compressed address.`

The linker requires the contents of a region to be fixed before it can be compressed and cannot modify it after it has been compressed. Therefore a compressible region cannot refer to a memory location that depends on the compression process.

L6463U `Input Objects contain <archtype> instructions but could not find valid target for <archtype> architecture based on object attributes. Suggest using --cpu option to select a specific cpu.`

See the following in the *armlink Reference Guide*:

- *--cpu on page 2-31*.

L6464E `Only one of --dynamic_debug, --emit-relocs and --emit-debug-overlay-relocs can be selected.`

See the following in the *armlink Reference Guide*:

- *--emit_debug_overlay_relocs on page 2-45*.
- *--emit_relocs on page 2-48*.

L6468U `Only --pltgot=direct or --pltgot=none supported for --base_platform with multiple Load Regions containing code.`

See the following in the *armlink Reference Guide*:

- *--base_platform on page 2-13*.
- *--pltgot on page 2-108*.

L6469E `--base_platform does not support RELOC Load Regions containing non RELOC Execution Regions. Please use +0 for the Base Address of Execution Region <ername> in Load Region <lrname>.`

See the following in the *armlink Reference Guide*:

- *--base_platform on page 2-13*.
- *Inheritance rules for the RELOC address attribute on page 4-22*.

L6471E `Branch Relocation <rel_class>:<idx> in section <secname> from object <objname> refers to ARM Absolute <armsym> symbol from object <armobjname>, Suppress error to treat as a Thumb address.`

`Relocation #<rel_class>:<idx> in <objname>(<secname>) with respect to <armsym>. Branch refers to ARM Absolute Symbol defined in <armobjname>, Suppress error to treat as a Thumb address.`

L6475W `IMPORT/EXPORT commands ignored when --override_visibility is not specified`

The symbol you are trying to export, either with an `EXPORT` command in a steering file or with the `--undefined_and_export` command-line option, is not exported because of low visibility.

See the following in the *armlink Reference Guide*:

- *--override_visibility on page 2-101*.
- *--undefined_and_export on page 2-150*.
- *EXPORT on page 3-2*.

L6629E `Unmatched parentheses expecting ) but found <character> at position <col> on line <line>`

This message indicates a parsing error in the scatter file.

L6630E `Invalid token start expected number or ( but found <character> at position <col> on line <line>`

This message indicates a parsing error in the scatter file.

L6631E `Division by zero on line <line>`

This message indicates an expression evaluation error in the scatter file.

L6632W `Subtraction underflow on line <line>`

This message indicates an expression evaluation error in the scatter file.

L6634E `Pre-processor command in '<filename>'too long, maximum length of <max_size>`

This message indicates a problem with pre-processing the scatter file.

L6635E `Could not open intermediate file '<filename>' produced by pre-processor: <reason>`

This message indicates a problem with pre-processing the scatter file.

L6636E `Pre-processor step failed for '<filename>'`

This message indicates a problem with pre-processing the scatter file.

L6637W `No input objects specified. At least one input object or library(object) must be specified.`

At least one input object or library(object) must be specified.

L6638U `Object <objname> has a link order dependency cycle, check sections with SHF_LINK_ORDER`

L6640E `PDTTable section not least static data address, least static data section is <secname>`

This error relates to features that are unsupported in ARM Compiler 6.0.

L6643E `The virtual function elimination information in section <sectionname> refers to the wrong section.`

This message might indicate a compiler fault. Contact your supplier.

L6644E `Unexpectedly reached the end of the buffer when reading the virtual function elimination information in section <oepname>(<sectionname>).`

This message might indicate a compiler fault. Contact your supplier.

L6645E `The virtual function elimination information in section <oepname>(<sectionname>) is incorrect: there should be a relocation at offset <offset>.`

This message might indicate a compiler fault. Contact your supplier.

L6646W `The virtual function elimination information in section <oepname>(<sectionname>) contains garbage from offset <offset> onwards.`

This message might indicate a compiler fault. Contact your supplier.

L6647E `The virtual function elimination information for <vcall_objectname>(<vcall_sectionname>) incorrectly indicates that section <curr_sectionname>(object <curr_objectname>), offset <offset> is a relocation (to a virtual function or RTTI), but there is no relocation at that offset.`

This message might indicate a compiler fault. Contact your supplier.

L6649E `EMPTY region <regname> must have a maximum size.`

See the following in the *armlink Reference Guide*:

- *Execution region attributes on page 4-13*.

L6654E `Rejected Local symbol <symname> referred to from a non group member <objname>(<nongrpname>)`

This message might indicate a compiler fault. Contact your supplier.

L6656E `Internal error: the vfe section list contains a non-vfe section called <oepname>(<secname>).`

This message might indicate a compiler fault. Contact your supplier.

L6664W `Relocation #<rel_class>:<rel_number> in <objname>(<secname>) is with respect to a symbol(#<idx> before last Map Symbol #<last>).`

L6665W `Neither Lib$$Request$$armlib Lib$$Request$$cpplib defined, not searching ARM libraries.`

The following code produces this warning:

```
  AREA Block, CODE, READONLY
  EXPORT func1
  ;IMPORT || Lib$$Request$$armlib||
  IMPORT printf
func1
  LDR r0,=string
  BL printf
  BX lr
  AREA BlockData, DATA
string DCB "mystring"
  END
```

The linker has not been told to look in the libraries and so cannot find the symbol `printf`.

This also causes an error:

```
L6218E: Undefined symbol printf (referred from L6665W.o).
```

If you do not want the libraries, then ignore this message. Otherwise, to fix both the error and the warning uncomment the line:

```
IMPORT || Lib$$Request$$armlib||
```

Note

Use the `--force_scanlib` option to tell `armlink` to link with the ARM libraries.

See the following in the *armlink Reference Guide*:

- *--force_scanlib on page 2-58*.

L6707E `Padding value not specified with PADVALUE attribute for execution region <regionname>.`

See the following in the *armlink Reference Guide*:

- *Execution region attributes on page 4-13*.

L6720U `Exception table <spname> from object <oepname> present in image, --noexceptions specified.`

See the following in the *armlink Reference Guide*:

- *--exceptions, --no_exceptions on page 2-51*.

L6738E `Relocation #<rel_class>:<relocnum> in <oepname>(<secname>) with respect to <wrtsym> is a GOT-relative relocation, but _GLOBAL_OFFSET_TABLE_ is undefined.`

Some GNU produced images can refer to the symbol named `_GLOBAL_OFFSET_TABLE_`. If there are no `GOT Slot` generating relocations and the linker is unable to pick a suitable address for the `GOT base` the linker issues this error message.

L6747W `Raising target architecture from <oldversion> to <newversion>.`

If the linker detects objects that specify the obsolete ARMv3, it upgrades these to ARMv4 to be usable with ARM libraries.

L6765W `Shared object entry points must be ARM-state when linking architecture 4T objects.`

This can occur when linking with GNU C libraries. The GNU startup code `crt1.o` does not have any build attributes for the entry point, so the linker cannot determine which execution state (A32 or T32) the code runs in. Because the GNU C library startup code is A32 code, you can safely ignore this warning, or suppress it with `--diag_suppress 6765`.

L6783E `Mapping symbol #<symnum> '<msym>' in <oepname>(<secnum>:<secname>) defined at the end of, or beyond, the section size (symbol offset=0x<moffset>, section size=0x<secsize>)`

This indicates that the address for a section points to a location at the end of or outside of the ELF section. This can be caused by an empty inlined data section and indicates there might be a problem with the object file. You can use `--diag_warning 6783` to suppress this error.

L6784E `Symbol #<symnum> '<symname>' in <oepname>(<secnum>:<secname>) with value <value> has size 0x<size> that extends to outside the section.`

The linker encountered a symbol with a size that extends outside of its containing section.

L6788E `Scatter-loading of execution region <er1name> to [<base1>,<limit1>) will cause the contents of execution region <er2name> at [<base2>,<limit2>) to be corrupted at run-time.`

This might occur during the placement of an execution region when scatter-loading takes place. It happens if an execution region is put in a position where it overwrites partially or wholly itself or another execution region.

For example, this works:

```
LOAD_ROM 0x0000 0x4000
{
  EXEC1 0x0000 0x4000
```

```
  {
    * (+RO)
  }
  EXEC2 0x4000 0x4000
  {
    * (+RW,+ZI)
  }
}
```

This generates the error:

```
LOAD_ROM 0x0000 0x4000
{
  EXEC1 0x4000 0x4000
  {
    * (+RW,+ZI)
  }
  EXEC2 0x0000 0x4000
  {
    * (+RO)
  }
}
```

It reports:

```
Error: L6788E: Scatter-loading of execution region EXEC2 will cause the
contents of execution region EXEC2 to be corrupted at run-time.
```

See the following in the *armlink User Guide*:

- Chapter 8 *Scatter-loading Features*.

L6799E `Expecting Landing Pad reference at offset 0x<offset> in Function Specification descriptor <oepname>(<secname>).`

A landing pad is code that cleans up after an exception has been raised. If the linker detects old-format exception tables, it automatically converts them to the new format.

This message does not appear unless you are using a later version of the linker with an earlier version of the compiler.

L6800W `Cannot convert generic model personality routine at 0x<offset> <oepname>(<secname>).`

A personality routine unwinds the exception handling stack. If the linker detects old-format exception tables then it automatically converts them to the new format. This message indicates a fault in the compiler. Contact your supplier.

L6801E `<objname>(<secname>) containing <secarmthumb> code cannot use the address of '~IW (The user intended not all code should interwork)' <funarmthumb> function <sym>.`

The linker can diagnose where a non-interworking (`~IW`) function has its address taken by code in the other state. This error is disabled by default, but can be enabled by linking with `--strict`. The error can be downgraded to a warning with `--diag_warning 6801` and subsequently suppressed completely if required with `--diag_suppress 6801`

Where code, for example, in `a.c` uses the address of a non-interworking function in `t.c`:

```
armclang --target=armv8a-arm-none-eabi -marm -c a.c
armclang --target=armv8a-arm-none-eabi -mthumb -c t.c
armlink --force_scanlib t.o a.o --strict
```

reports:

```
Error: L6801E: a.o(.text) containing ARM code cannot use the address of
'~IW' Thumb function foo.
```

L6810E `Relocation <rel_class>:<idx> in <objname>(<secname>) is of obsolete type <rtype>`

Relocation errors and warnings are most likely to occur if you are linking object files built with previous versions of the ARM tools.

To show relocation errors and warnings use the `--strict_relocations` switch. This option enables you to ensure ABI compliance of objects. It is off by default, and deprecated and obsolete relocations are handled silently by the linker.

See the following in the *armlink Reference Guide*:

- *--strict_relocations, --no_strict_relocations on page 2-137*.

L6813U `Could not find Symbol <symname> to rename to <newname>.`

See the following in the *armlink Reference Guide*:

- *RENAME on page 3-5*.

L6815U `Out of memory. Allocation Size:<alloc_size> System Size:<system_size>.`

This error provides information about the amount of memory available and the amount of memory required to perform the link step.

This error occurs because the linker does not have enough memory to link your target object. This is not common, but might be triggered for a number of reasons, such as:

- Linking very large objects or libraries together.
- Generating a large amount of debug information.
- Having very large regions defined in your scatter file.

In these cases, your workstation might run out of virtual memory.

This issue might also occur if you use the `FIXED` scatter-loading attribute. The `FIXED` attribute forces an execution region to become a root region in ROM at a fixed address. The linker might have to add padding bytes between the end of the previous execution region and the `FIXED` region, to generate the ROM image. The linker might run out of memory if large amounts of padding are added when the address of the `FIXED` region is far away from the end of the execution region. The link step might succeed if the gap is reduced.

See the following in the *armlink Reference Guide*:

- *Execution region attributes on page 4-13*.

See the following in the *armlink User Guide*:

- *Using the FIXED attribute to create root regions on page 8-17*.

While the linker can generate images of almost any size, it requires a larger amount of memory to run and finish the link. Try the following solutions to improve link-time performance, to avoid the `Out of memory` error:

1. Shut down all non-essential applications and processes when you are linking.

   For example, if you are running under Eclipse, try running your linker from the command-line, or exiting and restarting Eclipse between builds.

2. Use the `--no_debug` linker option.

   This command tells the linker to create the object without including any debug information. See the following in the *armlink Reference Guide*:

   - *--debug, --no_debug on page 2-35*.

Note

It is not possible to perform source level debugging if you use this option.

3. Reduce debug information.

   There are methods you can use to try and reduce debug information. See `-Onum` in the *armclang Reference Guide*.

   You can also use the fromelf utility to strip debug information from objects and libraries that you do not have to debug. See the following in the *fromelf User Guide*:

   - *--strip=option[,option,...]* on page 4-66.

4. Use partial linking.

   You can use partial linking to split the link stage over a few smaller operations. Doing this also stops duplication of the object files in memory in the final link.

   See the following in the *armlink Reference Guide*:

   - *--partial on page 2-105*.

5. Increase memory support on Windows operating systems.

   On some Windows operating systems it is possible to increase the virtual address space from 2GB (the default) to 3GB. For more information, see the article *Memory Support and Windows Operating Systems* at *Microsoft Developer Network* `http://msdn.microsoft.com/`.

6. Use the `--no_eager_load_debug` linker option.

   This option causes the linker to remove debug section data from memory after object loading. This lowers the peak memory usage of the linker at the expense of some linker performance, because much of the debug data has to be loaded again when the final image is written.

   See the following in the *armlink Reference Guide*:

   - *--eager_load_debug, --no_eager_load_debug on page 2-43*.

If you are still experiencing the same problem, raise a support case.

L6828E `Relocation #<rel_class>:<idx> in <objname>(<secname>) with respect to <symname>, Branch source address <srcaddr> cannot reach next available pool at [<pool_base>,<pool_limit>). Please use the --veneer_pool_size option to increase the contingency.`

The `--veneer_inject_type=pool` veneer generation model requires branches to veneers in the pool to be able to reach the pool limit, which is the highest possible address a veneer can use. If a branch is later found that cannot reach the pool limit, and `armlink` is able to fit all the veneers in the pool into the lower pool limit, then `armlink` reduces the pool limit to accommodate the branch. Error message L6828 is issued only if `armlink` is unable to lower the pool limit.

See the following in the *armlink Reference Guide*:

- *--veneer_inject_type on page 2-155*.

L6915E `Library reports error: <msg>`

The message is typically one of the following:

- `Error: L6915E: Library reports error: scatter-load file declares no heap or stack regions and __user_initial_stackheap is not defined.`

  or

  `Error: L6915E: Library reports error: The semihosting __user_initial_stackheap cannot reliably set up a usable heap region if scatter loading is in use`

It is most likely that you have not re-implemented `__user_setup_stackheap()` or you have not defined `ARM_LIB_STACK` or `ARM_LIB_HEAP` regions in the respective scatter file.

Note

`__user_setup_stackheap()` supersedes the deprecated function `__user_initial_stackheap()`.

See the following in the *ARM C and C++ Libraries and Floating-Point Support Reference Guide*:

— *__user_setup_stackheap()* on page 2-58.

See the following in the *armlink User Guide*:

— *Reserving an empty region on page 8-56*.

- `Error: L6915E: Library reports error: __use_no_semihosting was requested but <function> was referenced.`

  Where `<function>` represents `__user_initial_stackheap`, `_sys_exit`, `_sys_open`, `_sys_tmpnam`, `_ttywrch`, `system`, `remove`, `rename`, `_sys_command_string`, `time`, or `clock`

  This error can appear when retargeting semihosting-using functions, to avoid any `SVC/BKPT` instructions being linked-in from the C libraries.

  Ensure that no semihosting-using functions are linked in from the C library.

  To resolve this, you might have to provide your own implementations of these C library functions.

  The `emb_sw_dev` directory contains examples of how to re-implement some of the more common semihosting-using functions. See the file `retarget.c`.

  See the following in the *ARM C and C++ Libraries and Floating-Point Support User Guide*:

  — *Using the libraries in a nonsemihosting environment* on page 2-35.

  Note

  The linker does not report any semihosting-using functions, such as `__semihost()`, in your own application code.

  To identify which semihosting-using functions are still being linked-in from the C libraries:

  — Link with `armlink --verbose --list err.txt`

  — Search `err.txt` for occurrences of `__I_use_semihosting`

  For example:

  ```
  …
  Loading member sys_exit.o from c_4.l.
  reference : __I_use_semihosting
  definition: _sys_exit
  …
  ```

  This shows that the semihosting-using function `_sys_exit` is linked-in from the C library. To prevent this, you must provide your own implementation of this function.

- `Error: L6915E: Library reports error:__use_no_heap was requested, but <reason> was referenced`

  If `<reason>` represents `malloc`, `free`, `__heapstats`, or `__heapvalid`, the use of `__use_no_heap` conflicts with these functions.

- `Error: L6915E: Library reports error:__use_no_heap_region was requested, but <reason> was referenced`

If <reason> represents malloc, free, __heapstats, __heapvalid, or __argv_alloc, the use of __use_no_heap_region conflicts with these functions.

L6971E <objname>(<secname>) type <type> incompatible with <prevobj>(<prevname>) type <prevtype> in er <ername>.

You might see this message when placing __at sections with a scatter file.

See the following in the *armlink User Guide*:

- *Methods of placing functions and data at specific addresses on page 8-18*.
- *Placement of sections at a specific address with __attribute__((section(".ARM.__at_address"))) on page 8-41*.

L6974E AT section <name> does not have a base address.

See the following in the *armlink User Guide*:

- *Placement of sections at a specific address with __attribute__((section(".ARM.__at_address"))) on page 8-41*.

L6980W FIRST and LAST ignored for <objname>(<secname>) with required base address.

See the following in the *armlink User Guide*:

- *Placing sections with FIRST and LAST attributes on page 4-20*.

L6981E __AT incompatible with BPABI and SystemV Image types

See the following in the *armlink User Guide*:

- *Restrictions on placing __at sections on page 8-42*.

L6982E AT section <objname>(<spname>) with base <base> limit <limit> overlaps address range with AT section <obj2name>(<sp2name>) with base <base2> limit <limit2>.

See the following in the *armlink User Guide*:

- *Placement of sections at a specific address with __attribute__((section(".ARM.__at_address"))) on page 8-41*.

L6983E AT section <objname>(<spname>) with required base address <base> out of range for ER <ername> with base <erbase> and limit <erlimit>.

This can occur if you specify __attribute__((section(.ARM.__at_*address*))) in your code, the *address* is outside the range of all execution regions specified in your scatter file, and you specify --no_autoat option on the linker command-line. In this case, the address part of .ARM.__at_*address* must be within an execution region. For example:

```
int x1 __attribute__((section(.ARM.__at_0x4000)));  // defined in
function.c

; scatter file
LR1 0x0
{
    ...
        function.o(.ARM.__at_0x4000)
    ...
}
```

See the following in the *armlink User Guide*:

- *Placement of sections at a specific address with __attribute__((section(".ARM.__at_address"))) on page 8-41*.

See the following in the *armlink Reference Guide*:

- *--autoat, --no_autoat on page 2-11*.

L6984E `AT section <objname>(<spname>) has required base address <base> which is not aligned to section alignment <alignment>.`

See the following in the *armlink User Guide*:

- *Placement of sections at a specific address with __attribute__((section(".ARM.__at_address"))) on page 8-41*.

L6985E `Unable to automatically place AT section <objname>(<spname>) with required base address <base>. Please manually place in the scatter file using the --no_autoat option.`

This can occur if you specify `__attribute__((section(.ARM.__at_address)))` in your code, the *address* is outside the range of all execution regions specified in your scatter file, and you specify `--autoat` option on the linker command-line. In this case, the address part of `.ARM.__at_address` must be within an execution region, and you must specify the `--no_autoat` option on the linker command-line.

See the following in the *armlink User Guide*:

- *Placement of sections at a specific address with __attribute__((section(".ARM.__at_address"))) on page 8-41*.

See the following in the *armlink Reference Guide*:

- *--autoat, --no_autoat on page 2-11*.

L9511E `Product definition file was not found`

The linker cannot find the required product license mapping (`.elmap`) files.

This might be because:

- The `.elmap` files are missing, for example if your installation is corrupt.
- The linker is looking for the `.elmap` files in the wrong place because the `ARM_PRODUCT_PATH` environment variable is incorrectly set.
- The linker is looking for a non-existant `.elmap` file because the `ARM_TOOL_VARIANT` environment variable is incorrectly set.

Chapter 4
ELF Image Converter Errors and Warnings

The following topic describes the error and warning messages for the ELF image converter, `fromelf`:

4.1 List of the fromelf error and warning messages

The error and warning messages for `fromelf` are:

Q0122E `Could not open file '<filename>': <reason>`

If `<reason>` is Invalid argument, this might be because you have invalid characters on the command-line. For example, on Windows you might have used the escape character \ when specifying a filter with an archive file:

```
fromelf --elf --strip=all t.a\(test*.o\) -o filtered/
```

On Windows, use:

```
fromelf --elf --strip=all t.a(test*.o) -o filtered/
```

See the following in the *fromelf User Guide*:

- *input_file* on page 4-47.

Q0128E `File i/o failure.`

This error can occur if you specify a directory for the `--output` command-line option, but you did not terminate the directory with a path separator. For example, `--output=my_elf_files/`.

See the following in the *fromelf User Guide*:

- *--output=destination* on page 4-55.

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. Object (`.o`) files and executable (`.axf`) files are in ELF format.

Q0147E `Failed to create Directory <dir>: <reason>`

If `<reason>` is File exists, this might be because you have specified a directory that has the same name as a file that already exists. For example, if a file called `filtered` already exists, then the following command produces this error:

```
fromelf --elf --strip=all t.a(test*.o) -o filtered/
```

The path separator character / informs fromelf that `filtered` is a directory.

See the following in the *fromelf User Guide*:

- *--output=destination* on page 4-55.

Q0171E `Invalid st_name index into string table <idx>.`

See Q0131E.

Q0172E `Invalid index into symbol table <idx>.`

See Q0131E.

Q0186E `This option requires debugging information to be present`

The `--fieldoffsets` option requires the image to be built with dwarf debug tables.

Q0425W `Incorrectly formed virtual function elimination header in file`

This might indicate a compiler fault. Contact your supplier.

Q0426E `Error reading vtable information from file`

This might indicate a compiler fault. Contact your supplier.

Q0427E `Error getting string for symbol in a vtable`

This might indicate a compiler fault. Contact your supplier.

Q0448W `Read past the end of the compressed data while decompressing section '<secname>' #<secnum> in <file>`

This might indicate an internal fault. Contact your supplier.

Q0449W `Write past the end of the uncompressed data buffer of size <bufsize> while decompressing section '<secname>' #<secnum> in <file>`

This might indicate an internal fault. Contact your supplier.

Q9511E `Product definition file was not found`

`fromelf` cannot find the required product license mapping (`.elmap`) files.

This might be because:

- The `.elmap` files are missing, for example if your installation is corrupt.
- `fromelf` is looking for the `.elmap` files in the wrong place because the `ARM_PRODUCT_PATH` environment variable is incorrectly set.
- `fromelf` is looking for a non-existant `.elmap` file because the `ARM_TOOL_VARIANT` environment variable is incorrectly set.

Chapter 5
Librarian Errors and Warnings

The following topic describes the error and warning messages for the ARM Librarian, `armar`:

- *List of the armar error and warning messages* on page 5-2.

5.1 List of the armar error and warning messages

The error and warning messages for `armar` are:

L6874W `Minor variants of archive member '<member>' include no base variant`

Minor variants of the same function exist within a library. Find the two equivalent objects and remove one of them.

L9511E `Product definition file was not found`

`armar` cannot find the required product license mapping (`.elmap`) files.

This might be because:

- The `.elmap` files are missing, for example if your installation is corrupt.
- `armar` is looking for the `.elmap` files in the wrong place because the `ARM_PRODUCT_PATH` environment variable is incorrectly set.
- `armar` is looking for a non-existant `.elmap` file because the `ARM_TOOL_VARIANT` environment variable is incorrectly set.

Chapter 6
Other Errors and Warnings

The following topics describe error and warning messages that might be displayed by any of the tools:

- *Internal faults and other unexpected failures* on page 6-2.
- *List of other error and warning messages* on page 6-3.

6.1 Internal faults and other unexpected failures

Internal faults indicate that the tool has failed an internal consistency check or has encountered some unexpected input that it could not deal with. They might point to a potential issue in the tool itself.

For example:

```
Internal fault: [0x76fd03:600448]
```

contains:

- The message description (`Internal fault`).
- A six hexadecimal digit fault code for the error that occurred (`0x76fd03`).
- The version number (60 is ARM Compiler 6.0).
- The build number (0448 in this example).

If you see an internal fault, contact your supplier.

To facilitate the investigation, try to send only the single source file or function that is causing the error, plus the command-line options used.

It might be necessary to preprocess the file (that is, to take account of files added with `#include`). To do this, pass the file through the preprocessor as follows:

```
armclang <options> –E sourcefile.c > PPsourcefile.c
```

where `<options>` are your normal compile switches, such as `-O2`, `-g`, `-I`, `-D`, but without `-c`.

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

```
armclang <options> -c PPsourcefile.c
```

and then provide the `PPsourcefile.c` file and the `<options>` to your supplier.

6.2 List of other error and warning messages

The following error and warning messages can be produced by any of the tools.

Note

When the message is displayed, the *X* prefixing the message number is replaced by the appropriate letter relating to the application. For example, the code `X3900U`, is displayed as `L3900U` by the linker when you have specified an unrecognized option.

***X*3900U** `Unrecognized option '<dashes><option>'.`

`<option>` is not recognized by the tool. This could be because of a spelling error or the use of an unsupported abbreviation of an option.

***X*3903U** `Argument '<argument>' not permitted for option '<option>'.`

Possible reasons include malformed integers or unknown arguments.