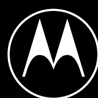
Freescale Semiconductor, Inc...

MCUez

Easy development software from the company that knows MCU hardware best

MCUez Linker User's Manual
MCUEZLNK/D
Rev. 1



MOTOROLA

Freescale Semiconductor, Inc.

MCUEZLNK/D
Rev. 1

MCUez Linker

User's Manual

Freescale Semiconductor, Inc.

MOTOROLA

Important Notice to Users

While every effort has been made to ensure the accuracy of all information in this document, Motorola assumes no liability to any party for any loss or damage caused by errors or omissions or by statements of any kind in this document, its updates, supplements, or special editions, whether such errors are omissions or statements resulting from negligence, accident, or any other cause. Motorola further assumes no liability arising out of the application or use of any information, product, or system described herein; nor any liability for incidental or consequential damages arising from the use of this document. Motorola disclaims all warranties regarding the information contained herein, whether expressed, implied, or statutory, *including implied warranties of merchantability or fitness for a particular purpose*. Motorola makes no representation that the interconnection of products in the manner described herein will not infringe on existing or future patent rights, nor do the descriptions contained herein imply the granting or license to make, use or sell equipment constructed in accordance with this description.

The computer program contains material copyrighted by Motorola, Inc., first published in 1997, and may be used only under a license such as the License For Computer Programs (Article 14) contained in Motorola's Terms and Conditions of Sale, Rev. 1/79.

Trademarks

This document includes these trademarks:

MCUez is a trademark of Motorola, Inc.

Microsoft Windows is a registered trademark of Microsoft Corporation.

WinEdit is a trademark of Wilson WindowWare.

© Motorola, Inc., and HIWARE AG., 1999; All Rights Reserved

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

List of Sections

Section 1. General Information 19

Section 2. Graphical User Interface (GUI). 23

Section 3. Files 37

Section 4. Operating Procedures 41

Section 5. Environment Variables 89

Section 6. Linker Messages 103

Index. 179

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

List of Sections

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Table of Contents

Section 1. General Information

1.1 Contents . . . 19
1.2 Introduction. . . . 19
1.3 Functional Description . . . 19
1.4 Features. . . . 20
1.5 Typographic Styles in This Manual . . . 20

Section 2. Graphical User Interface (GUI)

2.1 Contents . . . 23
2.2 Introduction. . . . 23
2.3 Linker Graphical User Interface . . . 24
2.3.1 Toolbar . . . 25
2.3.2 Content Area . . . 26
2.3.3 Status Bar . . . 27
2.3.4 Menu Bar . . . 27
2.3.4.1 File Menu . . . 27
2.3.4.2 Linker Menu . . . 31
2.3.4.3 View Menu . . . 34
2.3.4.4 Help Menu. . . . 34
2.3.5 Specifying the Input File . . . 34
2.3.5.1 Using the Command Line . . . 34
2.3.5.2 Using the Menu Entry File | Link 35
2.3.5.3 Using Drag and Drop . . . 35
2.3.6 Error Feedback. . . . 35

Section 3. Files

3.1 Contents . . . 37
3.2 Introduction. . . . 37
3.3 Parameter Files: Input. . . . 37
3.4 Absolute Files: Output . . . 38
3.5 Motorola S Files: Output . . . 38
3.6 Map Files . . . 38

Freescale Semiconductor, Inc.

Section 4. Operating Procedures

4.1 Contents 41
4.2 Introduction. 42
4.3 Parameter File 42
4.3.1 Syntax of the Parameter File 42
4.3.2 Mandatory Parameter File Linker Commands. 44
4.4 Linker Commands. 45
4.4.1 ENTRIES: List of Objects to Link with Application. 45
4.4.2 INIT: Specify Application Entry Point 47
4.4.3 LINK: Specify Name of Output File 47
4.4.4 MAIN: Specify Root Function. 49
4.4.5 MAPFILE: Configure Map File. 49
4.4.6 NAMES: List Files. 52
4.4.7 SEGMENTS: Define Memory Map 53
4.4.7.1 Segment Qualifier 55
4.4.7.2 Segment Alignment. 56
4.4.7.3 Segment Fill Pattern 59
4.4.8 PLACEMENT: Place Sections into Segments. 61
4.4.8.1 Specifying a List of Sections. 63
4.4.8.2 Specifying a List of Segments. 64
4.4.8.3 Predefined Sections 65
4.4.8.4 Allocating User-Defined Sections. 67
4.4.9 STACKSIZE: Define Stack Size 68
4.4.10 STACKTOP: Define Stack Pointer Initial Value 69
4.4.11 VECTOR: Initialize Vector Table 70
4.4.11.1 Initializing Vector Table in Linker Parameter File 72
4.4.11.2 Initializing Vector Table in Assembly Source File Using a Relocatable Section 74
4.4.11.3 Initializing Vector Table in Assembly Source File Using an Absolute Section 76
4.5 Smart Linking 79
4.5.1 Mandatory Linking from an Object 79
4.5.2 Mandatory Linking from All Objects Defined in a File 80
4.5.3 Switching Off Smart Linking for the Application 80
4.5.4 Linking an Assembly Application 80
4.5.5 Warning Messages 81
4.6 Program Startup 84
4.6.1 Startup Descriptor 84
4.6.2 User-Defined Startup Structure. 87
4.6.3 User-Defined Startup Routines 88

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Section 5. Environment Variables

5.1 Contents 89
5.2 Introduction. 90
5.3 Linker Options 90
5.3.1 -E 91
5.3.2 -H 91
5.3.3 -L 92
5.3.4 -M. 92
5.3.5 -N 92
5.3.6 -O 93
5.3.7 -S 93
5.3.8 -V 94
5.3.9 -W1. 94
5.3.10 -W2. 94
5.3.11 -Wmsg8x3 95
5.3.12 -WmsgFb[v|m] 95
5.3.13 -WmsgFi[v|m] 95
5.3.14 -WmsgNe 96
5.3.15 -WmsgNi 96
5.3.16 -WmsgNw 96
5.4 Setting Environment Variables in MCUez Shell 97
5.4.1 Path Variables 97
5.5 Variable Descriptions 97
5.5.1 GENPATH 98
5.5.2 OBJPATH 99
5.5.3 LIBPATH. 99
5.5.4 ABSPATH 100
5.5.5 TEXTPATH 100
5.5.6 SRECORD 101
5.5.7 ERRORFILE 102

Section 6. Linker Messages

6.1 Contents 103
6.2 Introduction. 108
6.3 Linker Messages Reference 108
6.3.1 L1: Unknown Message Occurred 109
6.3.2 L2: Message Overflow, Skipping <type> Messages 109
6.3.3 L64: Line Continuation Occurred in <FileName>. 109
6.3.4 L1000: <Command Name> not Found 110
6.3.5 L1001: <Command Name> Multiply Defined. 111

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

6.3.6 L1002: Command <Command Name> Overwritten by Option <Option Name> . . . 112
6.3.7 L1003: Only a Single SEGMENTS or SECTIONS Block is Allowed. . . . 113
6.3.8 L1004: <Separator> Expected . . . 113
6.3.9 L1005: Fill Pattern Will Be Truncated (>0xFF) . . . 114
6.3.10 L1006: <Token> not Allowed . . . 114
6.3.11 L1007: <Character> not Allowed in Filename (Restriction). . . . 115
6.3.12 L1008: Only Single Object Allowed at Absolute Address . . . 116
6.3.13 L1009: Segment Name <Segment Name> Unknown . . . 117
6.3.14 L1010: Section Name <section name> Unknown . . . 118
6.3.15 L1011: Incompatible Segment Qualifier: <Qualifier1> in Previous Segment and <Qualifier2> in <Segment Name>. . . 119
6.3.16 L1012: Segment is not Aligned on a <bytes> Boundary . . . 120
6.3.17 L1015: No Binary Input File Specified . . . 120
6.3.18 L1016: File <Filename> Found Twice in NAMES Block. . . . 121
6.3.19 L1037: ***** Linking of <parameter file> Failed **** . . . 121
6.3.20 L1038: Success. Executable File Written to <absfile> . . . 121
6.3.21 L1039: Limited Version. Too Many Objects or Code Linked. . . 122
6.3.22 L1050: Running <versiontype> . . . 122
6.3.23 L1052: User Requested Stop . . . 122
6.3.24 L1100: Segments <Segment1 Name> and <Segment2 Name> Overlap . . . 123
6.3.25 L1102: Out of Allocation Space in Segment <Segment Name> at Address <First Address Free>. . . . 124
6.3.26 L1103: <Section Name> not Specified in PLACEMENT Block . . . 125
6.3.27 L1104: Absolute Object <Object Name> Overlaps with Segment <Segment Name>. . . . 126
6.3.28 L1105: Absolute Object <object name> Overlaps with Another Absolute Allocated Object or with a Vector. . . 127
6.3.29 L1106: <Object Name> not Found . . . 128
6.3.30 L1107: <Object Name> not Found . . . 129
6.3.31 L1109: <Segment Name> Appears Twice in SEGMENTS Block. . . . 130
6.3.32 L1110: <Segment Name> Appears Twice in PLACEMENT Block . . . 131
6.3.33 L1111: <Section Name> Appears Twice in PLACEMENT Block . . . 132
6.3.34 L1112: <Section name> Section Has Segment Type <Segment Qualifier> (Illegal). . . . 132

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

6.3.35 L1113: <Section name> Section Has Segment Type <Segment Qualifier> (Illegal) 134
6.3.36 L1114: <Section Name> Section Has Segment Type <Segment Qualifier> (Initialization Problem) 135
6.3.37 L1115: Function <Function Name> not Found 137
6.3.38 L1118: Vector Allocated at Absolute Address <Address> Overlaps with Another Vector or an Absolute Allocated Object 138
6.3.39 L1119: Vector Allocated at Absolute Address <Address> Overlaps with Sections Placed in Segment <Segment Name> 139
6.3.40 L1120: Vector Allocated at Absolute Address <Address> Placed in Segment <Segment Name>, Which Has No READ_ONLY Qualifier 140
6.3.41 L1121: Out of Allocation Space at Address <Address> for .copy Section 140
6.3.42 L1122: Section .copy Must Be Last Section in Section List 141
6.3.43 L1123: Invalid Range Defined for Segment <Segment Name> — End Address Must Be Bigger Than Start Address 142
6.3.44 L1124: '+' or '-' Should Directly Follow Filename 143
6.3.45 L1125: In Small Memory Model, Code and Data Must Be Located on Bank 0 144
6.3.46 L1127: Object Allocated Outside of Segment Bounds (HC12) 145
6.3.47 L1200: Both STACKTOP and STACKSIZE Defined 146
6.3.48 L1201: No Stack Defined 147
6.3.49 L1202: Stack Cannot Be Allocated on More Than One Segment 148
6.3.50 L1203: STACKSIZE Command Defines a Size of <Size> But .stack Specifies a Stacksize of <Size> 149
6.3.51 L1204: STACKTOP Command Defines Initial Value of <Stack Top> But .stack Specifies Initial Value of <Initial Value> 151
6.3.52 L1205: STACKTOP Command Incompatible with .stack Being Part of List of Sections 152
6.3.53 L1206: Stack Overlaps with a Segment Which Appears in PLACEMENT Block 153
6.3.54 L1207: STACKSIZE Command is Missing 154
6.3.55 L1301: Cannot Open File <Filename> 155
6.3.56 L1302: File <Filename> not Found 155
6.3.57 L1303: <Filename> is not a Valid ELF File 156

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.58 L1304: <Filename> is not a Valid Hex File 156
6.3.59 L1305: <Filename> is not an ELF Format Object File (ELF Object File Expected) 156
6.3.60 L1309: Cannot Open <File> 157
6.3.61 L1400: Incompatible Processor: <Processor Name> in Previous Files and <Processor Name> in Current File. . . . 157
6.3.62 L1401: Incompatible Memory Model: <Memory Model Name> in Previous Files and <Memory Model Name> in Current File 157
6.3.63 L1403: Unknown Processor <Processor Constant>. 158
6.3.64 L1404: Unknown Memory Model <Memory Model Constant> 158
6.3.65 L1501: <Symbol Name> Cannot be Moved in Section <Section Name> (Invalid Qualifier <Segment Qualifier>) . . 159
6.3.66 L1502: <Object Name> Cannot be Moved from Section <Source Section Name> to Section <Destination Section Name> 160
6.3.67 L1503: <Object Name> (from file <Filename>) Cannot be Moved from Section <Source Section Name> to Section <Destination Section Name> 161
6.3.68 L1504: <Object Name> (from section <Section Name>) Cannot be Moved from Section <Source Section Name> to Section <Destination Section Name> 162
6.3.69 L1600: Main Function Detected in ROM Library 163
6.3.70 L1601: Startup Function Detected in ROM Library 163
6.3.71 L1620: Bad Digit in Binary Number 163
6.3.72 L1621: Bad Digit in Octal Number 163
6.3.73 L1622: Bad Digit in Decimal Number. 163
6.3.74 L1623: Number too Big 164
6.3.75 L1624: Ident too Long. Cut after 31 Characters 164
6.3.76 L1625: Comment not Closed 164
6.3.77 L1626: Unexpected End of File 164
6.3.78 L1627: PRESTART Command not Supported Yet 165
6.3.79 L1628: HEXFILE Command not Supported Yet 165
6.3.80 L1629: START_DATA Command not Supported Yet 165
6.3.81 L1700: File <Filename> Should Contain DWARF Information . 165
6.3.82 L1701: Startup Data Structure is Empty 166
6.3.83 L1800: Read Error in <File> 166
6.3.84 L1803: Out of Memory in <Function Name> 166
6.3.85 L1804: No ELF Section Header Table Found in <Filename>. . . 166
6.3.86 L1806: ELF File <Filename> Appears to be Corrupted 167

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

6.3.87 L1808: String Overflow in <Function Name>, Contact Vendor . . . 167
6.3.88 L1809: Section <Section Name> Located in a Segment with Invalid Qualifier . . . 167
6.3.89 L1811: Symbol <Symbol Number> - < Symbol Name> Duplicated in <First Filename> and <Second Filename> . . . 167
6.3.90 L1818: Symbol <Symbol Number> - < Symbol Name> Duplicated in <First Filename> and <Second Filename> . . . 168
6.3.91 L1820: Weak Symbol <Symbol Name> Duplicated in <First Filename> and <Second Filename>. . . 168
6.3.92 L1821: Symbol <id1> Conflicts with <id2> in File <File> (Same Code) . . . 168
6.3.93 L1822: Symbol <Symbol Name> in File <Filename> is Undefined . . . 168
6.3.94 L1823: External Object <Symbol Name> in <Filename> Created by Default . . . 169
6.3.95 L1824: Invalid Mark Type for <Ident> . . . 169
6.3.96 L1826: Can't Read File. <Filename> is not an ELF Library Containing ELF Objects (ELF Objects Expected). . . 169
6.3.97 L1902: <Cmd> Command not Supported . . . 169
6.3.98 L1903: Unexpected Symbol in Link Parameter File . . . 170
6.3.99 L1905: Invalid Section Attribute for Program Header . . . 170
6.3.100 L1906: Fixup Out of Buffer (<Obj> Referenced at Offset <Address>). . . 170
6.3.101 L1907: Fixup Overflow in <Object>, Type <objType> at Offset <Address> . . . 170
6.3.102 L1908: Fixup Error in <Object>, Type <objType> at Offset <Address> . . . 171
6.3.103 L1910: Invalid Section Attribute for Program Header . . . 171
6.3.104 L1911: Program Header End is not Aligned on the End of a Section . . . 171
6.3.105 L1912: Object <obj> Overlaps with Another (last addr: <addr>, Object Address: <objadr> . . . 171
6.3.106 L1913: Object Filler Overlaps with Something Else . . . 171
6.3.107 L1914: Invalid Object: <Object>. . . 172
6.3.108 L1915: Gap in <Ident> at <address> before <Object> is too Big . . . 172
6.3.109 L1916: Section Name <Section> is too Long. Name is Cut to 90 Characters Length . . . 172
6.3.110 L1919: Duplicate Definition of <Object> in Library File(s) <File1> and/or <File2> Discarded . . . 172

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.

6.3.111 L1921: Marking: Too Many Nested Procedure Calls 173
6.3.112 L1922: File <filename> Has DWARF Data of Different Version, DWARF Data may not be Generated. 173
6.3.113 L1927: Fixups for DWARF Section <sectionname> not Correctly Generated 173
6.3.114 L1928: Limitation: Code Size <num> 173
6.3.115 L1929: Limitation: Too many Mections (<num>). 174
6.3.116 L1930: Unknown Fixup Type in <ident>, Type <type>, at Offset <offset> 174
6.3.117 L1931: Program Header Begin is not Aligned on the Beginning of a Section 174
6.3.118 L1932: Program Header Overflow in <name> at <index> 174
6.3.119 L1933: ELF: <details> Warning 174
6.3.120 L1934: ELF: <details> Error 175
6.3.121 L1936: ELF Output: <details> Error 176
6.3.122 L1938: Type Clash in Segment (Corrupt Object: <name>). 177
6.3.123 L4000: Could not Open Object File (<objFile>) in NAMES List 177
6.3.124 L4001: Link Parameter File <PRMFile> not Found 177
6.3.125 L4002: NAMES Section was not Found in Linker Parameter File <PRM File> 177
6.3.126 L4004: Linking <PRM File> as ELF/DWARF Format Link Parameter File. 178
6.3.127 L4005: Illegal File Format of Object File (<objFile>) in NAMES List 178
6.3.128 L4006: Failed to Create Temporary File 178
6.3.129 L4007: Include File Nesting too Deep in Link Parameter File . . 178
6.3.130 L4008: Include File <includefile> not Found 178

Index

Index. 179

For More Information On This Product,
Go to: www.freescale.com

List of Figures

Figure	Title	Page
2-1	MCUez Shell	24
2-2	MCUez Linker Tip of the Day Window	24
2-3	MCUez Linker Main Window	25
2-4	MCUez Linker Toolbar	26
2-5	MCUez Linker Status Bar	27
2-6	Configuration Dialog Box	29
2-7	Save Configuration Dialog Box	30
2-8	Option Settings Dialog Box	31
2-9	Message Settings Dialog Box	32
3-1	Related Linker Files and Location	39
5-1	Linker Command Line	90

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

List of Figures

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

List of Tables

Table	Title	Page
2-1	Menu List	27
2-2	Option Groups	31
2-3	Message Group Definitions	33
4-1	ENTRIES Block Supported	46
4-2	Map File Sections	50
4-3	Map File Options	51
4-4	Segment Qualifier Descriptions	55
4-5	Segment Alignment Rule Format	56
4-6	Segment Alignment Items List	58
4-7	VECTOR Command Syntax	71
4-8	Setting Startup Descriptor Flags	85

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

List of Tables

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Section 1. General Information

1.1 Contents

1.2 Introduction. 19
1.3 Functional Description 19
1.4 Features. 20
1.5 Typographic Styles in This Manual 20

1.2 Introduction

This manual describes Motorola's MCUez linker. The linker merges the various object files of an application into one file, an absolute file (*.abs*). The file is termed an absolute file because it contains absolute code (not relocatable code) that can be loaded into the target and burnt onto an EPROM (erasable programmable read-only memory) using the MCUez debugger.

1.3 Functional Description

Linking is the process of assigning memory to all global objects (functions, global data, strings, and initialization data) needed for a given application and combining these objects into a format suitable for downloading into a target system or an emulator.

The MCUez linker is a smart linker, only linking objects actually used by the application. Various optimization capabilities ensure low memory requirements for the linked program. Unused functions and variables will not occupy memory in the target system. Also, initialization of global variables is stored in compact form and memory is reserved only once for identical strings.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

1.4 Features

The most important features supported by the MCUez linker are:

- Complete control over placement of objects in memory — It is possible to allocate different groups of functions or variables to different memory areas (segmentation).
- Initialization of vectors

When linking high-level language modules (C, C++, etc.), the linker supports these features:

- User-defined startup — The application startup script is in a separate file written in "inline assembly" and can be easily modified. The startup file is named *startup.c* or *startup.o*. This is a generic filename that needs to be replaced by the real target startup file given in the `\LIB\COMPILER` directory. Usually, the filename is *start*.c* or *start*.o*, where `*` is the name or part of the MCU name and might also contain an abbreviation of the memory model.
- Mixed language linking — Modula-2, assembly, and C object files can be mixed in the same application.

1.5 Typographic Styles in This Manual

These typographic conventions are used in this manual:

- **Bold face** type is used for literal strings that must be used exactly as shown in the example and for the names of menus, windows, dialog boxes, icons, and buttons.
- `Courier` type face is used for all C-code program listings, command lines, and directories..
- *Italics* are used where the string is a place holder that may be substituted for a string of the user's own design.
- Variable user inputs are in *`Courier`* italics.
- Filenames are in italics with all lower case letters, for example, *proj.ext.*

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

These styles are used in this manual to define notational conventions:

- **Numeric constants** — Numeric constants are displayed in the C language format. Constants that are in the 0x format are hexadecimal. Constants that have no prefix are assumed to be decimal. The notation k, unless to denote a frequency setting in kilohertz, defines a number multiplied by 1024.
- **Function prototypes** — Structures and function call descriptions are given in terms of the C language. This does not limit the implementation of calling programs to C, but it is the calling routine's responsibility to provide the correct link to these routines.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Section 2. Graphical User Interface (GUI)

2.1 Contents

2.2 Introduction. 23

2.3 Linker Graphical User Interface 24
2.3.1 Toolbar 25
2.3.2 Content Area 26
2.3.3 Status Bar 27
2.3.4 Menu Bar 27
2.3.4.1 File Menu 27
2.3.4.2 Linker Menu 31
2.3.4.3 View Menu 34
2.3.4.4 Help Menu. 34
2.3.5 Specifying the Input File 34
2.3.5.1 Using the Command Line 34
2.3.5.2 Using the Menu Entry File | Link 35
2.3.5.3 Using Drag and Drop 35
2.3.6 Error Feedback. 35

2.2 Introduction

The MCUez linker is a Microsoft Windows® compatible application that uses a standard graphical user interface (GUI). This section describes:

- The MCUez linker graphical user interface
- How to start the linker

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

2.3 Linker Graphical User Interface

Click the **ezLink** icon on the **MCUez Shell** toolbar to run the linker (see **Figure 2-1**).

Figure 2-1. MCUez Shell

When the linker is started, a standard **Tip of the Day** window (see **Figure 2-2**) containing features about the linker is displayed.

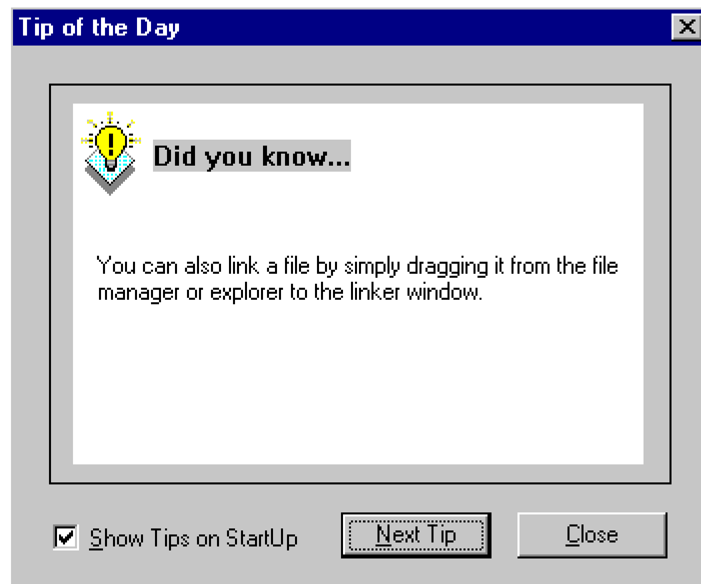


Figure 2-2. MCUez Linker Tip of the Day Window

Click **Next Tip** to view more information about the linker. Click **Close** to close the **Tip of the Day** dialog. To disable the tips window when the linker is started, uncheck **Show Tips on StartUp**. Select **Help | Tip of the Day ...** then check **Show Tips on StartUp** to re-enable the tips window.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Figure 2-3 is an example of the main linker window. The linker window provides a menu bar, toolbar, content area, and status bar.

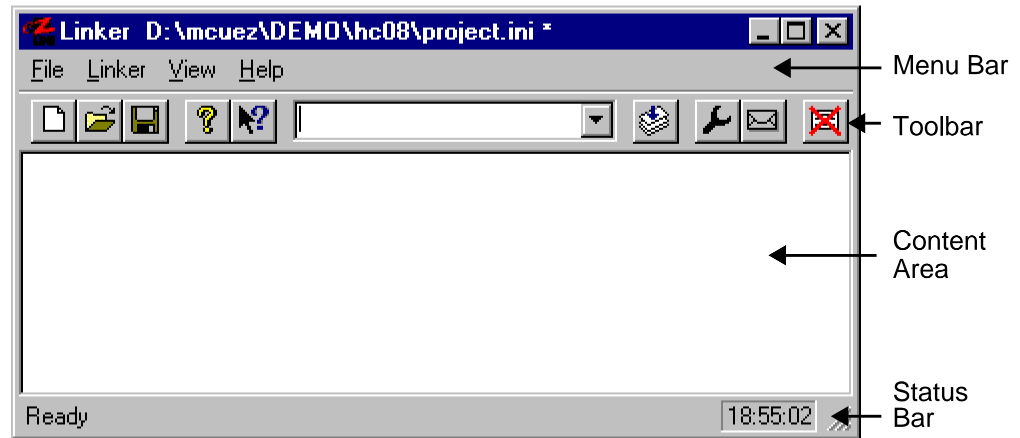


Figure 2-3. MCUez Linker Main Window

2.3.1 Toolbar

Figure 2-4 illustrates the linker toolbar. Note that:

- The **New**, **Load**, and **Save** buttons are linked to the corresponding entries of the **File** menu.
- The **?** and **Context Help** buttons correspond with entries in the **Help** menu.
- The command line is for entering linker commands or selecting previously entered commands. Click the **Link** button to execute a command.
- The **Options** button opens the **Options** dialog box.
- The **Message** button opens the **Message Settings** dialog box.
- The **Clear** button clears all information in the content area.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

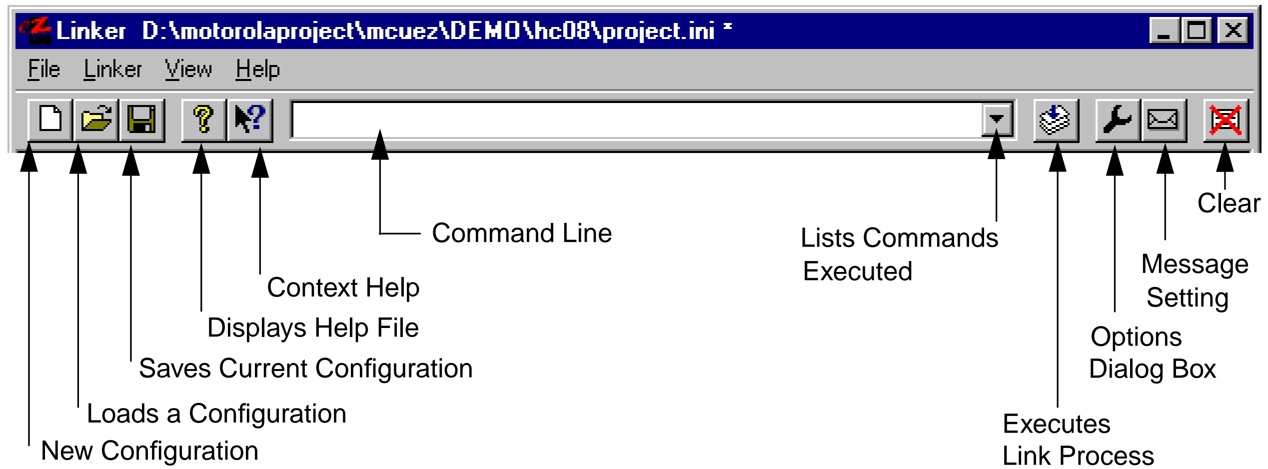


Figure 2-4. MCUez Linker Toolbar

2.3.2 Content Area

The content area displays information about the link session. This information consists of:

- The name of the *.prm* (parameter) file being linked
- The name (including full path) of the files building the application
- A list of error, warning, and information messages

Additional information is available for all lines related to errors. Double click on a line to open the related file in the project editor or select the line and click the right mouse button to open a menu. If the menu contains an **Open...** entry for the selected line, it will open the related file and highlight the line that has an error. See **2.3.5 Specifying the Input File**.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

2.3.3 Status Bar

Figure 2-5 shows the linker status bar.

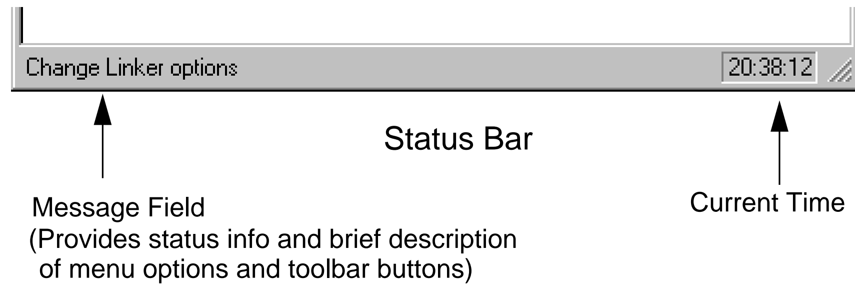


Figure 2-5. MCUez Linker Status Bar

2.3.4 Menu Bar

The menus listed in Table 2-1 are available on the menu bar. Refer to Figure 2-3 for an illustration of the menu bar.

Table 2-1. Menu List

Menu Entry	Description
File	Linker configuration file management
Linker	Linker option settings
View	Linker window settings
Help	Standard windows help menu

2.3.4.1 File Menu

The **File** menu provides options to manage project configuration files. Typical linker settings in the *project.ini* file consist of:

- Settings specified in the **Options Settings** dialog box
- Editor associated with the linker

Linker configuration information is stored in the *project.ini* file under the [LINKER] and [EDITOR] sections.

Freescale Semiconductor, Inc.

Project configuration files are ASCII files with a *.ini* extension. The user can define as many of these files as needed for a project. Switch between different configuration files by selecting **File | Load Configuration** and **File | Save Configuration** or by clicking the corresponding toolbar buttons.

Select **File | Linker** to open a standard **Open File** dialog box that displays a list of all *.prm* files in the project directory. Select the input file to be linked and click **OK**.

Select **File | New/Default Configuration** to reset the linker settings to the values contained in the current *project.ini* file.

Select **File | Load Configuration** to open the **Open File** dialog box and display a list of all *.ini* files in the project directory. Select a configuration file to be loaded.

Select **File | Save Configuration** to store the current settings in the project configuration file displayed on the window title bar.

Select **File | Save Configuration as ...** to open a standard **Save As** dialog box and display a list of all *.ini* files. Specify the name and location of the configuration file to store the current settings. Click **OK**.

Select **File | Configuration ...** to specify an editor to be used for error feedback and additional information to be saved in the configuration file. Click **Save** in the **Save Configuration** tab to instantly save settings in the [EDITOR] section of the *project.ini* file. See **Figure 2-6**.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

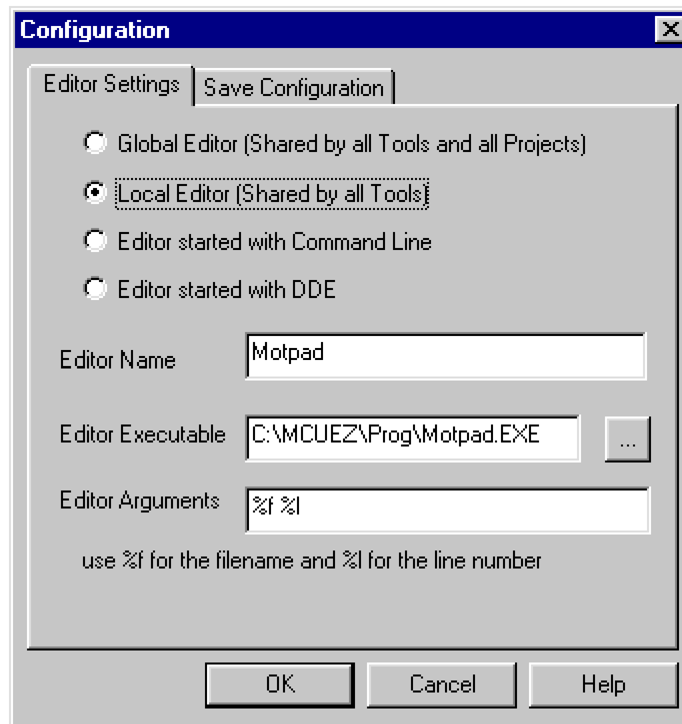


Figure 2-6. Configuration Dialog Box

Some editors may contain modifiers. The %f modifier refers to the filename (including path) where an error has been detected. The %l modifier refers to the line number in the file that contains an error.

Check the *MCUez Installation and Configuration User's Manual*, Motorola document order number MCUEZINS/D, to define the command line used to start an editor when an error occurs.

Error messages are listed in the linker window. To open the editor, double click on a line that refers to the file that contains an error.

CAUTION: *The `%l` modifier can be used only with an editor that can be started with a line number as a parameter. Editors such as WinEdit™ version 3.1 or lower and Notepad do not allow this modifier.*

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

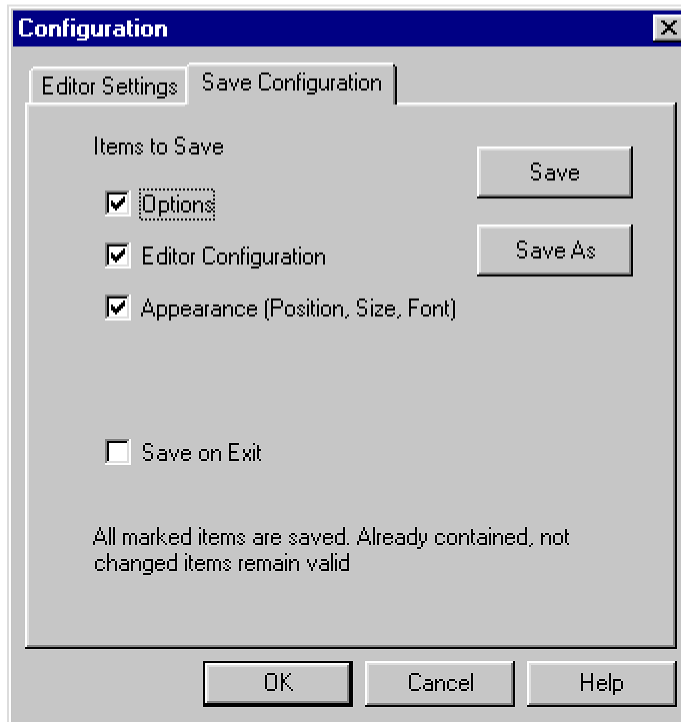


Figure 2-7. Save Configuration Dialog Box

The **Save Configuration** tab of the **Configuration** dialog is used to save all user-defined settings to the *project.ini* file. Under **Items to Save**, check the items to be saved and uncheck items to not be saved.

Options refers to settings specified in the **Option Settings** dialog box. This dialog is accessed by the **Linker | Options** menu selection or the equivalent toolbar button.

Editor Configuration refers to the settings specified in the **Editor Settings** tab.

Appearance refers to the position and size of the linker window and the font specified in the **View | Log | Change Font** menu selection.

Check **Save on Exit** to save settings when exiting the MCUez linker.

Click the **Save** button to instantly save the settings or click **Save As** to save settings to a new project configuration file.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

2.3.4.2 Linker Menu

The **Linker** menu allows the user to define file and message options (**Linker | Options**) and recategorize the class assigned to linker messages (**Linker | Messages**).

The **Option Settings** dialog box (**Figure 2-8**) allows the user to set and reset linker options. Available options are arranged in different groups. **Table 2-2** describes the option groups.

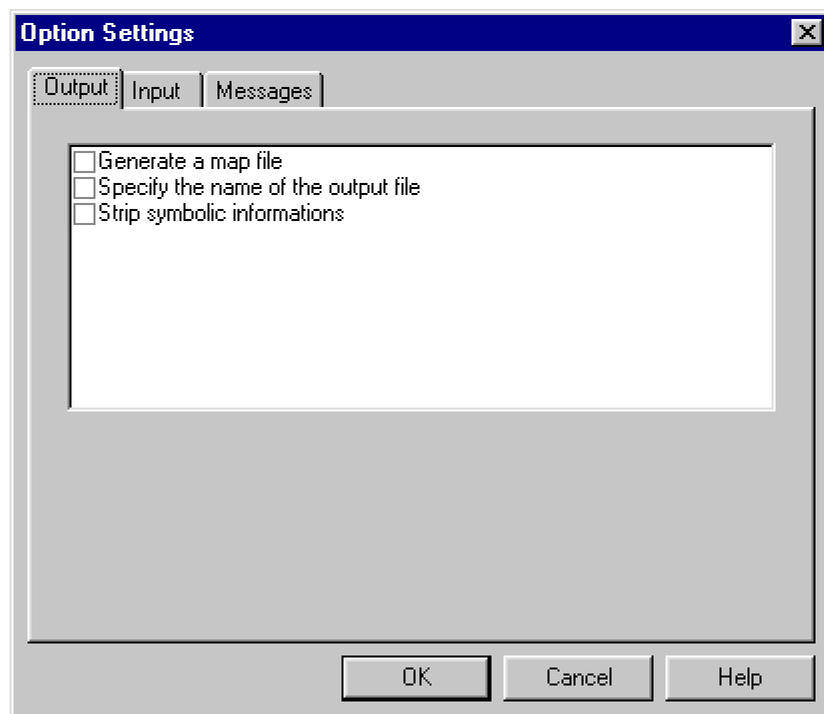


Figure 2-8. Option Settings Dialog Box

Table 2-2. Option Groups

Option Group	Description
Output	Lists options related to generated output files (type of files to be generated)
Input	Lists options related to input files
Messages	Lists options that control generation of error messages

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

NOTE: *When options requiring additional parameters are selected, an edit box or subwindow appears in the dialog box.*

Check options to be defined and enter additional information, if applicable to that option. Click **OK** to activate selected options. However, save settings to the *project.ini* file by using the **Configuraton** dialog as mentioned in **2.3.4.1 File Menu**.

Figure 2-9 shows the **Message Settings** dialog box.

This dialog box allows the user to map linker messages to a different message class. For example, a linker message such as “L1404: Unknown memory model <Model>” can be specified as a warning message instead of an error message.

Table 2-3 identifies and defines each message class.

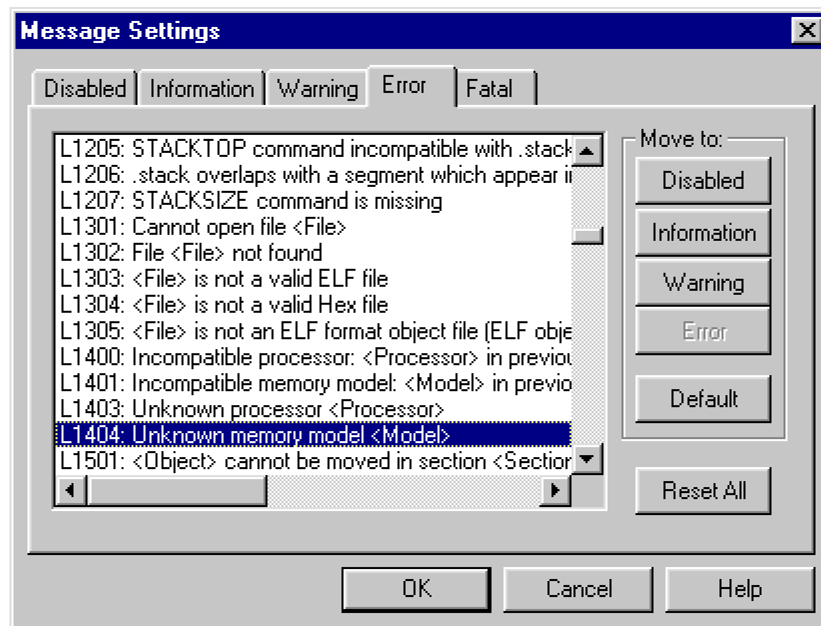


Figure 2-9. Message Settings Dialog Box

Freescale Semiconductor, Inc.

Table 2-3. Message Group Definitions

Message Group	Description
Disabled	Lists all disabled messages. Messages displayed in this list box will not be generated by the linker.
Information	Lists all information messages. Information messages depict action taken by the linker.
Warning	Lists all warning messages. When such a message is generated, linking continues and an absolute file is generated.
Error	Lists all error messages. When such a message is generated, linking of the input application continues but no absolute file will be generated.
Fatal	Lists all fatal error messages. When such a message is generated, linking stops immediately.

Each message has a character (L for linker message) followed by a 4- to 5-digit number. This number allows the message to be easily searched in the manual or online help.

The user can map messages to different classes by using the buttons located on the right side of the dialog box. Each button refers to a message class. To change the class associated with a message, select the message in the list box and click the button associated with the class where the message is to be moved.

The **Default** button will reset selected messages to their default class. The **Reset All** button will reset all messages to their default class.

Example: To define the warning message “L1201: No stack defined” as an error message:

1. Click the **Warning** tab to display the list of all warning messages.
2. Click on the string **L1201: No stack defined** in the list box to select the message.
3. Click the **Error** button to define this message as an error message.

Click **Yes** to validate the change or **No** to retain the previous mapping.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

2.3.4.3 View Menu

This menu enables the user to customize the linker window:

- Select **View | Toolbar** to display or hide the toolbar.
- Select **View | Status Bar** to display or hide the status bar.
- Select **View | Log ... | Change Font** to open a standard **Font Selection** dialog box. Options selected in this dialog are applied to information displayed in the content area.
- Select **View | Log ... | Clear Log** to clear the content area.

2.3.4.4 Help Menu

This menu consists of these selections:

Select **Help | Tip of the Day** to display the tips dialog box.

Select **Help | Help Topics** to open the help file.

Select **Help | About** to display version information and the current working directory.

2.3.5 Specifying the Input File

The input file to be linked can be specified in several ways. During the link session, the options will be set according to the configuration set by the user in the **Option Settings** dialog box. Before linking a file, ensure that a project directory is associated with the linker.

2.3.5.1 Using the Command Line

Linking a new file — A new filename and additional linker options can be entered on the command line. Click the **Link** or **Enter** buttons to link the specified file.

Linking a file that has already been linked — Previously linked files can be displayed by selecting the arrow button on the right side of the command line. Select a file and click the **Link** button.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

2.3.5.2 Using the Menu Entry File | Link ...

Select **File | Link...** to open a standard **Open File** dialog box. Select an input file and click **OK** to link the selected file.

2.3.5.3 Using Drag and Drop

A filename can be dragged from another program (for example, File Manager) and dropped into the linker window. The dropped file will be linked as soon as the mouse button is released.

2.3.6 Error Feedback

After a parameter (*.prm*) file has been linked, any error or warning messages will have this format:

‘>> <FileName>, line <line number>, col <column number>,
pos <absolute position in file> <Portion of code generating the problem>
<message class> <message number>: <Message string>‘

Example:

```
>> in "placemen\tstpla8.prm", line 23, col 0, pos 668
fpm_data_sec INTO  MY_RAM2;

END

ERROR L1110: MY_RAM2 appears twice in PLACEMENT block
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Section 3. Files

3.1 Contents

3.2 Introduction 37

3.3 Parameter Files: Input 37

3.4 Absolute Files: Output 38

3.5 Motorola S Files: Output 38

3.6 Map Files 38

3.2 Introduction

This section describes the files used and generated by the MCUez linker.

3.3 Parameter Files: Input

The linker parameter file is an ASCII text file that is required for each application. It contains linker commands that define the linking process. No special extension is required. However, it is suggested that parameter filenames have the extension *.prm*. Parameter files are searched for in the project directory and then in the GENPATH directories.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

3.4 Absolute Files: Output

After a successful link session, the linker generates an absolute file containing the target code as well as some debugging information. This file is written to the directory assigned to the environment variable ABSPATH. If the variable contains more than one path, the absolute file is written to the first directory specified. If this variable is not set, the absolute file is written to the directory where the parameter file was found. Absolute files always get the extension *.abs*.

3.5 Motorola S Files: Output

After a successful link session, the linker generates a Motorola S-record file, which can be burnt into an EPROM. This file contains information stored in all READ_ONLY sections in the application. The extension for the generated Motorola S-record file depends on the setting of the SRECORD variable.

- If SRECORD = S1, the Motorola S-record file extension is *.s1*.
- If SRECORD = S2, the extension is *.s2*.
- If SRECORD = S3, the extension is *.s3*.
- If SRECORD is not set, the Motorola S-record file extension is *.sx*.

This file is written to the directory specified in the environment variable ABSPATH. If the variable contains more than one path, the S-record file is written to the first directory specified. If this variable is not set, the S-record file is written to the directory where the parameter file was found.

3.6 Map Files

After a successful link session, the linker generates a map file containing information about the link process (**Figure 3-1**). This file is written to the directory specified in the environment variable TEXTPATH. If the variable contains more than one path, the map file is written to the first directory specified. If this variable is not set, the map file is written to the directory where the parameter file was found. map files always get the extension *.map*.

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.

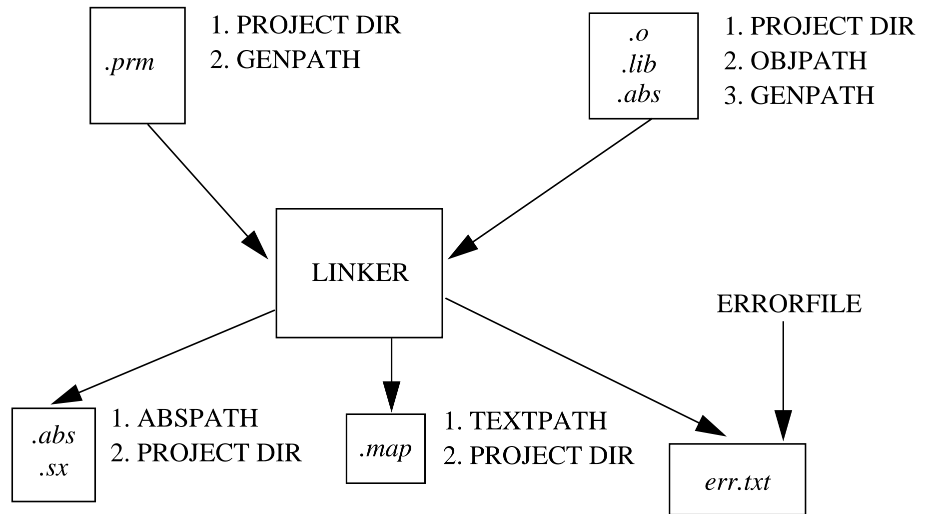


Figure 3-1. Related Linker Files and Location

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Section 4. Operating Procedures

4.1 Contents

4.2 Introduction 42

4.3 Parameter File 42
4.3.1 Syntax of the Parameter File 42
4.3.2 Mandatory Parameter File Linker Commands 44

4.4 Linker Commands 45
4.4.1 ENTRIES: List of Objects to Link with Application 45
4.4.2 INIT: Specify Application Entry Point 47
4.4.3 LINK: Specify Name of Output File 47
4.4.4 MAIN: Specify Root Function 49
4.4.5 MAPFILE: Configure Map File 49
4.4.6 NAMES: List Files 52
4.4.7 SEGMENTS: Define Memory Map 53
4.4.7.1 Segment Qualifier 55
4.4.7.2 Segment Alignment 56
4.4.7.3 Segment Fill Pattern 59
4.4.8 PLACEMENT: Place Sections into Segments 61
4.4.8.1 Specifying a List of Sections 63
4.4.8.2 Specifying a List of Segments 64
4.4.8.3 Predefined Sections 65
4.4.8.4 Allocating User-Defined Sections 67
4.4.9 STACKSIZE: Define Stack Size 68
4.4.10 STACKTOP: Define Stack Pointer Initial Value 69
4.4.11 VECTOR: Initialize Vector Table 70
4.4.11.1 Initializing Vector Table in Linker Parameter File 72
4.4.11.2 Initializing Vector Table in Assembly Source File Using a Relocatable Section 74
4.4.11.3 Initializing Vector Table in Assembly Source File Using an Absolute Section 76

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

4.5 Smart Linking 79
4.5.1 Mandatory Linking from an Object 79
4.5.2 Mandatory Linking from All Objects Defined in a File 80
4.5.3 Switching Off Smart Linking for the Application 80
4.5.4 Linking an Assembly Application 80
4.5.5 Warning Messages 81

4.6 Program Startup 84
4.6.1 Startup Descriptor 84
4.6.2 User-Defined Startup Structure. 87
4.6.3 User-Defined Startup Routines 88

4.2 Introduction

This section provides operating procedures for the MCUez linker.

4.3 Parameter File

The linker parameter file (*.prm*) is an ASCII text file that is required for each application. It contains linker commands that define the linking process. This section describes the parameter file in detail, giving examples that can be used as templates. Also, refer to the example parameter files included during the MCUez installation.

4.3.1 Syntax of the Parameter File

This is the EBNF (Extended Backus-Naur Form) syntax of the parameter file:

```
ParameterFile={Command}
Command= LINK NameOfABSFile
|  NAMES ObjFile {ObjFile} END
|  SEGMENTS {SegmentDef} END
|  PLACEMENT {Placement} END
|  (STACKTOP | STACKSIZE) exp
|  MAPFILE MapSecSpecList
|  ENTRIES EntrySpec {EntrySpec } END
|  VECTOR (InitByAddr | InitByNumber)
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.


```
|  INIT FuncName
|  MAIN FuncName
NameOfABSFile= FileName
ObjFile= FileName [”+”]
ObjName= Ident
QualIden = FileName “:” Ident
FuncName= ObjName | QualIdent
MapSecSpecList= MapSecSpec “,” { MapSecSpec }
EntrySpec= [FileName“:”] (* | ObjName)
MapSecSpec= ALL | NONE | TARGET | FILE | STARTUP | SEC_ALLOC |
    OBJ_ALLOC | OBJ_DEP | OBJ_UNUSED | COPYDOWN | STATISTIC
SegmentDef= SegmentName “=“ SegmentSpec “;”
SegmentName= Ident
SegmentSpec= StorageDevice Range [Alignment] [FILL CharacterList]
StorageDevice= READ_ONLY | READ_WRITE | PAGED | NO_INIT
Range= exp (TO | SIZE) exp
Alignment= ALIGN [exp] {“[“ObjSizeRange“:” exp”]”}
ObjSizeRange=  Number | Number TO Number | CompareOp Number
CompareOp= (“<“ | “>=“ | “>“ | “>=“)
CharacterList= HexByte { HexByte}
Placement= SectionList INTO SegmentList “;”
SectionList= SectionName {“,” SectionName}
SectionName=Ident
SegmentList= Segment {“,” Segment}
Segment= SegmentName | SegmentSpec
InitByAddr= ADDRESS Address Vector
InitByNumber= VectorNumber Vector
Address= Number
VectorNumber= Number
Vector= (FuncName [OFFSET exp] | exp) [“,” exp]
Ident= <any C style identifier>
FileName= <any file name>
exp= Number
Number= DecimalNumber | HexNumber | OctalNumber
HexNumber= 0xHexDigit{HexDigit}
DecimalNumber= DecimalDigit{DecimalDigit}
HexByte= HexDigit HexDigit
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
HexDigit= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
A | B | C | D | E | F | a | b | c | d | e | f
DecimalDigit= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
```

Comments may appear anywhere in a parameter file, except where filenames are expected. Use either C-style comments (/* */) or C++(//) style comments.

Filenames should not contain paths. This keeps sources portable. Otherwise, the sources are copied to another directory and the linker might not find all files needed. The linker uses the paths in the environment variables GENPATH, OBJPATH, LIBPATH, TEXTPATH, and ABSPATH to determine where to look for files and write output files.

Default predefined sections are named `.data`, `.text`, `.stack`, `.copy`, `.rodata1`, `.rodata`, `.startData` and `.init`.

NOTE: *The order of commands in the parameter file does not matter. However, ensure that the SEGMENTS block is specified before the PLACEMENT block.*

4.3.2 Mandatory Parameter File Linker Commands

A linker parameter file always contains at least the `LINK`, `NAMES`, and `PLACEMENT` commands. All other commands are optional. This example shows the minimal parameter file:

```
LINK mini.abs   /* Name of resulting ABS file */
NAMES
  mini.o startup.o   /* Files to link */
END
STACKSIZE 0x20        /* in bytes */
PLACEMENT
  .text INTO READ_ONLY  0xA00 TO 0xBFF;
  .data INTO READ_WRITE 0x800 TO 0x8FF;
END
```

The first placement statement

```
.text INTO READ_ONLY 0xA00 TO 0xBFF;
```

reserves the address range from `0xA00` to `0xBFF` for allocation of read-only objects (hence the qualifier `READ_ONLY`). The `.text` section includes all linked

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

functions, constant variables, string constants, and initialization parts of variables copied to RAM at startup.

The second placement statement

```
.data INTO READ_WRITE 0x800 TO 0x8FF;
```

reserves the address range from `0x800` to `0x8FF` for allocation of variables.

4.4 Linker Commands

This section describes all linker commands.

4.4.1 ENTRIES: List of Objects to Link with Application

Syntax: `ENTRIES [Filename:] (*|objName)`

Description: The ENTRIES block is optional in a parameter (PRM) file.

Use the ENTRIES block to list objects (referenced or not) that are always linked with the application. All objects referenced within these objects will also be linked with the application.

If a filename specified in the ENTRIES block is not present in the NAMES block, the filename will be inserted in the list of binary files building the application. The file specified in the ENTRIES block also may be present in the NAMES block. Names of absolute, ROM library, or library files are not allowed in the ENTRIES block.

Table 4-1 identifies the syntax supported in the ENTRIES block.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Table 4-1. ENTRIES Block Supported

Syntax	Meaning
<Object Name>	The specified global object will be linked with the application.
<File Name>:<Object Name>	The local object defined in the binary file will be linked with the application. This notation is only valid when referring to a symbol defined in a high-level language (ANSI C or C++) module.
<File Name>:*	All objects defined within the specified file will be linked with the application.
*	All objects will be linked with the application. This switches off smart linking for the application.

Symbols defined in an assembly module, which are used as additional entry points, must be published (specified in XDEF directive).

Example:

```
NAMES
   startup.o
END

ENTRIES
   fibo.o:*
END
```

In the previous example, the application is built from the files *fibo.o* and *startup.o*.

Example:

```
NAMES
   fibo.o startup.o
END

ENTRIES
   fibo.o:*
END
```

In the previous example, the application is built from the files *fibo.o* and *startup.o*. The file *fibo.o* specified in the NAMES block is the same as the one specified in the ENTRIES block.

Freescale Semiconductor, Inc.

NOTE: *It is strongly recommended to avoid switching smart linking off when the ANSI library is linked with the application. The ANSI library contains the implementation of all run time and standard functions. This generates a large amount of code, which is not required by the application.*

4.4.2 INIT: Specify Application Entry Point

Syntax:

```
INIT  FuncName
```

Description: The `INIT` command is recommended for an assembly application and can only be specified once in the PRM file. This command defines the entry point for the application. When `INIT` is not specified in the PRM file, the linker looks for a function named `_Startup` and uses it as the application entry point. If an `INIT` command is specified in the PRM file, the linker uses the specified function as the application entry point.

Specify any static or global function as an entry point.

Example:

```
INIT MyGlobStart /* Specify a global variable as
                    application entry point.*/
INIT myFile.o:myLocStart /* Specify a local
             variable as application entry point.*/
```

Local symbols defined in an assembly module cannot be specified as an entry point for an application.

4.4.3 LINK: Specify Name of Output File

Syntax:

```
LINK  <NameOfABSFile>
```

Description: The `LINK` command defines the file to be generated by the link session. This command is mandatory and can be specified only once in a PRM file.

After a successful link session, the output file is created. If the environment variable ABSPATH is defined, the absolute file is generated in the first directory assigned to the variable. Otherwise, it is written to the directory where the parameter file was found. If a file with this name already exists, it is overwritten.

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

A successful link session also creates a map file with the same base name as the absolute file with extension *.map*. If the environment variable TEXTPATH is defined, the map file is generated in the first directory assigned to the variable. Otherwise, it is written to the directory where the parameter file was found. If a file with this name already exists, it is overwritten.

A successful link session also creates an S-record file with the same base name as the absolute file with extension *.Sx*. If the environment variable ABSPATH is defined, the S-record file is generated in the first directory assigned to the variable. Otherwise, it is written to the directory where the parameter file was found.

If a file with this name already exists, it is overwritten.

Example:

```
LINK fibo.abs

NAMES  fibo.o startup.o END
SEGMENTS
    MY_RAM = READ_WRITE 0x1000 TO 0x18FF;
    MY_ROM = READ_ONLY  0x8000 TO 0x8FFF;
    MY_STK = READ_WRITE 0x1900 TO 0x1FFF;
PLACEMENT
    DEFAULT_ROM    INTO  MY_ROM;
    DEFAULT_RAM    INTO  MY_RAM;
    SSTACK         INTO  MY_STK;
END
VECTOR ADDRESS 0xFFFE _Startup /*set reset vector*/
```

The files *fibo.abs*, *fibo.sx*, and *fibo.map* are generated after a successful link session.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

4.4.4 MAIN: Specify Root Function

Syntax:

```
MAIN  FuncName
```

Description: The `MAIN` command is optional. If this command is specified in a *.prm* file, it defines the root function for an ANSI C or C++ application (function invoked at the end of startup function). When `MAIN` is not specified in the *.prm* file, the linker looks for a function named MAIN and uses it as root. The user can specify any static or global function as the application root function.

Assembly applications do not require a MAIN function.

Example:

```
MAIN MyGlobMain /* Specify a global variable as
                   application root */
MAIN myFile.o:myLocMain /* Specify a local
                  variable as application root */
```

Local symbols defined in an assembly module cannot be specified as the root function.

4.4.5 MAPFILE: Configure Map File

Syntax:

```
MAPFILE (ALL | NONE | TARGET | FILE |
STARTUP_STRUCT | SEC_ALLOC | OBJ_ALLOC | OBJ_DEP |
OBJ_UNUSED | COPYDOWN | STATISTIC)
```

Description: A protocol of the link process is written to a list file, referred to as the map file. The name of the map file is the same as the *.abs* file, but with extension *.map*. The map file is written to the directory given by the environment variable TEXTPATH. The map file consists of up to nine sections. **Table 4-2** lists and defines each section.

The `MAPFILE` command is optional and controls generation of the map file. `MAPFILE ALL` is the default. The `ALL` option creates a map file that contains all sections, which provide link time information. **Table 4-3** lists all command options and information generated by each one.

NOTE: *A map file is not created if objects are not found in an object file and the linking process fails.*

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Table 4-2. Map File Sections

Section	Description
TARGET	This section names the target processor and memory model.
FILE	This section lists the names of all files from which objects were used or referenced during the link process. In most cases, these are the same names listed in the linker parameter file between the keywords NAMES and END.
STARTUP	This section lists the prestart code and the values used to initialize the startup descriptor _startupData. The startup descriptor is listed member by member with the initialization data at the right hand side of the member name.
SEGMENT ALLOCATION	This section lists segments, in which at least one object was allocated. At the right hand side of the segment name is a pair of numbers, which gives the address range the objects belonging to the segment were allocated.
VECTOR ALLOCATION	This section provides the address and initial value and function for the vector.
OBJECT ALLOCATION	This section contains the names of all allocated objects and their addresses. The objects are grouped by module. If an address of an object is followed by the @ sign, the object comes from a ROM library. In this case, the absolute file contains no code for the object (if it is a function), but the object's address was used for linking. If an address of a string object is followed by a dash (–), the string is a suffix of some other string. As an example, if the strings abc and bc are present in the same program, the string bc is not allocated and its address is the address of abc plus 1.
UNUSED OBJECTS	This section lists all objects found in the object files that were not linked.
COPYDOWN	This section lists all blocks copied from ROM to RAM at program startup.
OBJECT DEPENDENCY	This section lists the names of global objects used by functions and variables.
STATISTICS	This section generates information about the size of the code generated.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Table 4-3. Map File Options

Option	Meaning
ALL	A map file will be generated containing all information available.
COPYDOWN	Information about the initialization value for objects allocated in RAM will be written to the map file (COPYDOWN section). This section is only relevant for high-level language (ANSI C or C++) applications.
FILE	Information about application source files will be inserted in the map file.
NONE	No map file will be generated.
OBJ_ALLOC	Information about allocated objects will be inserted in the map file (OBJECT ALLOCATION section).
OBJ_UNUSED	List of all unused objects will be inserted in the map file (UNUSED OBJECTS section).
OBJ_DEP	Dependencies between objects in the application will be inserted in the map file (OBJECT DEPENDENCY section).
SEC_ALLOC	Information about sections used in the application will be inserted in the map file (SECTION ALLOCATION section).
STARTUP_STRUCT	Information about the startup structure will be inserted in the map file (STARTUP section). This section is only relevant for high-level language (ANSI C or C++) applications.
STATISTIC	Statistic information about the link session will be inserted in the map file (STATISTICS section).
TARGET	Information about the target processor and memory model will be inserted in the map file (TARGET section).

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

4.4.6 NAMES: List Files

Syntax:
```
NAMES <FileName>[+] <FileName>[+] END
```

Description: The NAMES block contains the list of all binary files building the application. This is the only place absolute, library, or object library files can be specified. This block is mandatory and can be specified only once in a *.prm* file. The linker reads all files given between NAMES and END. The files are searched for in the project directory, then in the directories specified in the environment variables OBJPATH, LIBPATH, and GENPATH. The files may be either object files, absolute files, or libraries.

Since the linker is a smart linker, only referenced objects (variables and functions) are linked to the application.

A plus sign after a filename (for example, FileName+) switches off smart linking for the specified file. No blank space is allowed between the filename and the plus sign. All objects defined in this file will be linked with the application, regardless of whether they are used or not. This is equivalent to specifying the filename followed by a * (FileName:*) in the ENTRIES block.

Example:
```
LINK fibo.abs

NAMES  fibo.o startup.o END
SEGMENTS
    MY_RAM = READ_WRITE 0x1000 TO 0x18FF;
    MY_ROM = READ_ONLY  0x8000 TO 0x8FFF;
    MY_STK = READ_WRITE 0x1900 TO 0x1FFF;
PLACEMENT
    DEFAULT_ROM   INTO  MY_ROM;
    DEFAULT_RAM   INTO  MY_RAM;
    SSTACK        INTO  MY_STK;
END
VECTOR ADDRESS 0xFFFE _Startup /*set reset vector*/
```

In this example, the *fibo.abs* application is built from the files *fibo.o* and *startup.o*.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

4.4.7 SEGMENTS: Define Memory Map

Syntax:

```
SEGMENTS {(READ_ONLY | READ_WRITE |NO_INIT |PAGED)
<startAddr> (TO <endAddr> | SIZE <size>)
[ALIGN <alignmentRule>] [FILL <fillPattern>]}
END
```

Description: The SEGMENTS block is optional in a PRM file. The SEGMENTS command allows the user to assign meaningful names to address ranges on the target board. These names can then be used in subsequent PLACEMENT statements, thus increasing the readability of the parameter file.

Each address range defined is associated with:

- A qualifier
- A start and end address or a start address and a size
- An optional alignment rule
- An optional fill pattern

Segments are closely related to hardware memory areas. For example, there may be one READ_ONLY segment for each bank of the target board ROM area and another one covering the RAM area.

Example: Using the small memory model, the user can define a segment for the RAM area and another one for the ROM area.

```
LINK   test.abs
NAMES  test.o startup.o END
SEGMENTS
  RAM_AREA = READ_WRITE 0x00000 TO 0x07FFF;
  ROM_AREA = READ_ONLY  0x08000 TO 0x0FFFF;
END
PLACEMENT
     .data              INTO RAM_AREA;
     .text              INTO ROM_AREA;
END
STACKSIZE 0x50
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Example: Using the banked memory model, a segment can be defined for the RAM area, another for the non-banked ROM area, and one for each target processor bank.

```
LINK    test.abs
NAMES   test.o startup.o END
SEGMENTS
  RAM_AREA        = READ_WRITE 0x00000 TO 0x07FFF;
  NON_BANKED_AREA = READ_ONLY  0x0C000 TO 0x0FFFF;
  BANK0_AREA      = READ_ONLY  0x08000 TO 0x0BFFF;
  BANK1_AREA      = READ_ONLY  0x18000 TO 0x1BFFF;
  BANK2_AREA      = READ_ONLY  0x28000 TO 0x2BFFF;
END
PLACEMENT
    .data              INTO RAM_AREA;
    .init, .startData,
    .rodata1,
    NON_BANKED, .copy  INTO NON_BANKED_AREA;
    .text              INTO BANK0_AREA, BANK1_AREA,
                            BANK2_AREA;
END
STACKSIZE 0x50
```

A physical segment may be split into several virtual segments, allowing a better structuring of object allocation and taking advantage of processor properties.

Example: In the small memory model, the user can define a segment for the direct page area, another for the rest of the RAM area, and another one for the ROM area.

```
LINK    test.abs
NAMES   test.o startup.o END

SEGMENTS
  DIRECT_RAM = READ_WRITE 0x00000 TO 0x000FF;
  RAM_AREA   = READ_WRITE 0x00100 TO 0x07FFF;
  ROM_AREA   = READ_ONLY  0x08000 TO 0x0FFFF;
END
PLACEMENT
    myRegister         INTO DIRECT_RAM;
    .data              INTO RAM_AREA;
    .text              INTO ROM_AREA;
END
STACKSIZE 0x50
```


Freescale Semiconductor, Inc.
For More Information On This Product,
Go to: www.freescale.com

4.4.7.1 Segment Qualifier

Different qualifiers are available for segments. Table 4-4 identifies and defines all available qualifiers.

Table 4-4. Segment Qualifier Descriptions

Qualifier	Meaning
READ_ONLY	Qualifies a segment, where read-only access is allowed. Objects within such a segment are initialized at application loading time.
READ_WRITE	Qualifies a segment, where read and write accesses are allowed. Objects within such a segment are initialized at application startup. This is only the case when linking a high-level language (ANSI C or C++) application.
NO_INIT	Qualifies a segment, where read and write accesses are allowed. Objects within such a segment remain unchanged during application startup. This qualifier may be used for segments referring to a battery backed RAM. Sections placed in a NO_INIT segment should not contain an initialized variable (variable defined as int c = 8). This is only the case when linking a high-level language (ANSI C or C++) application.
PAGED	Qualifies a segment, where read and write accesses are allowed. Objects within such a segment remain unchanged during application startup. Additionally, objects located in two PAGED segments may overlap. This qualifier is used for memory areas, where some user-defined page switching mechanism is required. Sections placed in a NO_INIT segment should not contain an initialized variable (variable defined as int c = 8). This is only the case when linking a high-level language (ANSI C or C++) application.

Example:

```
SEGMENTS
  ROM    = READ_ONLY  0x1000 SIZE 0x2000;
  CLOCK  = NO_INIT    0xFF00 TO   0xFFFF;
  RAM    = READ_WRITE 0x3000 TO   0x3EFF;
  Page0  = PAGED      0x4000 TO   0x4FFF;
  Page1  = PAGED      0x4000 TO   0x4FFF;
END
```

In the previous example:

- Segment ROM is a READ_ONLY memory area. It starts at address 0x1000 and is 0x2000 bytes from address 0x1000 to 0x2FFF.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

- Segment `RAM` is a READ_WRITE memory area. It starts at address 0x3000 and ends at 0x3FFF (size = 0x1000 bytes).
- Segment `CLOCK` is a READ_WRITE memory area. It starts at address 0xFF00 and ends at 0xFFFF (size = 256 bytes).
- Segments `Page0` and `Page1` are READ_WRITE memory areas. These are overlapping segments. It is the user's responsibility to select the correct page before accessing data from these segments.

4.4.7.2 Segment Alignment

The HC12, HC08, and HC05 processors do not require alignment for code or data objects. Users can choose to define their own alignment rule for a segment.

An alignment rule can be associated with each segment in the application. This may be useful when specific alignment rules are expected on a certain memory range due to hardware restrictions.

The alignment rule has this format:

```
[defaultAlignment] [ObjSizeRange:alignment]
```

Table 4-5. Segment Alignment Rule Format

Item	Description
defaultAlignment	The alignment value for all objects that do not match the conditions of a defined range
ObjSizeRange	Defines a certain condition. The condition has the form: size : rule applies to objects, where size is equal to size < size : rule applies to objects, where size is smaller than size > size: rule applies to objects, where size is bigger than size <= size: rule applies to objects, where size is smaller or equal to size >= size: rule applies to objects, where size is bigger or equal to size From size1 to size2: the rule applies to objects where size is greater or equal to size1 and smaller or equal to size2
alignment	Defines the alignment value for objects matching the condition defined in the current alignment block (enclosed in square brackets).

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Example:

```
LINK    test.abs
NAMES   test.o startup.o END

SEGMENTS
  DIRECT_RAM = READ_WRITE 0x00000 TO 0x000FF
               ALIGN 2 [< 2: 1];
  RAM_AREA   = READ_WRITE 0x00100 TO 0x07FFF
               ALIGN [1:1] [2..3:2] [>=4:4];
  ROM_AREA   = READ_ONLY  0x08000 TO 0x0FFFF;
END
PLACEMENT
    myRegister        INTO DIRECT_RAM;
    .data             INTO RAM_AREA;
    .text             INTO ROM_AREA;
END
STACKSIZE 0x50
```

In the previous example:

- In `DIRECT_RAM` segment, objects (whose size is one byte) are aligned on byte boundary; all other objects are aligned on 2-byte boundary.
- In `RAM_AREA` segment, 1-byte objects are aligned on byte boundary, objects equal to two or three bytes are aligned on 2-byte boundary, and all other objects are aligned on 4-byte boundary.
- Default alignment rule applies to the `ROM_AREA` segment.

An alignment rule can be specified as follows:

```
ALIGN [<defaultAlignment>] [{'['(<Number>| <Number> 'TO'
<Number>| ('<' | '>' | '<=' | '>=')<Number>)']:'<alignment>}]
```

`defaultAlignment` is used to specify the alignment factor for objects that are not specified by a condition in the alignment list in **Table 4-6**. If no alignment list is specified, the default alignment factor applies to all objects allocated in the segment. The default alignment factor is optional.

The specified alignment applies to each object inside the segment.

Freescale Semiconductor, Inc.

Table 4-6. Segment Alignment Items List

Notation	Meaning
[<size>:<align.>]	Size is equal to <size>.
[<sz1> to <sz2>:<align.>]	Size is bigger or equal to <sz1> and smaller or equal to <sz2>.
[<<size>:<align.>]	Size is smaller than <size>.
[<=<size>:<align.>]	Size is smaller or equal to <size>.
[><size>:<align.>]	Size is bigger than <size>.
[>=<size>:<align.>]	Size is bigger or equal to <size>.

Example:

```
SEGMENTS
    RAM_1  = READ_WRITE 0x800 TO 0x8FF
             ALIGN 2 [1:1];
    RAM_2  = READ_WRITE 0x900 TO 0x9FF
             ALIGN [2 TO 3:2] [>= 4:4];
    RAM_3  = READ_WRITE 0xA00 TO 0xAFF
             ALIGN 1 [>=2:2];
END
```

In the previous example:

- Inside segment RAM_1, all objects with size equal to one byte are aligned on a 1-byte boundary and all other objects are aligned on a 2-byte boundary.
- Inside of segment RAM_2, all objects with size equal to two or three bytes are aligned on a 2-byte boundary and all objects bigger or equal to four are aligned on a 4-byte boundary. One-byte objects follow the default processor alignment rule.
- Inside segment RAM_3, all objects bigger or equal to two bytes are aligned on a 2-byte boundary and all other objects are aligned on a 1-byte boundary.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

4.4.7.3 Segment Fill Pattern

A fill pattern can be associated with each segment in the application. This may be useful to automatically initialize uninitialized variables in the segments with a predefined pattern. For assembly applications, the fill pattern can be used only in `READ_ONLY` segments.

The default fill pattern for code and data segments is the null character. Users can define their own fill pattern for a segment. The fill pattern definition in the segment block overrides the default fill pattern. A fill pattern can be defined for the READ_WRITE memory area only when linking a high-level language (ANSI C, C++) application.

A fill pattern can be specified like this:

Syntax:

```
FILL <HexByte> {<HexByte>}
```

Example:

```
SEGMENTS
     ROM_1   = READ_ONLY 0x800 TO 0x8FF
               FILL 0xAA 0x55;
END
```

In the previous example, fill bytes are initialized with the pattern 0xAA55.

If the size of an object to be initialized is higher than the size of the specified pattern, the pattern is repeated as many times as required to fill the objects. In the previous example, an object of four bytes will be initialized with 0xAA55AA55.

If the size of an object to be initialized is smaller than the size of the specified pattern, the pattern is truncated to match the size of the object. In the previous example, an object of one byte will be initialized with 0xAA.

When the value specified in an element of a fill pattern does not fit in a byte, it is truncated to a byte value.

Example:

```
SEGMENTS
     ROM_1   = READ_ONLY 0x800 TO 0x8FF
               FILL 0xAA55;
END
```

Freescale Semiconductor, Inc.

In the previous example, fill bytes are initialized with the pattern 0x55. The specified fill pattern is truncated to a 1-byte value. Fill patterns provide an initial value to the padding bytes inserted between two objects during object allocation. This marks the unused position with a specific marker and can be detected inside the application. For example, an unused position inside a code section can be initialized with the hexadecimal code for the NOP instruction.

High-level language (C, C++) Example:

```
LINK   test.abs
NAMES  test.o startup.o END

SEGMENTS
  DIRECT_RAM = READ_WRITE 0x00000 TO 0x000FF
               FILL 0xAA;
  RAM_AREA   = READ_WRITE 0x00100 TO 0x07FFF
               FILL 0x22;
  ROM_AREA   = READ_ONLY  0x08000 TO 0x0FFFF;
END
PLACEMENT
    myRegister        INTO DIRECT_RAM;
    .data             INTO RAM_AREA;
    .text             INTO ROM_AREA;
END
STACKSIZE 0x50
```

In the previous example:

- In DIRECT_RAM, alignment bytes between objects are initialized with 0xAA.
- In RAM_AREA, alignment bytes are initialized with 0x22.
- In ROM_AREA, alignment bytes are initialized with 0x00.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

4.4.8 PLACEMENT: Place Sections into Segments

The PLACEMENT block allows the user to physically place each section in a specific memory area (segment). The sections specified in a PLACEMENT block may be linker-predefined sections or user sections specified in one of the source files used to build the application.

A section is a named group of global objects declared in the source file, such as functions and global variables. A segment is not necessarily a contiguous memory range. In the linker parameter file, each section is associated with a segment so the linker knows where to allocate objects belonging to a section.

A programmer may decide to organize data into sections:

- To enhance application structure
- To ensure that common purpose data is grouped together
- To take advantage of target processor specific addressing mode

Syntax:

```
PLACEMENT
SectionName{,sectionName} INTO SegSpec{,SegSpec};
{SectionName{,sectionName} INTO SegSpec{,SegSpec};}
END
```

Description: The PLACEMENT block is mandatory in a *.prm* file. Each placement statement between the PLACEMENT and END defines a relation between logical sections and physical memory ranges called segments.

Example:

```
SEGMENTS
    MY_RAM = READ_WRITE 0x1000 TO 0x18FF;
    ROM_1    = READ_ONLY  0x8000 TO 0x8FFF;
END
PLACEMENT
   .text, .rodata INTO ROM_1;
END
```

In the previous example, objects from section `.text` are allocated first and then objects from section `.rodata` are allocated into segment `ROM_1`.

Starting with the first section, objects are allocated in the first memory range in the list. If a segment is full, allocation continues in the next segment.

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

Example:

```
SEGMENTS
    MY_RAM = READ_WRITE 0x1000 TO 0x18FF;
    ROM_1   = READ_ONLY  0x8000 TO 0x8FFF;
    ROM_2   = READ_ONLY  0xA000 TO 0xAFFF;
END
PLACEMENT
   .text INTO ROM_1, ROM_2;
END
```

In the previous example, objects from section `.text` are allocated first in segment `ROM_1` and continue in section `ROM_2`. A statement inside the PLACEMENT block can be split over several lines and terminated with a semicolon. The SEGMENTS block must always be defined before the PLACEMENT block because segments referenced in the PLACEMENT block must be defined previously in the SEGMENTS block.

Some restrictions apply to commands specified in the PLACEMENT block:

- The `.copy` section should be the last section in the section list to be specified in the PLACEMENT block.
- When the `.stack` section is specified in the PLACEMENT block along with other sections, an additional `STACKSIZE` command is required in the PRM file.
- Predefined sections `.text` and `.data` must always be specified in the PLACEMENT block. They are used to retrieve the default placement for code or variable sections. All code or constant sections, which do not appear in the PLACEMENT block, are allocated in the same segment list as the `.text` section. All variable sections, which do not appear in the PLACEMENT block, are allocated in the same segment list as the `.data` section.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

4.4.8.1 Specifying a List of Sections

When several sections are specified in a PLACEMENT block, the sections are allocated in the sequence where they are listed.

Example:

```
LINK    test.abs
NAMES   test.o startup.o END

SEGMENTS
  RAM_AREA    = READ_WRITE 0x00100 TO 0x002FF;
  STK_AREA    = READ_WRITE 0x00300 TO 0x003FF;
  ROM_AREA    = READ_ONLY  0x08000 TO 0x0FFFF;
END
PLACEMENT
     .data, dataSec1,
      dataSec2           INTO RAM_AREA;
     .text, myCode       INTO ROM_AREA;
     .stack              INTO STK_AREA;
END
```

In the previous example:

- Inside segment `RAM_AREA`, the objects defined in the `.data` section are allocated first, then objects defined in section `dataSec1`, and finally objects defined in section `dataSec2`.
- Inside segment `ROM_AREA`, objects defined in the `.text` section are allocated, then objects are defined in section `myCode`.

NOTE: *The linker is case sensitive. Section names specified in the PLACEMENT block must be valid predefined or user-defined sections. Sections* `DataSec1` *and* `dataSec1` *are different sections.*

Freescale Semiconductor, Inc.

4.4.8.2 Specifying a List of Segments

When several segments are specified in a PLACEMENT block, the segments are used in the sequence where they are listed. Allocation is performed for the first segment in the list, until this segment is full. Then allocation continues for the next segment in the list, and so on until all objects are allocated.

Example:

```
LINK   test.abs
 NAMES  test.o startup.o END
 SEGMENTS
   RAM_AREA        = READ_WRITE 0x00100 TO 0x002FF;
   STK_AREA        = READ_WRITE 0x00300 TO 0x003FF;
   NON_BANKED_AREA = READ_ONLY  0x0C000 TO 0x0FFFF;
   BANK0_AREA      = READ_ONLY  0x08000 TO 0x0BFFF;
   BANK1_AREA      = READ_ONLY  0x18000 TO 0x1BFFF;
   BANK2_AREA      = READ_ONLY  0x28000 TO 0x2BFFF;
 END
 PLACEMENT
     .data                INTO RAM_AREA;
     .stack               INTO STK_AREA;
     .init, .startData,
     .rodata1,
     NON_BANKED, .copy INTO NON_BANKED_AREA;
     .text                INTO BANK0_AREA, BANK1_AREA,
                               BANK2_AREA;
 END
```

In the previous example:

- Functions implemented in section `.text` are allocated first in segment `BANK0_AREA`. When memory for this segment is filled, allocation continues in segment `BANK_1_AREA`, then in `BANK2_AREA`.

NOTE: *Segment names specified in the PLACEMENT block must be valid segment names defined in the SEGMENTS block. The linker is case sensitive. Segments `Ram_Area` and `RAM_AREA` are different segments.*

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

4.4.8.3 Predefined Sections

When linking a high-level language (ANSI C or C++) application, a couple of predefined section names can be grouped into sections named by the run-time routines.

For instance,

- Sections for things besides variables and functions: `.rodata1`, `.copy`, `.stack`
- Sections for grouping large sets of objects: `.data`, `.text`
- A section for placing objects initialized by the linker: `.startData`
- A section to allocate read-only variables: `.rodata`

NOTE: *The sections `.data` and `.text` provide default sections for allocating objects.*

.rodata1 All string literals (for example, “This is a string”) are allocated in section `.rodata1`. If this section is associated with a segment qualified as `READ_WRITE`, the strings are copied from ROM to RAM at startup.

.rodata Any constant variable declared as `const` in a C module or as DC in an assembler module, which is not allocated in a user-defined section, is allocated in section `.rodata`. Usually, the `.rodata` section is associated with the `READ_ONLY` segment.

.copy Initialization data belongs to section `.copy`. If a source file contains the declaration

```
int a[] = {1, 2, 3};
```

the hex string `000100020003` (six bytes), which is copied to a location in RAM at program startup, belongs to segment `.copy`.

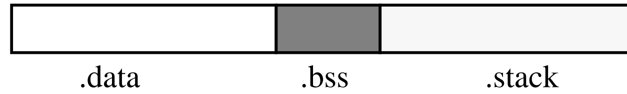
If the `.rodata1` or `.rodata` section is allocated to a `READ_WRITE` segment, all strings or constants also belong to the `.copy` section. Objects in this section are copied at startup from ROM to RAM.

.stack The runtime stack has its own segment named `.stack`. It should always be allocated to a `READ_WRITE` segment.

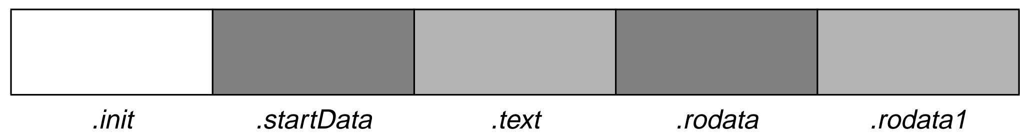
For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

.data The `.data` section is the default for all objects normally allocated to RAM. It is used for variables not belonging to any section or to a section not assigned a segment in the PLACEMENT block. If the `.bss` or `.stack` sections are not associated with a segment, they are included in the `.data` memory area in the following structure.



.text The `.text` section is is the default section for all functions. If a function is not assigned to a certain section in the source code or if its section is not associated with a segment in the parameter file, it is automatically added to the `.text` section. If the `.rodata`, `.rodata1`, `.startData`, or `.init` sections are not associated with a segment, they are included in the `.text` memory area in the following structure.



.StartData The startup description data initialized by the linker and used by the startup routine is allocated to segment `.startData`. This section must be allocated to a `READ_ONLY` segment.

.init The application entry point is stored in the `.init` section. This section also has to be associated with a `READ_ONLY` segment.

NOTE: *The `.data` and `.text` sections must always be associated with a segment.*

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

4.4.8.4 Allocating User-Defined Sections

Not all sections need to be listed in the PLACEMENT block. Segments in which sections are allocated depend on the type of section.

For example:

- Sections containing data are allocated next to the `.data` section.
- Sections containing code, constant variables, or string constants are allocated next to the `.text` section.

In the segment where `.data` is placed, allocation is performed as follows:

- Objects from section `.data` are allocated.
- Objects from section `.bss` are allocated (if `.bss` is not specified in the PLACEMENT block).
- Objects from the first user-defined data section (not specified in the PLACEMENT block) are allocated.
- Objects from the next user-defined data section (not specified in the PLACEMENT block) are allocated.
- This continues until all user-defined data sections are allocated.
- If the section `.stack` is not specified in the PLACEMENT block and is defined with a `STACKSIZE` command, the stack is allocated.

.data	.bss	user data section 1	user data section 2		user data section n	.stack

Allocation in the segment where `.text` is placed is performed as follows:

- Objects from section `.init` are allocated (if `.init` is not specified in the PLACEMENT block).
- Objects from section `.startData` are allocated (if `.startData` is not specified in the PLACEMENT block).
- Objects from section `.text` are allocated.

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

- Objects from section `.rodata` are allocated (if `.rodata` is not specified in the PLACEMENT block).
- Objects from section `.rodata1` are allocated (if `.rodata1` is not specified in the PLACEMENT block).
- Objects from the first user-defined code section (not specified in PLACEMENT block) are allocated.
- Objects from the next user-defined code section (not specified in the PLACEMENT block) are allocated.
- This continues until all user-defined code sections are allocated.
- Objects from section `.copy` are allocated (if `.copy` is not specified in the PLACEMENT block).

.init	.start-Data	.text	.rodata	rodata1	user section 1		user data section n	.copy

4.4.9 STACKSIZE: Define Stack Size

Syntax: `STACKSIZE Number`

Description: The `STACKSIZE` command is optional in a PRM file. Additionally, both `STACKTOP` and `STACKSIZE` commands cannot be specified in a PRM file. The `STACKSIZE` command defines the stack size. Use this command if it does not matter where the stack is allocated but only how large it is. When the stack is defined by a `STACKSIZE` command alone, the stack is placed next to the `.data` section.

Example:

```
SEGMENTS
   MY_RAM = READ_WRITE 0xA00 TO 0xAFF;
   MY_ROM = READ_ONLY  0x800 TO 0x9FF;
END
PLACEMENT
   .text   IN MY_ROM;
   .data   IN MY_RAM;
END
STACKSIZE 0x60
```


For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

In the previous example, if the section `.data` is four bytes wide (from address 0xA00 to 0xA03), the section `.stack` is allocated next to it from address 0xA63 down to address 0xA04. The stack initial value is set to 0xA62.

When the stack is defined by a `STACKSIZE` command associated with the placement of the `.stack` section, the stack should start at the segment start address. It is incremented by the specified value and defined to the start address of the segment, where `.stack` has been placed.

Example:

```
SEGMENTS
   MY_STK = NO_INIT    0xB00 TO 0xBFF;
   MY_RAM = READ_WRITE 0xA00 TO 0xAFF;
   MY_ROM = READ_ONLY  0x800 TO 0x9FF;
END
PLACEMENT
   .text   IN MY_ROM;
   .data   IN MY_RAM;
   .stack  IN MY_STK;
END
STACKSIZE 0x60
```

In the previous example, the section `.stack` is allocated from address 0xB5F down to address 0xB00. The stack initial value is set to 0xB5E.

In an assembly application, the stack pointer must be initialized in the source code. Defining the stack in the *.prm* file only ensures no overlap between the stack and the code or data sections in the application.

4.4.10 STACKTOP: Define Stack Pointer Initial Value

Syntax:

```
STACKTOP Number
```

Description: The `STACKTOP` command is optional in a PRM file. Additionally, the user cannot specify both `STACKTOP` and `STACKSIZE` commands in a PRM file. The `STACKTOP` command defines the initial value for the stack pointer.

Example: If `STACKTOP` is defined as `STACKTOP 0xBFF`, the stack pointer will be initialized with `0xBFF` at application startup.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

When the stack is defined by a STACKTOP command alone, a default size is assigned to the stack. This size depends on the processor and is big enough to store the target processor PC. When the stack is defined by a STACKTOP command associated with the placement of the .stack section, the stack should start at the specified address. It is defined down to the start address of the segment, where .stack has been placed.

Example:

```
SEGMENTS
   MY_STK = NO_INIT    0xB00 TO 0xBFF;
   MY_RAM = READ_WRITE 0xA00 TO 0xAFF;
   MY_ROM = READ_ONLY  0x800 TO 0x9FF;
END
PLACEMENT
   .text   IN MY_ROM;
   .data   IN MY_RAM;
   .stack  IN MY_STK;
END
STACKTOP 0xB7E
```

In the previous example, the stack pointer will be defined from address 0xB7E down to address 0xB00.

In an assembly application, the stack pointer must be initialized in the source code. Defining the stack in the *.prm* file only ensures no overlap between the stack and the code or data sections in the application.

4.4.11 VECTOR: Initialize Vector Table

Syntax:

```
VECTOR  (InitByAddr | InitByNumber)
```

Description: The VECTOR command initializes the vector table. The vector table can be initialized in the assembly source file or in the linker parameter file, although initialization in the *.prm* file is recommended.

A vector is a small amount of memory about the size of a function address. This command allows the user to initialize the processor vectors while downloading the absolute file. A VECTOR command consists of a vector location (containing vector location) and a vector target (containing the value to store in the vector).

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

The syntax VECTOR <Number> is only valid when the vector table starts at address 0x0000. The address where the vector is allocated is evaluated as <Number> * <Size of a Function Pointer>.

The syntax VECTOR ADDRESS is valid in any case. The size of entries in the vector table depends on the target processor.

The vector target can be specified:

- As a function name
- As an absolute address

Different syntaxes are available for the `VECTOR` command.

Table 4-7. VECTOR Command Syntax

Command	Meaning
VECTOR ADDRESS 0xFFFE 0x1000	Indicates that the value 0x1000 must be stored at address 0xFFFE
VECTOR ADDRESS 0xFFFE FName	Indicates that the address of the function name (FName) must be stored at address 0xFFFE
VECTOR ADDRESS 0xFFFE FName + 2	Indicates that the address of the function (FName) incremented by 2 must be stored at address 0xFFFE. This syntax may be useful when working with a common interrupt service routine.

Example:

```
VECTOR ADDRESS 0xFFFE _Startup
VECTOR ADDRESS 0xFFFC 0xA00
VECTOR 0 _Startup
VECTOR 1 0xA00
```

In the previous example, if the size of a function pointer is coded on two bytes:

- The vector located at address 0xFFFE is initialized with the address of the function `_Startup`.
- The vector located at address 0xFFFC is initialized with the absolute address 0xA00.
- Vector number 0 (located at address 0x000) is initialized with the address of the function `_Startup`.
- Vector number 1 (located at address 0x002) is initialized with the absolute address 0xA00.

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

The user can specify an additional offset when the vector target is a function name. In this case, the vector will be initialized with the address of the object plus the specified offset.

Example:

```
VECTOR ADDRESS 0xFFFE CommonISR + 0x10
```

In the previous example, the vector located at address 0xFFE is initialized with the address of the function `CommonISR` plus 0x10 bytes. If `CommonISR` starts at address 0x800, the vector will be initialized with 0x810. This notation is useful for the common interrupt handler. All objects specified in a `VECTOR` command are entry points in the application. They are always linked with the application, as well as the objects they refer to.

4.4.11.1 Initializing Vector Table in Linker Parameter File

Initializing the vector table from the parameter file allows initialization of single entries in the table (shown in next example). The user can initialize all entries in the vector table. The labels or functions must be inserted in the vector table and implemented in the assembly source file. All labels must be published; otherwise, they cannot be addressed in the linker parameter file.

Example:

```
XDEF
IRQFunc, XIRQFunc, SWIFunc, OpCodeFunc, ResetFunc
DataSec: SECTION
Data: DS.W 5
; Each interrupt increments another table element
CodeSec: SECTION
; Implementation of the interrupt functions
IRQFunc:
        LDAB #0
        BRA int
XIRQFunc:
        LDAB #2
        BRA int
SWIFunc:
        LDAB #4
        BRA int
OpCodeFunc:
        LDAB #6
        BRA int
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
ResetFunc:
        LDAB #8
        BRA entry
int:
        LDX #Data ;Load address of symbol Data in X
        ABX       ;X, address of element in table
        INC 0, X  ;The table element is incremented
        RTI
entry:
        LDS #$AFE
loop:   BRA loop
```

NOTE: *The functions IRQFunc, XIRQFunc, SWIFunc, OpCodeFunc, and ResetFunc are published. This is required because they are referenced in the PRM file.*

Since the HC12 processor automatically pushes all registers on the stack when an interrupt occurs, the interrupt function does not need to save and restore the registers it is using. All interrupt functions must be terminated with an RTI (return from interrupt) instruction.

The vector table is initialized using the linker command `VECTOR ADDRESS`.

Example:

```
LINK test.abs
NAMES
   test.o
END
SEGMENTS
   MY_ROM   = READ_ONLY   0x0800 TO 0x08FF;
   MY_RAM   = READ_WRITE 0x0B00 TO 0x0CFF;
END
PLACEMENT
   .data        INTO MY_RAM;
   .text        INTO MY_ROM;
END
INIT ResetFunc
VECTOR ADDRESS 0xFFF2 IRQFunc
VECTOR ADDRESS 0xFFF4 XIRQFunc
VECTOR ADDRESS 0xFFF6 SWIFunc
VECTOR ADDRESS 0xFFF8 OpCodeFunc
VECTOR ADDRESS 0xFFFE ResetFunc
```


For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

NOTE: *The statement `INIT ResetFunc` defines the application entry point. Usually, this entry point is initialized with the same address as the reset vector. The statement `VECTOR ADDRESS 0xFFF2 IRQFunc` specifies that the address of function `IRQFunc` should be written at address 0xFFF2.*

4.4.11.2 Initializing Vector Table in Assembly Source File Using a Relocatable Section

Initializing the vector table in the assembly source file requires that all entries in the table be initialized. Unused interrupts must be associated with a standard handler.

The labels or functions, inserted in the vector table, must be implemented in one of the assembler source files. The vector table can be defined in an assembly source file in an additional section containing constant variables.

Example for HC12:

```
        XDEF ResetFunc
DataSec:SECTION
Data:   DS.W 5 ;Each interrupt increments element
CodeSec:SECTION;Implementation of the interrupt functions
 IRQFunc:
          LDAB #0
          BRA  int
 XIRQFunc:
          LDAB #2
          BRA  int
 SWIFunc:
          LDAB #4
          BRA  int
 OpCodeFunc:
          LDAB #6
          BRA  int
 ResetFunc:
          LDAB #8
          BRA  entry
 DummyFunc:
          RTI
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
int:
            LDX   #Data
            ABX
            INC   0, X
            RTI
entry:
            LDS #$AFE
loop:       BRA loop

VectorTable:SECTION;Definition of vector table
IRQInt:       DC.W IRQFunc
XIRQInt:      DC.W XIRQFunc
SWIInt:       DC.W SWIFunc
OpCodeInt:    DC.W OpCodeFunc
COPResetInt:  DC.W DummyFunc   ;No function
                               ;attached to COP Reset
ClMonResInt:  DC.W DummyFunc   ;No function
                               ;attached to Clock
                               ;MonitorReset
ResetInt:     DC.W ResetFunc
```

NOTE: *Each constant in the section `VectorTable` is defined as a word (2-byte constant) because entries in the HC12 vector table are 16 bits wide. In the previous example, the constant `IRQInt` is initialized with the address of the label `IRQFunc`. The constant `XIRQInt` is initialized with the address of the label `XIRQFunc`. All labels specified as an initialization value must be defined, published (using XDEF), or imported (using XREF) before the vector table section. Forward referencing is not allowed in the DC directive.*

When developing a banked application, ensure that interrupt functions are located in the non-banked memory area.

The section should now be placed at the expected address. This is performed in the linker parameter file, shown in the next example.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Example:

```
LINK test.abs
NAMES  test.o END

SEGMENTS
   MY_ROM  = READ_ONLY  0x0800 TO 0x08FF;
   MY_RAM  = READ_WRITE 0x0A00 TO 0x0BFF;
 /* Define memory range for vector table */
   Vector  = READ_ONLY  0xFFF2 TO 0xFFFF;
END
PLACEMENT
   .data        INTO MY_RAM;
   .text        INTO MY_ROM;
   VectorTable  INTO Vector;
END

INIT ResetFunc
ENTRIES
   *
END
```

NOTE: *The statement `Vector = READ_ONLY 0xFFF2 TO 0xFFFF` defines the memory range for the vector table. The statement `VectorTable INTO Vector` specifies that the vector table should be loaded in the read-only memory area vector. The constant `IRQInt` will be allocated at address 0xFFF2, the constant `XIRQInt` will be allocated at address 0xFFF4, and so on. The constant `ResetInt` will be allocated at address 0xFFFE. The statement `ENTRIES * END` switches smart linking OFF. If this statement is missing from the PRM file, the vector table will not be linked with the application because it is never referenced. The smart linker only links objects referenced in the absolute file.*

4.4.11.3 Initializing Vector Table in Assembly Source File Using an Absolute Section

Initializing the vector table in the assembly source file requires that all entries in the table be initialized. Unused interrupts must be associated with a standard handler. Labels or functions inserted in the vector table must be implemented in one of the assembly source files. The vector table can be defined in an assembly source file in an additional section containing constant variables, shown in the next example.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Example for HC12:

```
            XDEF ResetFunc
DataSec:     SECTION
Data:        DS.W 5            ;Each interrupt increments
                               ;element of table
CodeSec:     SECTION           ;Implementation of the
                               ;interrupt functions
IRQFunc:
             LDAB #0
             BRA  int
XIRQFunc:
             LDAB #2
             BRA  int
SWIFunc:
             LDAB #4
             BRA  int
OpCodeFunc:
             LDAB #6
             BRA  int
ResetFunc:
             LDAB #8
             BRA  entry
DummyFunc:
             RTI
int:
             LDX  #Data
             ABX
             INC  0, X
             RTI
entry:
             LDS #$AFE
loop:        BRA loop

             ORG $FFF2         ;Definition of vector table
                               ;in absolute section
                               ;starting at address $FFF2
IRQInt       :DC.W IRQFunc
XIRQInt      :DC.W XIRQFunc
SWIInt       :DC.W SWIFunc
OpCodeInt    :DC.W OpCodeFunc
COPResetInt  :DC.W DummyFunc   ;No function attached
                               ;to COP Reset
ClMonResInt  :DC.W DummyFunc   ;No function attached
                               ;to Clock MonitorReset
ResetInt     :DC.W ResetFunc
```


For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

NOTE: *Each constant in the section `VectorTable` is defined as a word (2-byte constant) because the entry in the HC12 vector table is 16 bits wide. In the previous example, the constant `IRQInt` is initialized with the address of the label `IRQFunc`. In the previous example, the constant `XIRQInt` is initialized with the address of the label `XIRQFunc`. All labels specified as an initialization value must be defined, published (using XDEF), or imported (using XREF) before the vector table section. Forward referencing is not allowed in the DC directive. The statement `ORG $FFF2` specifies that the following section must start at address $FFF2.*

When developing a banked application, ensure that interrupt functions are located in the non-banked memory area.

The section should now be placed at the expected address. This is performed in the linker parameter file, shown in the next example.

Example:

```
LINK test.abs
NAMES
  test.o
END
SEGMENTS
  MY_ROM  = READ_ONLY  0x0800 TO 0x08FF;
  MY_RAM  = READ_WRITE 0x0A00 TO 0x0BFF;
END
PLACEMENT
  .data        INTO MY_RAM;
  .text        INTO MY_ROM;
END
INIT ResetFunc
ENTRIES
  *
END
```

NOTE: *The statement `ENTRY * END` switches smart linking off. If this statement is missing in the .prm file, the vector table will not be linked with the application because it is never referenced. The smart linker only links referenced objects in the absolute file.*

Freescale Semiconductor, Inc.

4.5 Smart Linking

Smart linking links referenced objects with the application. Application entry points are:

- The application `init` function
- The functions or constants located in an absolute section (section defined with ORG in the assembly source file)
- The function specified in a `VECTOR` command.

All previously listed entry points and the objects they referenced are automatically linked with the application. The user can specify additional entry points using the `ENTRIES` command in the PRM file.

4.5.1 Mandatory Linking from an Object

The user can choose to link non-referenced objects in an application. This may be useful to ensure that a software version number is linked with the application and stored in the final product EPROM. This may also be useful to ensure that a vector table, which has been defined as a constant table of function pointers or as a constant section, is linked with the application.

Example :

```
ENTRIES
   myVar1 myVar2 myProc1 myProc2
END
```

In this example, the variables `myVar1` and `myVar2` and functions `myProc1` and `myProc2` are specified to be additional entry points in the application.

Freescale Semiconductor, Inc.

4.5.2 Mandatory Linking from All Objects Defined in a File

The user can choose to link all objects defined in a specified object file.

Example:

```
ENTRIES
   myFile1.o:* myFile2.o:*
END
```

In this example, all objects (functions, variables, constant variables, or string constants) defined in *myFile1.o* and *myFile2.o* are specified as additional entry points in the application.

4.5.3 Switching Off Smart Linking for the Application

Switch smart linking off to link all objects in the application.

Example:

```
ENTRIES
   *
END
```

In this example, smart linking is switched off for the whole application. All objects, defined in one of the binary files that builds the application, are linked with the application.

4.5.4 Linking an Assembly Application

The example shows how to link an application.

When an application consists only of assembly files, the linker PRM file can be simplified. For instance:

- No startup structure is required.
- No stack initialization is required because the stack is directly initialized in the source file.
- No main function is required.
- An entry point in the application is required.
- All symbols referenced in the *.prm* file must be published (specified in an XDEF directive). No local symbol is defined in the assembler.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Example:

```
LINK    test.abs
NAMES   test.o test2.o END
SEGMENTS
  DIRECT_RAM = READ_WRITE 0x00000 TO 0x000FF;
  RAM_AREA   = READ_WRITE 0x00300 TO 0x07FFF;
  ROM_AREA   = READ_ONLY  0x08000 TO 0x0FFFF;
END
PLACEMENT
    myRegister         INTO DIRECT_RAM;
    .data              INTO RAM_AREA;
    .text              INTO ROM_AREA;
END
INIT Start ; Application entry point
VECTOR ADDRESS 0xFFFE Start;Initialize Reset Vector
```

In the previous example:

- All data sections defined in the assembly input files are allocated in the segment `RAM_AREA`.
- All code and constant sections defined in the assembly input files are allocated in the segment `ROM_AREA`.
- The START function defines an application entry point and a reset vector. START must be a global symbol defined in one of the assembly modules.

4.5.5 Warning Messages

An assembly application does not need a startup structure or root function.

Ignore these two warnings:

```
'WARNING: _startupData not found'
```

and

```
'WARNING: Function main not found'
```

Smart Linking — When an assembly application is linked, smart linking is performed on section level instead of object level. Sections containing referenced objects are linked with the application.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Examples for HC12: Assembly source file

```
            XDEF entry
dataSec1: SECTION
data1:    DS.W 1
dataSec2: SECTION
data2:    DS.W 2
codeSec:  SECTION
entry:
            NOP
            NOP
            LDX  #data1
            LDD  #5645
            STD  0, X
loop:     BRA loop
```

Linker *.prm* file

```
LINK   test.abs
NAMES  test.o END

SEGMENTS
   RAM_AREA   = READ_WRITE 0x00300 TO 0x07FFF;
   ROM_AREA   = READ_ONLY  0x08000 TO 0x0FFFF;
END
PLACEMENT
     .data             INTO RAM_AREA;
     .text             INTO ROM_AREA;
END
INIT entry
VECTOR ADDRESS 0xFFE entry
```

In the previous examples:

- The ENTRY function is defined as an application entry point and also specified as a reset vector.
- The data section `dataSec1` defined in the assembly input file is allocated in the segment `RAM_AREA` at address 0x300. This section is linked with the application because the label `data1` is referenced in the function `entry`.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

- The code section codeSec defined in the assembly input file is allocated in the segment ROM_AREA at address 0x8000. It is linked with the application because entry is the application entry point.
- The data section dataSec2 defined in the assembly input file is not linked with the application because the symbol data2 is never referenced.

The user can choose to switch smart linking off, so that assembly code and objects will be linked with the application.

For the previous example, the parameter file used to switch smart linking off will look like this:

```
LINK    test.abs
NAMES   test.o END

SEGMENTS
   RAM_AREA    = READ_WRITE 0x00300 TO 0x07FFF;
   ROM_AREA    = READ_ONLY  0x08000 TO 0x0FFFF;
END
PLACEMENT
     .data             INTO RAM_AREA;
     .text             INTO ROM_AREA;
END
INIT entry
VECTOR ADDRESS 0xFFE entry
ENTRIES * END
```

In the previous example:

- The ENTRY function is defined as an application entry point and also specified as a reset vector.
- The data section dataSec1 defined in the assembly input file is allocated in the segment RAM_AREA at address 0x300.
- The data section dataSec2 defined in the assembly input file is allocated next to the section dataSec1 at address 0x302.
- The code section codeSec defined in the assembly input file is allocated in the segment ROM_AREA at address 0x8000.

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

4.6 Program Startup

This section deals with advanced material and is relevant only for high-level language (ANSI C or C++) applications. First-time users of MCUez may skip this section. Standard startup modules are delivered with the MCUez programs and examples. Include startup modules to link the parameter file. For more information about startup modules, see the file *startup.txt* in the LIB subdirectory.

Prior to calling root function (`main`):

- Initialize the processor registers.
- Zero out memory.
- Copy initialization data from ROM to RAM.

Depending on the processor and application, different startup routines may be necessary. In MCUez, there are standard startup routines for every processor and memory model. Startup routines are based on a startup descriptor containing all information.

4.6.1 Startup Descriptor

The linker startup descriptor is declared as:

```
typedef struct{
  unsigned char *far beg;int size;
} _Range;
typedef struct{
    int size; unsigned char * far dest;
} _Copy;
typedef void (*_PFunc)(void);
typedef struct{
    _PFunc  *startup; /* address of startup desc */
} _LibInit;
typedef struct{
    _PFunc  *initFunc; /*address of init function*/
} _Cpp;
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
extern struct _tagStartup {
     unsigned short  flags;
     _PFunc          main;
     unsigned short  stackOffset;
     unsigned short  nofZeroOuts;
     _Range          *pZeroOut;
     _Copy           *toCopyDownBeg;
     unsigned short  nofLibInits;
     _LibInit        *libInits;
     unsigned short  nofInitBodies;
     _PFunc          *initBodies;
} _startupData;
```

The linker expects the `_startupData` variable to be declared somewhere in the application.

```
struct _tagStartup _startupData;
```

Fields of this `struct` are initialized by the linker and `struct` is allocated in ROM in the `.startData` section. If this variable is not declared, the linker does not create a startup descriptor. In this case, there is no `.copy` section and the stack is not initialized.

The fields have the following semantics:

`flags` Contains flags to detect special conditions at startup. Currently, two bits are used.

Table 4-8. Setting Startup Descriptor Flags

Bit Number	Set If ...
0	The application has been linked as a ROM library.
1	There is no stack specification.

This flag is tested in the startup code, to determine if the stack pointer should be initialized.

Freescale Semiconductor, Inc.

`main` A function pointer set to the application's root function. In a C program, this is usually function `main` unless a `MAIN` entry in the parameter file specifies another function as root. In a ROM library, this field is zeroed out. The standard startup code jumps to this address once initialization completes.

`stackOffset` Valid only if `flags=0`. This field contains the initial value of the stack pointer.

`nofZeroOuts` The number of `READ_WRITE` segments to fill with zero bytes at startup. This field is not required if there is no RAM memory area that should be initialized at startup.

NOTE: *Be careful because if the `nofZeroOuts` field is not present in the startup structure, the field `pZeroOut` must not be present either.*

`pZeroOut` A pointer to a vector with elements of type `_Range`. It has exactly `nofZeroOuts` elements, each describing a memory area to be cleared. This field is not required if there is no RAM memory area that should be initialized at startup. If this field is not present, the field `nofZeroOuts` must not be present.

`toCopyDownBeg` Contains the address of the first item to be copied from ROM to RAM at runtime. All data to be copied is stored in a contiguous piece of ROM memory and has this format:

$$CopyData = \{Size_{[2]}\ TargetAddr\ \{Byte\}^{Size}\}\ 0x0_{[2]}$$

The size is a binary number whose most significant byte is stored first. This field is not required. No RAM memory area should be initialized at startup.

`nofLibInits` The number of ROM libraries linked with the application that must be initialized at startup. This field is not required if no ROM libraries are linked with the application.

NOTE: *Be careful because if the `nofLibInits` field is not present in the startup structure, the field `libInits` must not be present.*

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

`libInits` — A vector of pointers to the `_startupData` records of all ROM libraries in the application. It has exactly `nofLibInits` elements. These addresses are needed to initialize the ROM libraries. This field is not required if no ROM libraries are linked with the application.

NOTE: *Be careful because if the `libInits` field is not present, the field `nofLibInits` must not be present.*

`nofInitBodies` — The number of C++ global constructors that must be executed prior to invoking the application root function. This field is not required if the application does not contain C++ modules. If this field is not present in the startup structure, the field `initBodies` must not be present.

`initBodies` — A pointer to a vector of function pointers containing addresses of the global C++ constructors. They are sorted in the order they need to be called. It has exactly `nofInitBodies` elements. If an application does not contain any C++ modules, the vector is empty. This field is not required if the application does not contain C++ modules. If this field is not present in the startup structure, the field `nofInitBodies` must not be present.

4.6.2 User-Defined Startup Structure

The user can define a startup structure. If the startup structure is changed, adapt the startup function to match the modifications.

Example: If there is no RAM area to initialize at startup and no ROM libraries and C++ modules, the user can define the startup structure as follows:

```
extern struct _tagStartup {
    unsigned short  flags;
    _PFunc          main;
    unsigned short  stackOffset;
} _startupData;
```


Freescale Semiconductor, Inc.

The startup code must be adapted accordingly:

```
extern void near _Startup(void) {
/*  purpose:  1)  initialize the stack
              2)  call main;
    parameters: NONE */
  do { /* forever; initialize program;
          call root-procedure */
    asm{
      LDD  _startupData.flags
      BNE  Initialize
      LDS  _startupData.stackOffset
    Initialize:
    }
    /* Here user defined code could be inserted,
       the stack can be used */
    /* call main() */
    (*_startupData.main)();
  } while(1); /* end loop forever */
}
```

NOTE: *Field names in the startup structure should not be changed. Fields inside the structure can be removed, but do not change names of the different fields.*

4.6.3 User-Defined Startup Routines

Two ways to replace the standard startup routine with a user-defined routine:

1. Provide a startup module containing a function named `_Startup` and link it with the application.
2. Implement a personal function and define it as an entry point for the application using the command `INIT`.

   ```
   INIT function_name
   ```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Section 5. Environment Variables

5.1 Contents

5.2 Introduction. 90

5.3 Linker Options 90
5.3.1 -E 91
5.3.2 -H 91
5.3.3 -L 92
5.3.4 -M. 92
5.3.5 -N 92
5.3.6 -O 93
5.3.7 -S 93
5.3.8 -V 94
5.3.9 -W1. 94
5.3.10 -W2. 94
5.3.11 -Wmsg8x3 95
5.3.12 -WmsgFb[v|m]. 95
5.3.13 -WmsgFi[v|m] 95
5.3.14 -WmsgNe 96
5.3.15 -WmsgNi 96
5.3.16 -WmsgNw 96

5.4 Setting Environment Variables in MCUez Shell 97
5.4.1 Path Variables 97

5.5 Variable Descriptions 97
5.5.1 GENPATH. 98
5.5.2 OBJPATH 99
5.5.3 LIBPATH. 99
5.5.4 ABSPATH 100
5.5.5 TEXTPATH. 100
5.5.6 SRECORD 101
5.5.7 ERRORFILE 102

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

5.2 Introduction

This section describes environment variables used by the MCUez linker. Environment variables are set in the *Paths or Additional* tab of the *MCUez Shell New Configuration* or *Current Configuration* dialog box. Refer to the *MCUez Installation and Configuration User's Manual*, Motorola document order number MCUEZINS/D. Environment variables that define paths (such as GENPATH, OBJPATH, ABSPATH, etc.) are used by the linker and other MCUez applications.

5.3 Linker Options

The MCUez linker offers a number of options to control linker operation. Options are composed of a hyphen (-) followed by one or more letters or digits, no spaces. Anything not starting with a hyphen is assumed to be the name of a parameter file to be linked.

All linker options (except -V and -H) can be defined in the linker **Option Settings** dialog box. Refer to **Figure 2-8** for an illustration of the **Option Settings** dialog box.

All options, including -V and -H, can be specified on the command line in the linker. Specify the parameter file followed by a space then the linker option. See **Figure 5-1**.

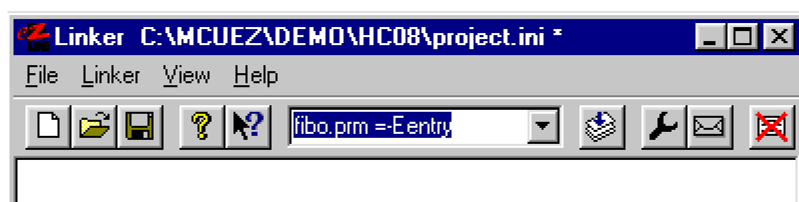


Figure 5-1. Linker Command Line

Options set in the **Option Settings** dialog box remain indefinitely for all linking sessions, until changed again by the user. Options specified on the command line are temporary and apply to the current linking session.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Options are not case sensitive. For example:

`fibo.prm -otest.abs` is the same as `FIBO.PRM -OTEST.ABS`

Options are listed in alphabetical order in the next subsections.

5.3.1 -E

-E:	Define application entry point.
Syntax:	`<parameter file> <option>=<function>`
Arguments:	<function>: Name of function that represents entry point in the application
Default:	None
Description:	This option specifies the application entry point. When the entry point is located in an assembly object file, the corresponding symbol must be a global symbol (specified in an XDEF directive).
Example:	`fibo.prm -E=entry` Similarly, the command `INIT entry` can be put in the *.prm* file.
See also:	**4.4.2 INIT: Specify Application Entry Point**

5.3.2 -H

-H:	Help
Syntax:	`<option>`
Arguments:	None
Default:	None
Description:	Displays the list of linker options and a brief description
Example:	`-H`
See also:	None

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

5.3.3 -L

-L:	Add path to search path.
Syntax:	`<parameter file> <option><path>`
Arguments:	<path> Complete directory path
Default:	None
Description:	Add another path to be searched.
Example:	`test.prm =-LC:\MCUez\misc`
See also:	None

5.3.4 -M

-M:	Generate map file.
Syntax:	`<parameter file> <option>`
Arguments:	None
Default:	None
Description:	This option causes a map file to be generated.
Example:	`test.prm -M`
See also:	**4.4.5 MAPFILE: Configure Map File**

5.3.5 -N

-N:	Show notification box in case of errors.
Syntax:	`<parameter file> <option>`
Arguments:	None
Default:	None
Description:	This option enables an error dialog box to be displayed.
Example:	`test.prm -N`
See also:	None

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

5.3.6 -O

-O: Define absolute filename.

Syntax: `<parameter file> <option><FileName>`

Arguments: <FileName>: Name of absolute file

Default: None

Description: This option defines the name of the *.abs* file to be generated.

Example: `test.prm -Otest.abs`

Similarly, the user can use the command `LINK test.abs` in the *.prm* file.

See also: **4.4.3 LINK: Specify Name of Output File**

5.3.7 -S

-S: Do not generate *DWARF* information.

Syntax: `<parameter file> <option>`

Arguments: None

Default: None

Description: This option excludes *DWARF* sections from being generated in the absolute file. This will reduce the amount of memory used on the PC.

Example: `test.prm -S`

See also: None

NOTE: *If the absolute file does not contain DWARF information, the file cannot be debugged.*

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

5.3.8 -V

-V:	Prints the linker version
Syntax:	`<option>`
Arguments:	None
Default:	None
Description:	Prints the linker version and project directory
Example:	`-V`
See also:	None

5.3.9 -W1

-W1:	Don’t print information messages.
Syntax:	`<parameter file> <option>`
Arguments:	None
Default:	None
Description:	Suppresses all information messages. Warning and error messages are printed.
Example:	`test.prm -W1`
See also:	None

5.3.10 -W2

-W2:	Don’t print information and warning messages.
Syntax:	`<parameter file> <option>`
Arguments:	None
Default:	None
Description:	Suppresses all information and warning messages. Only errors are printed.
Example:	`test.prm -W2`
See also:	None

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

5.3.11 -Wmsg8x3

-Wmsg8x3:	Convert filenames to DOS 8.3 format.
Syntax:	`<parameter file> <option>`
Arguments:	None
Default:	None
Description:	Reduces long filenames to eight characters plus extension
Example:	`test.prm -Wmsg8x3`
See also:	None

5.3.12 -WmsgFb[v|m]

-WmsgFb[v|m]:	Sets message file format for batch mode
Syntax:	`<parameter file> <option>`
Arguments:	`[v|m]`
Default:	M
Description:	Sets the message file format to verbose mode or Microsoft format. Verbose mode displays more information than the Microsoft format.
Example:	`test.prm -WmsgFbv`
See also:	None

5.3.13 -WmsgFi[v|m]

-WmsgFi[v|m]:	Sets message format for interactive mode
Syntax:	`<parameter file> <option>`
Arguments:	`[v|m]`
Default:	V
Description:	Sets the message file format to verbose mode or Microsoft format. Verbose mode displays more information than the Microsoft format.
Example:	`test.prm -WmsgFim`
See also:	None

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

5.3.14 -WmsgNe

-WmsgNe:	Sets maximum number of error messages
Syntax:	`<parameter file> <option><value>`
Arguments:	None
Default:	None
Description:	Sets the maximum number of error messages to be generated before the process is halted. Enter a value between 0 and 100.
Example:	`test.prm -WmsgNe10`
See also:	None

5.3.15 -WmsgNi

-WmsgNi:	Sets maximum number of information messages
Syntax:	`<parameter file> <option><value>`
Arguments:	None
Default:	None
Description:	Sets the maximum number of information messages to be generated. Enter a value between 0 and 100.
Example:	`test.prm -WmsgNi10`
See also:	None

5.3.16 -WmsgNw

-WmsgNw:	Sets maximum number of warning messages
Syntax:	`<parameter file> <option><value>`
Arguments:	None
Default:	None
Description:	Sets the maximum number of warning messages to be generated. Enter a value between 0 and 100.
Example:	`test.prm -WmsgNw10`
See also:	None

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

5.4 Setting Environment Variables in MCUez Shell

The syntax for environment variables set in the shell is:
`Variable=Definition`

NOTE: *No spaces are allowed in the definition of an environment variable.*

Example:

```
GENPATH=C:\INSTALL\LIB;
```

5.4.1 Path Variables

Environment variables that contain paths indicate where to look for files. A path list is a list of directory names separated by semicolons or a directory name preceded by an asterick. If a directory name is preceded by an asterisk (*), the programs recursively search the whole directory tree for a file, not just the given directory. Directories are searched in the order they appear in the path list.

Syntax:

```
DirSpec;DirSpec;DirSpec
*DirectoryName
```

Example:

```
GENPATH=C:\INSTALL\LIB;D:\PROJECTS\TESTS;
LIBPATH=*C:\INSTALL\LIB
```

5.5 Variable Descriptions

This section describes these variables:

- GENPATH
- OBJPATH
- LIBPATH
- ABSPATH
- TEXTPATH
- SRECORD
- ERRORFILE

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

5.5.1 GENPATH

Syntax: `GENPATH=<path>`

Arguments: `<path>`: Path separated by semicolons, without spaces

Description: The linker will look for the *.prm* file in the project directory, then in the directories listed in the environment variable GENPATH. The object and library files specified in the linker *.prm* file are searched for in the project directory, then in directories listed in the environment variables OBJPATH and LIBPATH, and finally in directories specified in GENPATH.

NOTE: *If a directory specification starts with an asterisk (*), the directory tree is searched recursively. Within one level in the tree, the search order of the subdirectories is indeterminate.*

Example: `GENPATH=\obj;..\..\lib;`

MCUez Shell: Click the **Change...** button to open the **Current Configuration** dialog box.

Select the **Paths** tab.
In the **Configure** selection box, select **General Path**.

Enter directories in the list box (one directory on each line).

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

5.5.2 OBJPATH

Syntax: `OBJPATH=<path>`

Arguments: `<path>`: Paths separated by semicolons, without spaces

Description: The linker searches the project directory for the object files specified in the linker *.prm* file. The linker then searches the directories specified in the environment variables OBJPATH and GENPATH.

Example: `OBJPATH=\sources\bin;..\..\headers;`

MCUez Shell: Select **Change...** button to open the **Current Configuration** dialog box.

Select the **Paths** tab.

In the **Configure** selection box, select **Object**.

Enter directories in the list box (one directory on each line).

5.5.3 LIBPATH

Syntax: `LIBPATH=<path>`

Arguments: `<path>`: Paths separated by semicolons, without spaces

Description: The linker searches the project directory for the library files specified in the linker *.prm* file. The linker then searches the directories specified in the environment variables OBJPATH and GENPATH.

Example: `LIBPATH=\sources\bin;..\..\lib;`

MCUez Shell: Select **Change...** button to open the **Current Configuration** dialog box.

Select the **Paths** tab.

In the **Configure** selection box, select **Library**.

Enter directories in the list box (one directory on each line).

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

5.5.4 ABSPATH

Syntax: `ABSPATH=<path>`

Arguments: `<path>`: Paths separated by semicolons, without spaces

Description: Set this environment variable to store the absolute files created by the linker in the first directory specified. If ABSPATH is not set, the generated absolute files will be stored in the directory where the parameter file was found.

Example: `ABSPATH=\sources\bin;..\..\headers;`

MCUez Shell: Click the **Change...** button to open the **Current Configuration** dialog box.

Select the **Paths** tab.

In the **Configure** selection box, select **Absolute**.

Enter directories in the list box (one directory on each line).

5.5.5 TEXTPATH

Syntax: `TEXTPATH=<path>`

Arguments: `<path>`: Paths separated by semicolons, without spaces

Description: Set this environment variable to store the map file created by the linker in the first directory specified. If TEXTPATH is not set, the generated map file will be stored in the directory where the *.prm* file is located.

Example: `TEXTPATH=\sources;\usr\local\txt;`

MCUez Shell: Click the **Change...** button to open the **Current Configuration** dialog box.

Select the **Paths** tab.

In the **Configure** box, select **Text**.

Enter directories in the list box (one directory on each line).

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

5.5.6 SRECORD

Syntax: `SRECORD=<RecordType>`

Arguments: `<Record Type>`: Specify the type of Motorola S record that must be generated. The value can be S1, S2, or S3.

Description: When this environment variable is defined, the linker will generate a Motorola S file containing records for the specified type (S1 records when S1 is specified, S2 records when S2 is specified, and S3 records when S3 is specified).

NOTE: *If SRECORD is set, the user is responsible for specifying the appropriate S record type. If S1 is specified while the code is loaded above 0xFFFF, the Motorola S file generated will not be correct, since the addresses will be truncated to 2-byte values.*

If this variable is not set, the type of S record generated will depend on the size of the address loaded. If the address can be coded on two bytes, an S1 record is generated. If the address is coded on three bytes, an S2 record is generated. Otherwise, an S3 record is generated.

Example: `SRECORD=S2`

MCUez Shell: Click the **Change...** button to open the **Current Configuration** dialog box.

Select the **Additional** tab.

Enter the environment variable in the list box.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

5.5.7 ERRORFILE

Syntax: `ERRORFILE=<filename>`

Arguments: `<filename>`: Filename with format specifiers

Description: ERRORFILE specifies the error file used by the linker.

Possible format specifiers are:

%n: Substitute with the filename, without the path.
%p: Substitute with the path of the source file.
%f: Substitute with the full path and filename

Example: `ERRORFILE=MyErrors.err`

Writes all errors to the file *MyErrors.err* in the project directory.

`ERRORFILE=\tmp\errors`

Writes all errors to the file *errors* in the directory *\tmp*.

`ERRORFILE=%f.err`

Writes all errors to a file with the same name as the source file, but with extension *.err*. The error file is placed in the same directory as the source file. For example, if the file *\sources\test.prm* is linked, an error list file *\sources\test.err* will be generated.

`ERRORFILE=\dir1\%n.err`

For a source file *test.prm*, an error list file *\dir1\test.err* is generated.

`ERRORFILE=%p\errors.txt`

For a source file *\dir1\dir2\test.prm*, an error list file *\dir1\dir2\errors.txt* will be generated.

If ERRORFILE is not set, errors are written to the default error file. The default error file is determined by how the assembler is configured and started. If a filename is provided on the assembler command line, errors are written to the *EDOUT* file in the project directory. If no filename is provided, errors are written to the *err.txt* file.

MCUez Shell: Open the **Current Configuration** dialog box.

Select the **Additional** tab.

Enter the environment variable definition in the list box.

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.

Section 6. Linker Messages

6.1 Contents

6.2 Introduction. 108
6.3 Linker Messages Reference 108
6.3.1 L1: Unknown Message Occurred 109
6.3.2 L2: Message Overflow, Skipping <type> Messages 109
6.3.3 L64: Line Continuation Occurred in <FileName>. 109
6.3.4 L1000: <Command Name> not Found 110
6.3.5 L1001: <Command Name> Multiply Defined. 111
6.3.6 L1002: Command <Command Name> Overwritten by Option <Option Name> 112
6.3.7 L1003: Only a Single SEGMENTS or SECTIONS Block is Allowed. 113
6.3.8 L1004: <Separator> Expected 113
6.3.9 L1005: Fill Pattern Will Be Truncated (>0xFF) 114
6.3.10 L1006: <Token> not Allowed 114
6.3.11 L1007: <Character> not Allowed in Filename (Restriction). . . . 115
6.3.12 L1008: Only Single Object Allowed at Absolute Address 116
6.3.13 L1009: Segment Name <Segment Name> Unknown 117
6.3.14 L1010: Section Name <section name> Unknown 118
6.3.15 L1011: Incompatible Segment Qualifier: <Qualifier1> in Previous Segment and <Qualifier2> in <Segment Name> 119
6.3.16 L1012: Segment is not Aligned on a <bytes> Boundary 120
6.3.17 L1015: No Binary Input File Specified 120
6.3.18 L1016: File <Filename> Found Twice in NAMES Block. 121
6.3.19 L1037: ***** Linking of <parameter file> Failed **** 121
6.3.20 L1038: Success. Executable File Written to <absfile> 121
6.3.21 L1039: Limited Version. Too Many Objects or Code Linked. . . 122
6.3.22 L1050: Running <versiontype>. 122
6.3.23 L1052: User Requested Stop 122
6.3.24 L1100: Segments <Segment1 Name> and <Segment2 Name> Overlap. 123

For More Information On This Product,
Go to: www.freescale.com

6.3.25 L1102: Out of Allocation Space in Segment <Segment Name> at Address <First Address Free>. 124
6.3.26 L1103: <Section Name> not Specified in PLACEMENT Block 125
6.3.27 L1104: Absolute Object <Object Name> Overlaps with Segment <Segment Name>. 126
6.3.28 L1105: Absolute Object <object name> Overlaps with Another Absolute Allocated Object or with a Vector. 127
6.3.29 L1106: <Object Name> not Found 128
6.3.30 L1107: <Object Name> not Found 129
6.3.31 L1109: <Segment Name> Appears Twice in SEGMENTS Block. 130
6.3.32 L1110: <Segment Name> Appears Twice in PLACEMENT Block 131
6.3.33 L1111: <Section Name> Appears Twice in PLACEMENT Block 132
6.3.34 L1112: <Section name> Section Has Segment Type <Segment Qualifier> (Illegal). 132
6.3.35 L1113: <Section name> Section Has Segment Type <Segment Qualifier> (Illegal). 134
6.3.36 L1114: <Section Name> Section Has Segment Type <Segment Qualifier> (Initialization Problem) 135
6.3.37 L1115: Function <Function Name> not Found 137
6.3.38 L1118: Vector Allocated at Absolute Address <Address> Overlaps with Another Vector or an Absolute Allocated Object 138
6.3.39 L1119: Vector Allocated at Absolute Address <Address> Overlaps with Sections Placed in Segment <Segment Name> 139
6.3.40 L1120: Vector Allocated at Absolute Address <Address> Placed in Segment <Segment Name>, Which Has No READ_ONLY Qualifier 140
6.3.41 L1121: Out of Allocation Space at Address <Address> for .copy Section 140
6.3.42 L1122: Section .copy Must Be Last Section in Section List 141
6.3.43 L1123: Invalid Range Defined for Segment <Segment Name> — End Address Must Be Bigger Than Start Address 142
6.3.44 L1124: '+' or '-' Should Directly Follow Filename. 143

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

6.3.45 L1125: In Small Memory Model, Code and Data Must Be Located on Bank 0 . . . 144
6.3.46 L1127: Object Allocated Outside of Segment Bounds (HC12). . 145
6.3.47 L1200: Both STACKTOP and STACKSIZE Defined . . . 146
6.3.48 L1201: No Stack Defined . . . 147
6.3.49 L1202: Stack Cannot Be Allocated on More Than One Segment . . . 148
6.3.50 L1203: STACKSIZE Command Defines a Size of <Size> But .stack Specifies a Stacksize of <Size>. . . 149
6.3.51 L1204: STACKTOP Command Defines Initial Value of <Stack Top> But .stack Specifies Initial Value of <Initial Value> . . . 151
6.3.52 L1205: STACKTOP Command Incompatible with .stack Being Part of List of Sections . . . 152
6.3.53 L1206: Stack Overlaps with a Segment Which Appears in PLACEMENT Block . . . 153
6.3.54 L1207: STACKSIZE Command is Missing . . . 154
6.3.55 L1301: Cannot Open File <Filename> . . . 155
6.3.56 L1302: File <Filename> not Found . . . 155
6.3.57 L1303: <Filename> is not a Valid ELF File . . . 156
6.3.58 L1304: <Filename> is not a Valid Hex File . . . 156
6.3.59 L1305: <Filename> is not an ELF Format Object File (ELF Object File Expected) . . . 156
6.3.60 L1309: Cannot Open <File> . . . 157
6.3.61 L1400: Incompatible Processor: <Processor Name> in Previous Files and <Processor Name> in Current File. . . . 157
6.3.62 L1401: Incompatible Memory Model: <Memory Model Name> in Previous Files and <Memory Model Name> in Current File . . . 157
6.3.63 L1403: Unknown Processor <Processor Constant>. . . 158
6.3.64 L1404: Unknown Memory Model <Memory Model Constant> . . . 158
6.3.65 L1501: <Symbol Name> Cannot be Moved in Section <Section Name> (Invalid Qualifier <Segment Qualifier>) . . . 159
6.3.66 L1502: <Object Name> Cannot be Moved from Section <Source Section Name> to Section <Destination Section Name> . . . 160
6.3.67 L1503: <Object Name> (from file <Filename>) Cannot be Moved from Section <Source Section Name> to Section <Destination Section Name> . . . 161

Freescale Semiconductor, Inc.

6.3.68 L1504: <Object Name> (from section <Section Name>) Cannot be Moved from Section <Source Section Name> to Section <Destination Section Name> . . . 162
6.3.69 L1600: Main Function Detected in ROM Library . . . 163
6.3.70 L1601: Startup Function Detected in ROM Library . . . 163
6.3.71 L1620: Bad Digit in Binary Number . . . 163
6.3.72 L1621: Bad Digit in Octal Number . . . 163
6.3.73 L1622: Bad Digit in Decimal Number . . . 163
6.3.74 L1623: Number too Big . . . 164
6.3.75 L1624: Ident too Long. Cut after 31 Characters . . . 164
6.3.76 L1625: Comment not Closed . . . 164
6.3.77 L1626: Unexpected End of File . . . 164
6.3.78 L1627: PRESTART Command not Supported Yet . . . 165
6.3.79 L1628: HEXFILE Command not Supported Yet . . . 165
6.3.80 L1629: START_DATA Command not Supported Yet . . . 165
6.3.81 L1700: File <Filename> Should Contain DWARF Information . . . 165
6.3.82 L1701: Startup Data Structure is Empty . . . 166
6.3.83 L1800: Read Error in <File> . . . 166
6.3.84 L1803: Out of Memory in <Function Name> . . . 166
6.3.85 L1804: No ELF Section Header Table Found in <Filename> . . . 166
6.3.86 L1806: ELF File <Filename> Appears to be Corrupted . . . 167
6.3.87 L1808: String Overflow in <Function Name>, Contact Vendor . . . 167
6.3.88 L1809: Section <Section Name> Located in a Segment with Invalid Qualifier . . . 167
6.3.89 L1811: Symbol <Symbol Number> - < Symbol Name> Duplicated in <First Filename> and <Second Filename> . . . 167
6.3.90 L1818: Symbol <Symbol Number> - < Symbol Name> Duplicated in <First Filename> and <Second Filename> . . . 168
6.3.91 L1820: Weak Symbol <Symbol Name> Duplicated in <First Filename> and <Second Filename> . . . 168
6.3.92 L1821: Symbol <id1> Conflicts with <id2> in File <File> (Same Code) . . . 168
6.3.93 L1822: Symbol <Symbol Name> in File <Filename> is Undefined . . . 168
6.3.94 L1823: External Object <Symbol Name> in <Filename> Created by Default . . . 169
6.3.95 L1824: Invalid Mark Type for <Ident> . . . 169
6.3.96 L1826: Can't Read File. <Filename> is not an ELF Library Containing ELF Objects (ELF Objects Expected) . . . 169

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.97 L1902: <Cmd> Command not Supported 169
6.3.98 L1903: Unexpected Symbol in Link Parameter File 170
6.3.99 L1905: Invalid Section Attribute for Program Header 170
6.3.100 L1906: Fixup Out of Buffer (<Obj> Referenced at Offset <Address>). 170
6.3.101 L1907: Fixup Overflow in <Object>, Type <objType> at Offset <Address> 170
6.3.102 L1908: Fixup Error in <Object>, Type <objType> at Offset <Address> 171
6.3.103 L1910: Invalid Section Attribute for Program Header 171
6.3.104 L1911: Program Header End is not Aligned on the End of a Section 171
6.3.105 L1912: Object <obj> Overlaps with Another (last addr: <addr>, Object Address: <objadr> 171
6.3.106 L1913: Object Filler Overlaps with Something Else 171
6.3.107 L1914: Invalid Object: <Object> 172
6.3.108 L1915: Gap in <Ident> at <address> before <Object> is too Big 172
6.3.109 L1916: Section Name <Section> is too Long. Name is Cut to 90 Characters Length 172
6.3.110 L1919: Duplicate Definition of <Object> in Library File(s) <File1> and/or <File2> Discarded 172
6.3.111 L1921: Marking: Too Many Nested Procedure Calls 173
6.3.112 L1922: File <filename> Has DWARF Data of Different Version, DWARF Data may not be Generated 173
6.3.113 L1927: Fixups for DWARF Section <sectionname> not Correctly Generated 173
6.3.114 L1928: Limitation: Code Size <num> 173
6.3.115 L1929: Limitation: Too many Mections (<num>). 174
6.3.116 L1930: Unknown Fixup Type in <ident>, Type <type>, at Offset <offset> 174
6.3.117 L1931: Program Header Begin is not Aligned on the Beginning of a Section 174
6.3.118 L1932: Program Header Overflow in <name> at <index> 174
6.3.119 L1933: ELF: <details> Warning 174
6.3.120 L1934: ELF: <details> Error 175
6.3.121 L1936: ELF Output: <details> Error 176
6.3.122 L1938: Type Clash in Segment (Corrupt Object: <name>). 177
6.3.123 L4000: Could not Open Object File (<objFile>) in NAMES List 177
6.3.124 L4001: Link Parameter File <PRMFile> not Found 177

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

6.3.125 L4002: NAMES Section was not Found in Linker Parameter File <PRM File> 177
6.3.126 L4004: Linking <PRM File> as ELF/DWARF Format Link Parameter File. 178
6.3.127 L4005: Illegal File Format of Object File (<objFile>) in NAMES List 178
6.3.128 L4006: Failed to Create Temporary File 178
6.3.129 L4007: Include File Nesting too Deep in Link Parameter File. 178
6.3.130 L4008: Include File <includefile> not Found 178

6.2 Introduction

This chapter lists and defines all messages generated by the MCUez linker.

6.3 Linker Messages Reference

Four types of messages are generated by the linker.

1. Information — A message is displayed and linking continues. Information messages do not interrupt linking and provide programming related information.
2. Warning — A message is displayed and linking continues. Warning messages indicate possible programming errors.
3. Error — A message is displayed and linking stops. Error messages indicate illegal syntax in the parameter (*.prm*) file.
4. Fatal — A message is displayed and linking is aborted. A fatal message indicates a severe error.

Linker messages are identified by a message code (L for linker) and a 4- to 5-digit number. Messages are documented in increasing order. Each message is described by its type, a description, an example (if available), and tips to fix a problem.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.1 L1: Unknown Message Occurred

Type: Fatal

Description: The linker tried to emit an undefined message. This is an internal error.

Tip: None

6.3.2 L2: Message Overflow, Skipping <type> Messages

Type: Information

Description: The maximum number of messages of a specific type have been displayed. The number of messages to display is controlled by the -WmsgNi, -WmsgNw, and -WmsgNe options.

Tip: Increase maximum setting in the **Message** tab of the **Option Settings** dialog box.

6.3.3 L64: Line Continuation Occurred in <FileName>

Type: Information

Description: In a parameter file, the back slash character (\) at the end of a path is interpreted as a line continuation function. The MS-DOS path separation character is also a slash.

Example:

```
...
LIBPATH=c:\mcuez\lib\
OBJPATH=c:\mcuez\work
...
```

Is interpreted by the compiler as:

```
...
LIBPATH=c:\mcuez\libOBJPATH=c:\mcuez\work
...
```

Tip: Enter a period (.) after paths that end with a slash or remove the trailing slash.

```
LIBPATH=c:\mcuez\lib\.
OBJPATH=c:\mcuez\work
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

6.3.4 L1000: <Command Name> not Found

Type: Error

Description: This message is generated when a mandatory linker command is missing from the parameter file. Mandatory commands are:

- `LINK` — Contains the name of the absolute file to generate. If the option –O is specified on the command line and the `LINK` command is missing from the parameter file, this message is not generated.
- `NAMES` — Lists the files building the application
- `PLACEMENT` — Associates at least the predefined sections `.text` and `.data` with a memory range

If `LINK` command is missing, the message is **LINK not found**.

If `NAMES` command is missing, the message is **NAMES not found**.

If `PLACEMENT` command is missing, message is **PLACEMENT not found**.

Example:

```
NAMES   fibo.o start12s.o ansis.lib END
SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
PLACEMENT
    .text   INTO MY_ROM;
    .data   INTO MY_RAM;
    .stack  INTO MY_STK;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Insert missing command in the PRM file.

```
LINK    fibo.abs
NAMES   fibo.o start12s.o ansis.lib END
SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
PLACEMENT
    .text   INTO MY_ROM;
    .data   INTO MY_RAM;
    .stack  INTO MY_STK;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.5 L1001: <Command Name> Multiply Defined

Type: Error

Description: This message is generated when a linker command is detected more than once in the parameter file.

These linker commands cannot be specified more than once in a parameter file.

- `LINK` — Contains the name of the absolute file to generate
- `NAMES` — Lists files building the application
- `SEGMENTS` — Associates a name with a memory area
- `PLACEMENT` — Sections are assigned to a memory range
- `ENTRIES` — Lists objects linked with the application
- `MAPFILE` — Specifies information to be stored in the map file
- `MAIN` — Defines the application main function
- `INIT` —Defines the application entry point
- `STACKSIZE` — Defines the stack size
- `STACKTOP` — Defines the stack pointer initial value

When the `LINK` command is detected more than once, the message will be:

```
LINK multiple defined
```

Example:

```
LINK   fibo.abs
NAMES  fibo.o start12s.o ansis.lib END
SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
END
PLACEMENT
    .text   INTO MY_ROM;
    .data   INTO MY_RAM;
    .stack  INTO MY_STK;
END
LINK   fibo.abs
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Tip: Remove one of the duplicated commands.

```
LINK   fibo.abs
NAMES  fibo.o start12s.o ansis.lib END
SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
END
PLACEMENT
    .text   INTO MY_ROM;
    .data   INTO MY_RAM;
    .stack  INTO MY_STK;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

6.3.6 L1002: Command <Command Name> Overwritten by Option <Option Name>

Type: Warning

Description: This message is generated when a command line option overrides a command in the parameter file.

<command name>: Name of the command in the *.prm* file

<option name>: Linker command line option

Commands that may be overridden by a command line option are:

- `LINK` — Overridden by the option –O; defines the output filename
- `MAPFILE` — Overridden by the option –M; enables generation of the map file
- `INIT` — Overridden by the option –E; defines the application entry point

When the `LINK` command is detected in the parameter file and the option –O is specified on the command line, this message is generated:

```
Command LINK overwritten by option -O
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.7 L1003: Only a Single SEGMENTS or SECTIONS Block is Allowed

Type: Error

Description: This error occurs when the parameter file contains both a SECTIONS and a SEGMENTS block. The SECTIONS block is a synonym for the SEGMENTS block. It is supported for compatibility with an older version of the parameter file.

Example:

```
LINK    fibo.abs
NAMES   fibo.o start12s.o ansis.lib END
SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
END
SECTIONS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
PLACEMENT
    .text   INTO MY_ROM;
    .data   INTO MY_RAM;
    .stack  INTO MY_STK;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Remove either the SEGMENTS or SECTIONS block.

6.3.8 L1004: <Separator> Expected

Type: Error

Description: This message is generated when the specified <separator> is missing from an expected position:

<separator>: character or expression

Example:

```
SEGMENTS
  MY_RAM    = READ_WRITE 0x800 TO 0x8FF
              ALIGN [2TO 4, 4]
                          ^
ERROR: : expected.
```

Tip: Insert the specified separator at the expected position.

Freescale Semiconductor, Inc.

6.3.9 L1005: Fill Pattern Will Be Truncated (>0xFF)

Type: Warning

Description: This message is generated when the constant specified as a fill pattern cannot be coded on a byte. The constant truncated to a byte value will be used as the fill pattern.

Example:

```
SEGMENTS
    MY_RAM = READ_WRITE 0x0800 TO 0x8FF FILL 0xA34;
END
```

Tip: To avoid this message, split the constant into 2-byte constants.

Example:

```
SEGMENTS
   MY_RAM = READ_WRITE 0x0800 TO 0x8FF FILL 0xA 0x34;
END
```

6.3.10 L1006: <Token> not Allowed

Type: Error

Description: This message is generated when a filename followed by an asterick (*) is specified in an OBJECT_ALLOCATION or LAYOUT block. This is not possible, because a section is either a read-only or a read/write section. When all objects defined in a file are moved to a section, the destination section will contain both code and variables. This is logically not possible.

Example:

```
OBJECT_ALLOCATION
    fibo.o:* INTO mySec;
          ^
ERROR: * not allowed
END
```

Tip: Move either all functions, variables, or constants to the destination section.

Example:

```
OBJECT_ALLOCATION
    fibo.o:CODE[*] INTO mySec;
END
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.11 L1007: <Character> not Allowed in Filename (Restriction)

Type: Error

Description: A filename specified in the parameter file contains an illegal character.

These specific characters are not allowed in a filename:

- Colon (:) — Used as separator to specify a local object (function or variable) in a parameter file
- Semi-colon (;) — Used as delimiter for a command line in a LAYOUT or OBJECT_ALLOCATION block
- Greater than symbol (>) — Used as a separator to refer to an object located in a section inside a LAYOUT or OBJECT_ALLOCATION block
- Plus and minus (+ and –) — This may cause a problem when used as a filename suffix in the NAMES block.

Example:

```
NAMES
    file:1.o;
         ^
ERROR: ':' or '>' not allowed in filename (restric-
tion)
END
```

or

```
NAMES
    file1.o file>2.lib;
                  ^
ERROR: ':' or '>' not allowed in filename (restric-
tion)
END
```

Tip: Change the filename and avoid the illegal characters.

Freescale Semiconductor, Inc.

6.3.12 L1008: Only Single Object Allowed at Absolute Address

Type: Error

Description: Multiple objects are placed at an absolute address in an OBJECT_ALLOCATION block. Only single objects are allowed there.

Example:

```
OBJECT_ALLOCATION
    var1 var2  AT 0x0800;
                  ^
ERROR: Only single object allowed at absolute address
END
```

or

```
OBJECT_ALLOCATION
     file.o:DATA[*] AT 0x900;
                       ^
ERROR: Only single object allowed at absolute address
END
```

Tip: Specify the objects on separate lines.

Example:

```
OBJECT_ALLOCATION
    var1  AT 0x0800;
    var2  AT 0x0802;
END
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

6.3.13 L1009: Segment Name <Segment Name> Unknown

Type: Error

Description: Segment specified in a PLACEMENT or LAYOUT command line was not previously defined in the SEGMENTS block.

<segment name>: name of unknown segment

Example:

```
LINK   fibo.abs
NAMES  fibo.o start12s.o ansis.lib END
SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
END
PLACEMENT
    .text   INTO ROM_AREA;
                         ^
ERROR: Segment Name ROM_AREA unknown
    .data   INTO MY_RAM;
    .stack  INTO MY_STK;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Define the segment names in the SEGMENTS block.

Example:

```
LINK   fibo.abs
NAMES  fibo.o start12s.o ansis.lib END
SEGMENTS
    RAM_AREA = READ_WRITE 0x800 TO 0x80F;
    ROM_AREA = READ_ONLY  0x810 TO 0xAFF;
    STK_AREA = READ_WRITE 0xB00 TO 0xBFF;
END
PLACEMENT
    .text   INTO ROM_AREA;
    .data   INTO RAM_AREA;
    .stack  INTO STK_AREA;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

6.3.14 L1010: Section Name <section name> Unknown

Type: Error

Description: The section name specified in a command in the OBJECT_ALLOCATION block was not previously specified in the PLACEMENT block.

<section name>: name of unknown section

Example:

```
LINK   fibo.abs
NAMES  fibo.o start12s.o ansis.lib END
SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
END
PLACEMENT
    .text   INTO MY_ROM;
    .data   INTO MY_RAM;
    .stack  INTO MY_STK;
END
OBJECT_ALLOCATION
    fibo.o:DATA[*] IN dataSec;
                             ^
ERROR: Section Name dataSec unknown
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Specify the section in the PLACEMENT block.

Example:

```
LINK   fibo.abs
NAMES  fibo.o start12s.o ansis.lib END
SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
END
PLACEMENT
    .text           INTO MY_ROM;
    .data, dataSec  INTO MY_RAM;
    .stack          INTO MY_STK;
END
OBJECT_ALLOCATION
    fibo.o:DATA[*] IN dataSec;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

6.3.15 L1011: Incompatible Segment Qualifier: <Qualifier1> in Previous Segment and <Qualifier2> in <Segment Name>

Type: Error

Description: Two segments specified in the same statement in the PLACEMENT block are not defined with the same qualifier.

<qualifier1>: Segment qualifier associated with the previous segment in the list. This qualifier may be READ_ONLY, READ_WRITE, NO_INIT, or PAGED.

<qualifier2> Segment qualifier associated with the current segment in the list. This qualifier may be READ_ONLY, READ_WRITE, NO_INIT, or PAGED.

<segment name >: Name of the current segment in the list

Example:

```
LINK   fibo.abs
NAMES  fibo.o start12s.o ansis.lib END
SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    SEC_RAM= READ_WRITE 0x020 TO 0x02F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
END
PLACEMENT
    .data     INTO MY_RAM;
    .text     INTO MY_ROM, SEC_RAM;
                                 ^
ERROR: Incompatible segment qualifier: READ_ONLY in
previous segment and READ_WRITE in SEC_RAM
    .stack    INTO MY_STK;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Modify the qualifier associated with the specified segment.

Example:

```
LINK   fibo.abs
NAMES  fibo.o start12s.o ansis.lib END
SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    SEC_ROM= READ_ONLY  0x020 TO 0x02F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
END
PLACEMENT
    .data     INTO MY_RAM;
    .text     INTO MY_ROM, SEC_ROM;
    .stack    INTO MY_STK;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.16 L1012: Segment is not Aligned on a <bytes> Boundary

Type: Warning

Description: Some targets (M•CORE) require aligned access for some objects.

Example (M•CORE): All 4-byte accesses must be aligned to four. According to the EABI, 8-byte doubles must be aligned to eight. But if an 8-byte structure only contains chars, then alignment is not needed.

Tip: Check whether the section contains objects that must be aligned.

6.3.17 L1015: No Binary Input File Specified

Type: Error

Description: No filenames specified in the NAMES block

Example:

```
LINK    fibo.abs
NAMES END
SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
END
PLACEMENT
    .text     INTO MY_ROM;
    .data     INTO MY_RAM;
    .stack    INTO MY_STK;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Specify at least one filename in the NAMES block.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.18 L1016: File <Filename> Found Twice in NAMES Block

Type: Error

Description: A filename is detected twice in the NAMES block.

<filename >: Name of file detected twice in the NAMES block.

Example:

```
LINK   fibo.abs
NAMES fibo.o startup.o fibo.o END
                              ^
ERROR: File fibo.o found twice in the NAMES block
SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
END
PLACEMENT
    .text     INTO MY_ROM;
    .data     INTO MY_RAM;
    .stack    INTO MY_STK;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Remove the second occurrence of the specified file.

6.3.19 L1037: ***** Linking of <parameter file> Failed ****

Type: Error

Description: An error occurred in the linking process. Linking is interrupted and no output is written. The destination absolute file and the map file are not created.

Tip: Ensure that parameter file is valid and can be located.

6.3.20 L1038: Success. Executable File Written to <absfile>

Type: Information

Description: No error occurred during the linking process. The destination absolute file and map file have been created by the linker.

Freescale Semiconductor, Inc.

6.3.21 L1039: Limited Version. Too Many Objects or Code Linked

Type: Error

Description: This message indicates that the user is running a demo version. The limits for demo versions are:

- 1024 bytes of code
- Maximum of 32 global objects linked

6.3.22 L1050: Running <versiontype>

Type: Information

Description: This message indicates that the user is running a special version of the linker, for example, a time limited version.

6.3.23 L1052: User Requested Stop

Type: Error

Description: The user has clicked the **Stop** button in the toolbar. The linker stops execution as soon as possible.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.24 L1100: Segments <Segment1 Name> and <Segment2 Name> Overlap

Type: Error

Description: Two segments defined in the PRM file overlap each other:

- <segment1 name >: Name of the first overlapping segment
- <segment2 name >: Name of the second overlapping segment

Example:

```
^
Segments MY_RAM and MY_ROM overlap
LINK   fibo.abs
NAMES fibo.o startup.o END
SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x805 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
END
PLACEMENT
    .text     INTO MY_ROM;
    .data     INTO MY_RAM;
    .stack    INTO MY_STK;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Modify the segment definition to remove the overlap.

Example:

```
LINK   fibo.abs
NAMES fibo.o startup.o END
SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
END
PLACEMENT
    .text     INTO MY_ROM;
    .data     INTO MY_RAM;
    .stack    INTO MY_STK;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.25 L1102: Out of Allocation Space in Segment <Segment Name> at Address <First Address Free>

Type: Error

Description: The specified segment is not big enough to contain all objects from sections placed in it.

- <segment name> : Name of the undersized segment
- <first address free>: First address free in this segment (for instance, address following the last address used)

Example: In the following example, assume the section `.data` contains a character variable and a structure of five bytes.

```
^
Out of allocation space in segment MY_RAM at address
0x801
LINK    fibo.abs
NAMES fibo.o startup.o END
SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x803;
    MY_ROM = READ_ONLY  0x805 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
END
PLACEMENT
    .text     INTO MY_ROM;
    .data     INTO MY_RAM;
    .stack    INTO MY_STK;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Set the end address of the specified segment to a higher value.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.26 L1103: <Section Name> not Specified in PLACEMENT Block

Type: Error

Description: Indicates that a mandatory section is not specified in the PLACEMENT block. Sections always specified in the PLACEMENT block are `.text` and `.data`.

Example:

```
^
ERROR: .text not specified in the PLACEMENT block
LINK   fibo.abs
NAMES fibo.o startup.o END
SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
END
PLACEMENT
    .init, .rodata     INTO MY_ROM;
    .data      INTO MY_RAM;
    .stack     INTO MY_STK;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Insert the missing section in the PLACEMENT block.

NOTE: *The section DEFAULT_RAM is a synonym for* `.data` *and DEFAULT_ROM is a synonym for* `.text`*. These two section names have been defined for compatibility with previous versions of the linker.*

Freescale Semiconductor, Inc.

6.3.27 L1104: Absolute Object <Object Name> Overlaps with Segment <Segment Name>

Type: Error

Description: An absolute object overlaps with a segment. This is not allowed because this may cause multiple objects to be allocated at the same address.

Example:

```
^
ERROR: Absolute object globInt overlaps with segment
MY_RAM
LINK   fibo.abs
NAMES fibo.o startup.o END
SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
END
PLACEMENT
    .text, .rodata     INTO MY_ROM;
    .data      INTO MY_RAM;
    .stack     INTO MY_STK;
END
OBJECT_ALLOCATION
      fiboCount AT 0x802;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Move the object to a free address.

Example:

```
LINK   fibo.abs
NAMES fibo.o startup.o END
SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
END
PLACEMENT
    .text      INTO MY_ROM;
    .data      INTO MY_RAM;
    .stack     INTO MY_STK;
END
OBJECT_ALLOCATION
     fiboCount AT 0xC00;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

NOTE: *An absolute object can also be placed in a segment where no sections are assigned.*

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Example:

```
LINK    fibo.abs
NAMES fibo.o startup.o END
SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
    ABS_MEM= READ_WRITE 0xC00 TO 0xC0F;
END
PLACEMENT
    .text      INTO MY_ROM;
    .data      INTO MY_RAM;
    .stack     INTO MY_STK;
END
OBJECT_ALLOCATION
      fiboCount AT 0xC00;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

6.3.28 L1105: Absolute Object <object name> Overlaps with Another Absolute Allocated Object or with a Vector

Type: Error

Description: An absolute object overlaps with another absolute object or with a vector.

Example:

```
^
ERROR: Absolute object globChar overlaps with another
absolute allocated object or with a vector
LINK    fibo.abs
NAMES fibo.o startup.o END
SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
END
PLACEMENT
    .text, .rodata     INTO MY_ROM;
    .data      INTO MY_RAM;
    .stack     INTO MY_STK;
END
OBJECT_ALLOCATION
      fiboCount AT 0xC02;
      counter   AT 0xC03;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Move the object to a free position.

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

6.3.29 L1106: <Object Name> not Found

Type: Error | Warning

Description: An object referenced in the parameter file or in the application is not found. This message is generated when:

- An object specified in a `VECTOR` or `VECTOR ADDRESS` command is not found (error).
- No startup structure is detected in the application (warning).
- An object (function or variable) referenced in another object is not found in the application (error).
- An object (function or variable) specified in the ENTRIES block is not found (error).

Example:

```
^
ERROR: globInt not found
LINK   fibo.abs
NAMES fibo.o startup.o END

SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
END
PLACEMENT
    .text, .rodata     INTO MY_ROM;
    .data      INTO MY_RAM;
    .stack     INTO MY_STK;
END

ENTRIES
      globInt;
END

/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tips:

- The missing object must be implemented in one of the modules building the application.
- Ensure that the definitions of OBJPATH and GENPATH are correct and the linker uses the latest version of object files.
- Check the NAMES block to ensure all binary files building the application are listed.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.30 L1107: <Object Name> not Found

Type: Error | Warning

Description: An object referenced in the parameter file or in the application is not found anywhere in the application. This message is generated in the following cases:

- An object moved to another section in the OBJECT_ALLOCATION block is not found anywhere in the application (warning).
- An object placed at an absolute address in the OBJECT_ALLOCATION block is not found anywhere in the application (error).
- An object specified in a VECTOR or VECTOR ADDRESS command is not found in the application (error).
- No startup structure detected in the application (warning).
- An object (function or variable) referenced in another object is not found in the application (error).
- An object (function or variable) specified in the ENTRIES block is not found in the application (error).

Example:

```
^
ERROR: globInt not found
LINK   fibo.abs
NAMES fibo.o startup.o END

SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
END
PLACEMENT
    .text, .rodata     INTO MY_ROM;
    .data     INTO MY_RAM;
    .stack    INTO MY_STK;
END

OBJECT_ALLOCATION
      globInt  AT 0xC02;
END

/* Set reset vector on _Startup */
 VECTOR ADDRESS 0xFFFE _Startup
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Tips: The missing object must be implemented in one of the modules building the application.

Ensure that path definitions are correct for OBJPATH and GENPATH and that the linker uses the last version of the object files.

Ensure that all binary files building the application are listed in the NAMES block.

6.3.31 L1109: <Segment Name> Appears Twice in SEGMENTS Block

Type: Error

Description: A segment name is specified twice in a parameter file. This is not allowed. When this segment name is referenced in the PLACEMENT block, the linker cannot detect which memory area is referenced.

Example:

```
LINK   fibo.abs
NAMES fibo.o startup.o END

SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
    MY_RAM = READ_WRITE 0xC00 TO 0xCFF;
           ^
ERROR: MY_RAM appears twice in SEGMENTS block
END
PLACEMENT
    .text, .rodata     INTO MY_ROM;
    .data      INTO MY_RAM;
    .stack     INTO MY_STK;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Change one of the segment names to generate unique segment names. If the same memory area is defined twice, remove one of the definitions.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.32 L1110: <Segment Name> Appears Twice in PLACEMENT Block

Type: Error

Description: The specified segment appears twice in a PLACEMENT block, and one of the PLACEMENT lines is part of a segment list. A segment name may appear in several lines in the PLACEMENT block, if it is the only segment specified in the segment list. Sections specified in both PLACEMENT lines are merged into one list of sections, which are allocated in the specified segment.

Example:

```
LINK   fibo.abs
NAMES fibo.o startup.o END

SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
    ROM_2  = READ_ONLY  0x500 TO 0x7FF;
END
PLACEMENT
    .text, .rodata     INTO MY_ROM;
    .data              INTO MY_RAM;
    .stack             INTO MY_STK;
    codeSec1, codeSec2 INTO ROM_2, MY_ROM;
                                        ^
ERROR: MY_ROM appears twice in PLACEMENT block
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Remove one instance of the segment from the PLACEMENT block.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.33 L1111: <Section Name> Appears Twice in PLACEMENT Block

Type: Error

Description: The specified section appears multiple times in a PLACEMENT block.

Example:

```
LINK    fibo.abs
NAMES fibo.o startup.o END

SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
    ROM_2  = READ_ONLY  0x500 TO 0x7FF;
END
PLACEMENT
    .text, .rodata      INTO MY_ROM;
    .data               INTO MY_RAM;
    .stack              INTO MY_STK;
    .text               INTO ROM_2;
         ^
ERROR: .text appears twice in PLACEMENT block
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Remove one occurrence of the specified section from the PLACEMENT block.

6.3.34 L1112: <Section name> Section Has Segment Type <Segment Qualifier> (Illegal)

Type: Error

Description: A section is placed in a segment defined with an incompatible qualifier. This message is generated when:

- The section `.stack` is placed in a READ_ONLY segment.
- The section `.bss` is placed in a READ_ONLY segment.
- The section `.startData` is placed in a READ_WRITE, NO_INIT, or PAGED segment.
- The section `.init` is placed in a READ_WRITE, NO_INIT, or PAGED segment.
- The section `.copy` is placed in a READ_WRITE, NO_INIT, or PAGED segment.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

- The section `.text` is placed in a READ_WRITE, NO_INIT, or PAGED segment.
- The section `.data` is placed in a READ_ONLY segment.
- A data section is placed in a READ_ONLY segment.
- A code section is placed in a READ_WRITE segment.

Example:

```
 ^
 ERROR: The .data section has segment type READ_ONLY
(illegal)
 LINK   fibo.abs
 NAMES fibo.o startup.o END

 SEGMENTS
     MY_RAM = READ_WRITE 0x800 TO 0x80F;
     MY_ROM = READ_ONLY  0x810 TO 0xAFF;
     MY_STK = READ_WRITE 0xB00 TO 0xBFF;
     ROM_2  = READ_ONLY  0x500 TO 0x7FF;
 END
 PLACEMENT
     .text, .rodata     INTO MY_ROM;
     .data              INTO ROM_2;
     .stack             INTO MY_STK;
 END
 /* Set reset vector on _Startup */
 VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Place the specified section in a segment that has been defined with an appropriate qualifier.

Example:

```
LINK   fibo.abs
NAMES fibo.o startup.o END

SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
    ROM_2  = READ_ONLY  0x500 TO 0x7FF;
END
PLACEMENT
    .text      INTO MY_ROM;
    .data      INTO MY_RAM;
    .stack     INTO MY_STK;
END

/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```


For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

6.3.35 L1113: <Section name> Section Has Segment Type <Segment Qualifier> (Illegal)

Type: Warning

Description: A section is placed in a segment, which has been defined with an incompatible qualifier. This message is generated in the following cases:

- The section `.stack` is placed in a READ_ONLY segment.
- The section `.bss` is placed in a READ_ONLY segment.
- The section `.startData` is placed in a READ_WRITE, NO_INIT, or PAGED segment.
- The section `.init` is placed in a READ_WRITE, NO_INIT, or PAGED segment.
- The section `.copy` is placed in a READ_WRITE, NO_INIT, or PAGED segment.
- The section `.text` is placed in a READ_WRITE, NO_INIT, or PAGED segment.
- The section `.data` is placed in a READ_ONLY segment.

Example:

```
 ^
 ERROR: The .data section has segment type READ_ONLY
(illegal)
 LINK   fibo.abs
 NAMES fibo.o startup.o END

 SEGMENTS
     MY_RAM = READ_WRITE 0x800 TO 0x80F;
     MY_ROM = READ_ONLY  0x810 TO 0xAFF;
     MY_STK = READ_WRITE 0xB00 TO 0xBFF;
     ROM_2  = READ_ONLY  0x500 TO 0x7FF;
 END
 PLACEMENT
     .text, .rodata     INTO MY_ROM;
     .data              INTO ROM_2;
     .stack             INTO MY_STK;
 END

/* Set reset vector on _Startup */
 VECTOR ADDRESS 0xFFFE _Startup
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Tip: Place the specified section in a segment that has been defined with an appropriate qualifier.

Example:

```
LINK   fibo.abs
NAMES fibo.o startup.o END

SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
    ROM_2  = READ_ONLY  0x500 TO 0x7FF;
END
PLACEMENT
    .text     INTO MY_ROM;
    .data     INTO MY_RAM;
    .stack    INTO MY_STK;
END

/* Set reset vector on _Startup */
 VECTOR ADDRESS 0xFFFE _Startup
```

6.3.36 L1114: <Section Name> Section Has Segment Type <Segment Qualifier> (Initialization Problem)

Type: Warning

Description: The specified section is loaded in a segment that has been defined with the qualifier NO_INIT or PAGED. This may generate a problem because the section contains some initialized constants, which will not be initialized at application startup. This message is generated when:

- The section `.rodata` is placed in a NO_INIT or PAGED segment.
- The section `.rodata1` is placed in a NO_INIT or PAGED segment.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Example:

```
  ^
WARNING: The .rodata section has segment type NO_INIT
(initialization problem)
LINK   fibo.abs
NAMES fibo.o startup.o END

SEGMENTS
     MY_RAM = READ_WRITE 0x800 TO 0x80F;
     MY_ROM = READ_ONLY  0x810 TO 0xAFF;
     MY_STK = READ_WRITE 0xB00 TO 0xBFF;
     RAM_2  = NO_INIT    0x500 TO 0x7FF;
END
PLACEMENT
     .text   INTO MY_ROM;
     .data   INTO MY_RAM;
     .stack  INTO MY_STK;
     .rodata INTO RAM_2;
END

/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Place the specified section in a segment defined with the READ_ONLY or READ_WRITE qualifier.

Example:

```
LINK   fibo.abs
NAMES fibo.o startup.o END

SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
    RAM_2  = NO_INIT    0x500 TO 0x7FF;
END
PLACEMENT
    .text     INTO MY_ROM;
    .data     INTO MY_RAM;
    .stack    INTO MY_STK;
    .rodata   INTO MY_ROM;
END

/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Freescale Semiconductor, Inc.

6.3.37 L1115: Function <Function Name> not Found

Type: Error | Warning

Description: The specified function is not found in the application. This message is generated when:

- No main function is available in the application. This function is not required for an assembly application. For ANSI C applications, if no main function is available, the programmer must ensure that application startup is performed correctly. Usually, the main function is called `main`, but a personal main function can be defined using the linker command `MAIN`.
- No init function is available. The init function defines the entry point in the application. This function is required for ANSI C and assembly applications. Usually, the init function is called `_Startup`, but a personal init function can be defined using the linker command `INIT`.

Tip: Provide application with the requested function.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.38 L1118: Vector Allocated at Absolute Address <Address> Overlaps with Another Vector or an Absolute Allocated Object

Type: Error

Description: A vector overlaps with an absolute object or another vector.

Example:

```
^
ERROR: Vector allocated at absolute address 0xFFFE
overlaps with another vector or an absolute allocated
object
LINK   fibo.abs
NAMES fibo.o startup.o END

SEGMENTS
     MY_RAM = READ_WRITE 0x800 TO 0x80F;
     MY_ROM = READ_ONLY  0x810 TO 0xAFF;
     MY_STK = READ_WRITE 0xB00 TO 0xBFF;
END
PLACEMENT
     .text, .rodata     INTO MY_ROM;
     .data      INTO MY_RAM;
     .stack     INTO MY_STK;
END

/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
VECTOR ADDRESS 0xFFFF 0x000A
```

Tip: Move the object or vector to a free position.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.39 L1119: Vector Allocated at Absolute Address <Address> Overlaps with Sections Placed in Segment <Segment Name>

Type: Error

Description: The specified vector is allocated inside a segment specified in the PLACEMENT block. This is not allowed because the vector may overlap with objects defined in the sections.

A vector may be allocated inside a segment that does not appear in the PLACEMENT block.

Example:

```
ERROR: Vector allocated at absolute address 0xFFFE
overlaps with sections placed in segment ROM_2
LINK   fibo.abs
NAMES fibo.o startup.o END

SEGMENTS
     MY_RAM = READ_WRITE 0x800  TO 0x80F;
     MY_ROM = READ_ONLY  0x810  TO 0xAFF;
     MY_STK = READ_WRITE 0xB00  TO 0xBFF;
     ROM_2  = READ_ONLY  0xFF00 TO 0xFFFF;
END
PLACEMENT
     .text   INTO MY_ROM;
     .data   INTO MY_RAM;
     .stack  INTO MY_STK;
     .rodata INTO ROM_2;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Define the specified segment outside the vector table.

Example:

```
LINK   fibo.abs
NAMES fibo.o startup.o END

SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
    ROM_2  = READ_ONLY  0xC00 TO 0xCFF;
END
PLACEMENT
    .text   INTO MY_ROM;
    .data   INTO MY_RAM;
    .stack  INTO MY_STK;
    .rodata INTO ROM_2;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```


For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

6.3.40 L1120: Vector Allocated at Absolute Address <Address> Placed in Segment <Segment Name>, Which Has No READ_ONLY Qualifier

Type: Error

Description: The specified vector is defined inside a segment not defined with the qualifier READ_ONLY. The vector table should be initialized at application load time during the debug phase. It should be burned into the EPROM when application development is terminated. For this reason, the vector table must always be located in a READ_ONLY memory area.

Example:

```
^
ERROR: Vector allocated at absolute address 0xFFFE
placed in segment RAM_2 which has not READ_ONLY qual-
ifier
LINK   fibo.abs
NAMES fibo.o startup.o END

SEGMENTS
     MY_RAM = READ_WRITE 0x800  TO 0x80F;
     MY_ROM = READ_ONLY  0x810  TO 0xAFF;
     MY_STK = READ_WRITE 0xB00  TO 0xBFF;
     RAM_2  = READ_WRITE 0xFF00 TO 0xFFFF;
END
PLACEMENT
     .text   INTO MY_ROM;
     .data   INTO MY_RAM;
     .stack  INTO MY_STK;
END

/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Define the specified segment with the READ_ONLY qualifier.

6.3.41 L1121: Out of Allocation Space at Address <Address> for .copy Section

Type: Error

Description: Insufficient memory to store information for initialized variables in the `.copy` section

Tip: Specify a higher end address for the segment, where the `.copy` section is allocated.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.42 L1122: Section .copy Must Be Last Section in Section List

Type: Error

Description: The `.copy` section is not specified at the end of a section list in the PLACEMENT block. Since the size of this section cannot be evaluated before all initialization values are written, the `.copy` section must be the last section in a section list.

Example:

```
 ^
 ERROR: Section .copy must be the last section in the
section list
 LINK   fibo.abs
 NAMES fibo.o startup.o END

 SEGMENTS
     MY_RAM = READ_WRITE 0x800  TO 0x80F;
     MY_ROM = READ_ONLY  0x810  TO 0xAFF;
     MY_STK = READ_WRITE 0xB00  TO 0xBFF;
 END
 PLACEMENT
     .copy, .text   INTO MY_ROM;
     .data          INTO MY_RAM;
     .stack         INTO MY_STK;
 END

 /* Set reset vector on _Startup */
 VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Move the section `.copy` to the last position in the section list or define it on a separate PLACEMENT line in a separate segment.

Example:

```
LINK   fibo.abs
NAMES fibo.o startup.o END

SEGMENTS
    MY_RAM = READ_WRITE 0x800 TO 0x80F;
    MY_ROM = READ_ONLY  0x810 TO 0xAFF;
    MY_STK = READ_WRITE 0xB00 TO 0xBFF;
    ROM_2  = READ_ONLY  0xC00 TO 0xDFF;
END
PLACEMENT
    .text   INTO MY_ROM;
    .data   INTO MY_RAM;
    .stack  INTO MY_STK;
    .copy   INTO ROM_2;
END

/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```


For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

6.3.43 L1123: Invalid Range Defined for Segment <Segment Name> — End Address Must Be Bigger Than Start Address

Type: Error

Description: The memory range specified in the segment definition is not valid. The segment end address is smaller than the segment start address.

Example:

```
LINK   fibo.abs
NAMES fibo.o startup.o END

SEGMENTS
     MY_RAM = READ_WRITE 0x800  TO 0x7FF;
                                        ^
ERROR: Invalid range defined for segment MY_RAM. End
address must be bigger than start address
     MY_ROM = READ_ONLY  0x810  TO 0xAFF;
     MY_STK = READ_WRITE 0xB00  TO 0xBFF;
END
PLACEMENT
    .text             INTO MY_ROM;
    .data            INTO MY_RAM;
    .stack           INTO MY_STK;
END

/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Change the segment start or end address to define a valid memory range.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.44 L1124: '+' or '-' Should Directly Follow Filename

Type: Error

Description: The + or - suffix specified after a filename in the NAMES block does not directly follow the filename. A space probably exists between the filename and suffix.

Example:

```
LINK    fibo.abs
NAMES fibo.o + startup.o END
              ^
ERROR: '+' or '-' should directly follow the filename
SEGMENTS
    MY_RAM = READ_WRITE 0x800   TO 0x80F;
    MY_ROM = READ_ONLY  0x810   TO 0xAFF;
    MY_STK = READ_WRITE 0xB00   TO 0xBFF;
END
PLACEMENT
    .text          INTO MY_ROM;
    .data          INTO MY_RAM;
    .stack         INTO MY_STK;
END

/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Remove the extra space after the filename.

Example:

```
LINK    fibo.abs
NAMES fibo.o+ startup.o END
SEGMENTS
    MY_RAM = READ_WRITE 0x800   TO 0x80F;
    MY_ROM = READ_ONLY  0x810   TO 0xAFF;
    MY_STK = READ_WRITE 0xB00   TO 0xBFF;
END
PLACEMENT
    .text          INTO MY_ROM;
    .data          INTO MY_RAM;
    .stack         INTO MY_STK;
END

/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```


For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

6.3.45 L1125: In Small Memory Model, Code and Data Must Be Located on Bank 0

Type: Error

Description: The application has been assembled or compiled in a small memory model and the memory area specified for a segment is not located on the first 64 Kbytes (0x0000 to 0xFFFF).

Example:

```
ERROR: In small memory model, code and data must be
located on bank 0
LINK   fibo.abs
NAMES fibo.o startup.o END
SEGMENTS
     MY_RAM = READ_WRITE 0x800   TO 0x80F;
     MY_ROM = READ_ONLY  0x10810  TO 0x10AFF;
     MY_STK = READ_WRITE 0xB00   TO 0xBFF;
END
PLACEMENT
     .text          INTO MY_ROM;
     .data          INTO MY_RAM;
     .stack         INTO MY_STK;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: If memory higher than 0xFFFF is required for the application, the application must be assembled or compiled using the banked memory model. If no memory above 0xFFFF is required, modify the memory range and place it on the first 64 Kbytes of memory.

Freescale Semiconductor, Inc.

6.3.46 L1127: Object Allocated Outside of Segment Bounds (HC12)

Type: Warning

Description: The application has been assembled or compiled in the small memory model and the memory area specified for a segment is not located on the first 64 Kbytes (0x0000 to 0xFFFF).

Example:

```
 ^
ERROR: In small memory model, code and data must be
located on bank 0
 LINK   fibo.abs
 NAMES fibo.o startup.o END
 SEGMENTS
     MY_RAM = READ_WRITE 0x800   TO 0x80F;
     MY_ROM = READ_ONLY  0x10810  TO 0x10AFF;
     MY_STK = READ_WRITE 0xB00   TO 0xBFF;
 END
 PLACEMENT
     .text          INTO MY_ROM;
     .data          INTO MY_RAM;
     .stack         INTO MY_STK;
 END

/* Set reset vector on _Startup */
 VECTOR ADDRESS 0xFFFE _Startup
```

Tip: If memory above 0xFFFF is required for the application, the application must be assembled or compiled using the medium memory model. If not, modify the memory range and place it on the first 64 Kbytes of memory.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.47 L1200: Both STACKTOP and STACKSIZE Defined

Type: Error

Description: The `STACKTOP` and `STACKSIZE` commands are specified in the *.prm* file. This is not allowed because it generates ambiguity in defining the stack.

Example:

```
^
ERROR: Both STACKTOP and STACKSIZE defined
LINK   fibo.abs
NAMES fibo.o startup.o END

STACKTOP 0xBFE
SEGMENTS
    MY_RAM = READ_WRITE 0x800  TO 0x80F;
    MY_ROM = READ_ONLY  0x810  TO 0xAFF;
END
PLACEMENT
    .text          INTO MY_ROM;
    .data          INTO MY_RAM;
END
STACKSIZE 0x60
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Remove either the `STACKTOP` or `STACKSIZE` command from the *.prm* file.

Freescale Semiconductor, Inc.

6.3.48 L1201: No Stack Defined

Type: Warning

Description: The parameter file does not contain a stack definition. In that case, it is the programmer's responsibility to initialize the stack pointer inside the application code. The stack can be defined in the parameter file in one of three ways:

1. Through the `STACKTOP` command
2. Through the `STACKSIZE` command
3. Through specification of the `.stack` section in the placement block

Example:

```
 ^
WARNING: No stack defined
 LINK   fibo.abs
 NAMES fibo.o startup.o END

 SEGMENTS
     MY_RAM = READ_WRITE 0x800  TO 0x80F;
     MY_ROM = READ_ONLY  0x810  TO 0xAFF;
 END
 PLACEMENT
     .text          INTO MY_ROM;
     .data          INTO MY_RAM;
 END
 /* Set reset vector on _Startup */
 VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Define the stack in one of the three ways specified.

NOTE: *If the programmer initializes the stack pointer inside the source code, initialization from the linker will be overridden.*

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.49 L1202: Stack Cannot Be Allocated on More Than One Segment

Type: Error

Description: The section `.stack` is specified on a PLACEMENT line where several segments are listed. This is not allowed because the memory area reserved for the stack must be contiguous and cannot be split over different memory ranges.

Example:

```
  ^
ERROR: stack cannot be allocated on more than one seg-
ment
 LINK   fibo.abs
 NAMES fibo.o startup.o END
 SEGMENTS
     MY_RAM = READ_WRITE 0x800  TO 0x80F;
     MY_ROM = READ_ONLY  0x810  TO 0xAFF;
     STK_1 = READ_WRITE 0xB00  TO 0xBFF;
     STK_2 = READ_WRITE 0xD00  TO 0xDFF;
 END
 PLACEMENT
     .text          INTO MY_ROM;
     .data          INTO MY_RAM;
     .stack         INTO STK_1, STK_2;
 END

 /* Set reset vector on _Startup */
 VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Define a single segment with the READ_WRITE or NO_INIT qualifier to allocate the stack.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.50 L1203: STACKSIZE Command Defines a Size of <Size> But .stack Specifies a Stacksize of <Size>

Type: Error

Description: The stack is defined through both a `STACKSIZE` command and placement of the `.stack` section in a READ_WRITE or NO_INIT segment. However, the size specified in the `STACKSIZE` command is bigger than the size of the segment where the stack is allocated.

Example:

```
ERROR: STACKSIZE command defines a size of 0x120 but
.stack specifies a stacksize of 0x100
LINK   fibo.abs
NAMES fibo.o startup.o END
SEGMENTS
     MY_RAM = READ_WRITE 0x800  TO 0x80F;
     MY_ROM = READ_ONLY  0x810  TO 0xAFF;
     STK_1  = READ_WRITE 0xB00  TO 0xBFF;
END
PLACEMENT
     .text          INTO MY_ROM;
     .data          INTO MY_RAM;
     .stack         INTO STK_1;
END

STACKSIZE 0x120

/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip:

- To avoid this message, either adapt the size specified in the `STACKSIZE` command to fit into the segment where `.stack` is allocated or simply remove the command `STACKSIZE`.
- If the command `STACKSIZE` is removed from the previous example, the linker will initialize a stack from 0x100 bytes. The stack pointer initial value will be set to 0xBFE.

Freescale Semiconductor, Inc.

Example:

```
LINK   fibo.abs
NAMES fibo.o startup.o END
SEGMENTS
    MY_RAM = READ_WRITE 0x800  TO 0x80F;
    MY_ROM = READ_ONLY  0x810  TO 0xAFF;
    MY_STK = READ_WRITE 0xB00  TO 0xBFF;
END
PLACEMENT
    .text          INTO MY_ROM;
    .data          INTO MY_RAM;
    .stack         INTO MY_STK;
END

/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

If the size specified in a STACKSIZE command is smaller than the size of the segment where the section .stack is allocated, the stack pointer initial value will be evaluated as follows:

```
&lt;segment start address&gt; + <size in STACKSIZE> -
<Additional Byte Required by the processor>
```

```
LINK   fibo.abs
NAMES fibo.o startup.o END
SEGMENTS
    MY_RAM = READ_WRITE 0x800  TO 0x80F;
    MY_ROM = READ_ONLY  0x810  TO 0xAFF;
    MY_STK = READ_WRITE 0xB00  TO 0xBFF;
END
PLACEMENT
    .text          INTO MY_ROM;
    .data          INTO MY_RAM;
    .stack         INTO MY_STK;
END
STACKSIZE 0x60
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

In the previous example, the initial value for the stack pointer is evaluated as:

```
0xB00 + 0x60s -2 = 0xB5E
```


For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.

6.3.51 L1204: STACKTOP Command Defines Initial Value of <Stack Top> But .stack Specifies Initial Value of <Initial Value>

Type: Error

Description: The stack is defined through both a `STACKTOP` command and placement of the `.stack` section in a READ_WRITE or NO_INIT segment. However, the value specified in the `STACKTOP` command is bigger than the end address of the segment where the stack is allocated.

Example:

```
 ^
 ERROR: STACKTOP command defines an initial value of
0xCFE but .stack specifies an initial value of 0xBFF
 LINK   fibo.abs
 NAMES fibo.o startup.o END
 SEGMENTS
     MY_RAM = READ_WRITE 0x800  TO 0x80F;
     MY_ROM = READ_ONLY  0x810  TO 0xAFF;
     STK_1  = READ_WRITE 0xB00  TO 0xBFF;
 END
 PLACEMENT
     .text          INTO MY_ROM;
     .data          INTO MY_RAM;
     .stack         INTO STK_1;
 END

 STACKTOP 0xCFE
 /* Set reset vector on _Startup */
 VECTOR ADDRESS 0xFFFE _Startup
```

Tips:

- To avoid this message, either adapt the address specified in the `STACKTOP` command to fit into the segment where `.stack` is allocated or simply remove the command `STACKTOP`.
- If the command `STACKTOP` is removed from the previous example, the stack pointer initial value will be set to 0xBFE.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Example:

```
LINK    fibo.abs
NAMES fibo.o startup.o END
SEGMENTS
    MY_RAM = READ_WRITE 0x800  TO 0x80F;
    MY_ROM = READ_ONLY  0x810  TO 0xAFF;
    MY_STK = READ_WRITE 0xB00  TO 0xBFF;
END
PLACEMENT
    .text          INTO MY_ROM;
    .data          INTO MY_RAM;
    .stack         INTO MY_STK;
END

/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

6.3.52 L1205: STACKTOP Command Incompatible with .stack Being Part of List of Sections

Type: Error

Description: The stack is defined through both a `STACKTOP` command and placement of the `.stack` section in a READ_WRITE or NO_INIT segment. The `.stack` section is specified in a list of sections in the PLACEMENT block.

Example:

```
^
ERROR: STACKTOP command incompatible with .stack be-
ing part of a list of sections
LINK    fibo.abs
NAMES fibo.o startup.o END
SEGMENTS
     MY_RAM = READ_WRITE 0x800  TO 0x80F;
     MY_ROM = READ_ONLY  0x810  TO 0xAFF;
     STK_1  = READ_WRITE 0xB00  TO 0xBFF;
END
PLACEMENT
     .text          INTO MY_ROM;
     .data, .stack  INTO STK_1;
END

STACKTOP 0xBFE
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Specify the `.stack` section in a placement line, where the stack alone is specified.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

6.3.53 L1206: Stack Overlaps with a Segment Which Appears in PLACEMENT Block

Type: Error

Description: The stack is defined through the command STACKTOP and the initial value is inside a segment, which is used in the PLACEMENT block. This is not allowed because the stack may overlap with allocated objects.

Example:

```
ERROR: .stack overlaps with a segment which appears
in the PLACEMENT block
LINK   fibo.abs
NAMES fibo.o startup.o END
SEGMENTS
     MY_RAM = READ_WRITE 0x800  TO 0x80F;
     MY_ROM = READ_ONLY  0x810  TO 0xAFF;
     STK_1  = READ_WRITE 0xB00  TO 0xBFF;
END
PLACEMENT
     .text          INTO MY_ROM;
     .data          INTO STK_1;
END

STACKTOP 0xBFE
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Define the stack initial value outside all segments specified in the PLACEMENT block.

Example:

```
LINK   fibo.abs
 NAMES fibo.o startup.o END
 SEGMENTS
     MY_RAM = READ_WRITE 0x800  TO 0x80F;
     MY_ROM = READ_ONLY  0x810  TO 0xAFF;
     MY_STK = READ_WRITE 0xB00  TO 0xBFF;
 END
 PLACEMENT
     .text          INTO MY_ROM;
     .data          INTO MY_RAM;
 END
 STACKTOP 0xBFE
/* Set reset vector on _Startup */
 VECTOR ADDRESS 0xFFFE _Startup
```

Freescale Semiconductor, Inc.

6.3.54 L1207: STACKSIZE Command is Missing

Type: Error

Description: The stack is defined by placing the `.stack` section in a READ_WRITE or NO_INIT segment, although the `.stack` section is not alone in the section list. In this case, a `STACKSIZE` command is required to specify the stack size.

Example:

```
^
ERROR: STACKSIZE command is missing
LINK   fibo.abs
NAMES fibo.o startup.o END
SEGMENTS
    MY_RAM = READ_WRITE 0x800  TO 0x80F;
    MY_ROM = READ_ONLY  0x810  TO 0xAFF;
    STK_1  = READ_WRITE 0xB00  TO 0xBFF;
END
PLACEMENT
    .text           INTO MY_ROM;
    .data, .stack   INTO STK_1;
END

/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Specify the stack size in a `STACKSIZE` command.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.55 L1301: Cannot Open File <Filename>

Type: Error

Description: The linker is unable to open the application map file, absolute file, or one of the binary files used to build the application.

Tips:

- If the *.abs* file or *.map* file cannot be found, ensure that memory is available for the directory to store the file and the directory has read/write access.
- If the environment variable TEXTPATH is defined, the map file is stored in the first directory specified; otherwise, it is created in the directory where the source file is detected.
- If the environment variable ABSPATH is defined, the absolute file is stored in the first directory specified; otherwise, it is created in the directory where the parameter file is detected.
- If a binary file cannot be found, make sure the file exists and is spelled correctly. Check if paths are defined correctly. The binary files must be located in one of the paths listed in the environment variables OBJPATH or GENPATH or in the working directory.

6.3.56 L1302: File <Filename> not Found

Type: Error

Description: A file required during the link session cannot be found. This message is generated when the parameter file specified on the command line cannot be found.

Tips:

- Make sure the file exists and is spelled correctly.
- Check if paths are defined correctly. The parameter file must be located in one of the paths listed in the environment variable GENPATH or in the project directory.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.57 L1303: <Filename> is not a Valid ELF File

Type: Error

Description: The specified file is not a valid ELF binary file. The linker is only able to link ELF binary files.

Tips:

- Check that the specified file has been compiled or assembled with the correct option to generate an ELF binary file.
- Check if paths are defined correctly. The binary files must be located in one of the paths listed in the environment variables OBJPATH or GENPATH or in the project directory.

6.3.58 L1304: <Filename> is not a Valid Hex File

Type: Error

Description: The file specified in a `HEXFILE` command is not a valid hex file.

Tips: Ensure that the file was generated correctly.

Ensure that paths are defined correctly. The hex files must be located in one of the paths listed in the environment variable OBJPATH or GENPATH or in the working directory.

6.3.59 L1305: <Filename> is not an ELF Format Object File (ELF Object File Expected)

Type: Error

Description: The specified file is not an ELF binary file. The linker is only able to link ELF binary files.

Tips:

- Ensure that the specified file has been compiled or assembled with the correct option to generate an ELF binary file.
- Ensure that paths are defined correctly. The binary files must be located in one of the paths listed in the environment variables OBJPATH or GENPATH or in the working directory.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.60 L1309: Cannot Open <File>

Type: Error

Description: An input file is missing or the linker cannot open it.

Tip: Ensure that paths are defined correctly.

6.3.61 L1400: Incompatible Processor: <Processor Name> in Previous Files and <Processor Name> in Current File

Type: Error

Description: The binary files building the application have been generated for a different target processor. In this case, the linked code cannot be compatible.

Tips:

- Make sure all sources are compiled and assembled for the same processor.
- Check if paths are defined correctly. The binary files must be located in one of the paths listed in the environment variables OBJPATH or GENPATH or in the project directory.

6.3.62 L1401: Incompatible Memory Model: <Memory Model Name> in Previous Files and <Memory Model Name> in Current File

Type: Error

Description: The binary files building the application have been generated for a different memory model. In this case, the linked code cannot be compatible.

Tips:

- Make sure all sources are compiled and assembled in the same memory model.
- Check if paths are defined correctly. The binary files must be located in one of the paths listed in the environment variables OBJPATH or GENPATH or in the project directory.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.63 L1403: Unknown Processor <Processor Constant>

Type: Error

Description: The processor encoded in the binary object file is not a valid processor constant.

Tip: Check if paths are defined correctly. The binary files must be located in one of the paths listed in the environment variables OBJPATH or GENPATH or in the project directory.

6.3.64 L1404: Unknown Memory Model <Memory Model Constant>

Type: Error

Description: The memory model encoded in the binary object file is not valid for the target processor.

Tip: Check if paths are defined correctly. The binary files must be located in one of the paths listed in the environment variables OBJPATH or GENPATH or in the project directory.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.65 L1501: <Symbol Name> Cannot be Moved in Section <Section Name> (Invalid Qualifier <Segment Qualifier>)

Type: Error

Description: An invalid move operation has been detected from an object inside a section, which appears only in the parameter file. In that case, the first object moved in a section determines the attribute associated with the section.

- If the object is a function, the section should be a code section.
- If the object is a constant, the section should be a constant section.
- Otherwise, it should be a data section.

This message is generated:

- When a variable is moved in a section, which is placed in a READ_ONLY segment
- When a function is moved in a section, which is placed in a READ_WRITE, NO_INIT, or PAGED segment

Example:

```
ERROR: counter cannot be moved in section sec2 (in-
valid qualifier READ_ONLY)
 LINK   fibo.abs
 NAMES fibo.o startup.o END
 SEGMENTS
     MY_RAM = READ_WRITE 0x800  TO 0x80F;
     MY_ROM = READ_ONLY  0x810  TO 0xAFF;
     STK_1  = READ_WRITE 0xB00  TO 0xBFF;
 END
 PLACEMENT
     .text, sec2    INTO MY_ROM;
     .data          INTO MY_RAM;
     .stack         INTO STK_1;
 END

 OBJECT_ALLOCATION
   counter IN sec2;
 END
/* Set reset vector on _Startup */
 VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Move the section in a segment with the required qualifier or remove the move operation.

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

6.3.66 L1502: <Object Name> Cannot be Moved from Section <Source Section Name> to Section <Destination Section Name>

Type: Error

Description: An invalid move operation has been detected from an object inside a section, which also appears in a binary file.

This message is generated when a variable is moved in a code or constant section or a function is moved in a data or constant section.

Example:

```
 ^
 ERROR: counter cannot be moved from section .data to
section .text
 LINK   fibo.abs
 NAMES fibo.o startup.o END
 SEGMENTS
     MY_RAM = READ_WRITE 0x800  TO 0x80F;
     MY_ROM = READ_ONLY  0x810  TO 0xAFF;
     STK_1  = READ_WRITE 0xB00  TO 0xBFF;
 END
 PLACEMENT
     .text          INTO MY_ROM;
     .data          INTO MY_RAM;
     .stack         INTO STK_1;
 END

 OBJECT_ALLOCATION
   counter IN .text;
 END
/* Set reset vector on _Startup */
 VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Move the object into a section with the required attribute or remove the move operation.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.67 L1503: <Object Name> (from file <Filename>) Cannot be Moved from Section <Source Section Name> to Section <Destination Section Name>

Type: Error

Description: An invalid move operation has been detected for objects defined in a binary file inside a section.

This message is generated when a variable is moved in a code or constant section or a function is moved in a data or constant section.

Example:

```
  ^
ERROR: counter (from file fibo.o) cannot be moved
from section .data to section .text
 LINK   fibo.abs
 NAMES fibo.o startup.o END
 SEGMENTS
     MY_RAM = READ_WRITE 0x800  TO 0x80F;
     MY_ROM = READ_ONLY  0x810  TO 0xAFF;
     STK_1  = READ_WRITE 0xB00  TO 0xBFF;
 END
 PLACEMENT
     .text          INTO MY_ROM;
     .data          INTO MY_RAM;
      .stack        INTO STK_1;
 END

 OBJECT_ALLOCATION
   fibo.o:[DATA] IN .text;
 END
/* Set reset vector on _Startup */
 VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Move the specified object into a section with the required attribute or remove the move operation.

Freescale Semiconductor, Inc.

6.3.68 L1504: <Object Name> (from section <Section Name>) Cannot be Moved from Section <Source Section Name> to Section <Destination Section Name>

Type: Error

Description: An invalid move operation has been detected for objects defined in a section inside of another section.

This message is generated when a variable is moved in a code or constant section or a function is moved in a data or constant section.

Example:

```
^
ERROR: counter (from section .data) cannot be moved
from section .data to section .text
LINK   fibo.abs
NAMES fibo.o startup.o END
SEGMENTS
    MY_RAM = READ_WRITE 0x800  TO 0x80F;
    MY_ROM = READ_ONLY  0x810  TO 0xAFF;
    STK_1  = READ_WRITE 0xB00  TO 0xBFF;
END
PLACEMENT
    .text          INTO MY_ROM;
    .data          INTO MY_RAM;
    .stack         INTO STK_1;
END
OBJECT_ALLOCATION
  .data>[*] IN .text;
END
/* Set reset vector on _Startup */
VECTOR ADDRESS 0xFFFE _Startup
```

Tip: Move the specified object into a section with the required attribute or remove the move operation.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.69 L1600: Main Function Detected in ROM Library

Type: Warning

Description: A main function has been detected in a ROM library. A main function is not required in a ROM library since they are not self-executable applications.

Tips:

- Remove the `MAIN` command from the parameter file.
- If the application contains the function `main`, rename it.

6.3.70 L1601: Startup Function Detected in ROM Library

Type: Warning

Description: An application entry point has been detected in a ROM library. An application entry point is not required in a ROM library.

Tips:

- Remove the `INIT` command from the parameter file.
- If the application contains a `_Startup` function, rename it.

6.3.71 L1620: Bad Digit in Binary Number

Type: Error

Description: Syntax error — Illegal character in a binary number

6.3.72 L1621: Bad Digit in Octal Number

Type: Error

Description: Syntax error — Illegal character in an octal number

6.3.73 L1622: Bad Digit in Decimal Number

Type: Error

Description: Syntax error — Illegal character in a decimal number

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

6.3.74 L1623: Number too Big

Type: Error

Description: Syntax error — An identifier in the linker parameter file is limited to a length of 31 characters.

Tip: Reduce the length of the identifier.

6.3.75 L1624: Ident too Long. Cut after 31 Characters

Type: Error

Description: Syntax error — An identifier in the linker parameter file is limited to a length of 31 characters. The identifier string is truncated after 31 characters.

Tip: Reduce the length of the identifier.

6.3.76 L1625: Comment not Closed

Type: Error

Description: An ANSI C comment (/* .... */) was opened, but not closed.

Tip: Close the comment.

6.3.77 L1626: Unexpected End of File

Type: Error

Description The end of file was encountered and the scanner was involved in the inner scope of an expression or structure nesting. This is illegal.

Tip: Check the syntax of the linker parameter file.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.78 L1627: PRESTART Command not Supported Yet

Type: Error

Description: The PRESTART command is recognized by the parser, but is not yet implemented.

Tip: Contact a Motorola representative for features of the next release.

6.3.79 L1628: HEXFILE Command not Supported Yet

Type: Error

Description: The HEXFILE command is recognized by the parser, but is not implemented yet.

Tip: Contact a Motorola representative for features of the next release.

6.3.80 L1629: START_DATA Command not Supported Yet

Type: Error

Description: The START_DATA command is recognized by the parser, but is not implemented yet.

Tip: Contact a Motorola representative for features of the next release.

6.3.81 L1700: File <Filename> Should Contain DWARF Information

Type: Error

Description: The binary file that defines the startup structure does not contain DWARF information. This is required because the type of startup structure is not fixed by the linker and depends on the field and field position inside the user-defined structure.

Tip: Insert DWARF information and recompile the ANSI C file containing the startup structure definition.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.82 L1701: Startup Data Structure is Empty

Type: Error

Description: The size of the user-defined startup structure is 0 bytes.

Tip: Check if a startup structure is needed. If a startup structure is available, ensure that the correct field name is listed.

6.3.83 L1800: Read Error in <File>

Type: Error

Description: An error occurred while reading one of the ELF input object files. The object file is corrupt.

Tip: Recompile sources. Contact a Motorola representative, if the error appears again.

6.3.84 L1803: Out of Memory in <Function Name>

Type: Error

Description: Insufficient memory to allocate the internal structure required by the linker

6.3.85 L1804: No ELF Section Header Table Found in <Filename>

Type: Error

Description: Section header table not detected in the binary file

Tips:

- Ensure that the correct binary file is used.
- Check if paths are defined correctly. The binary files must be located in one of the paths listed in the environment variables OBJPATH or GENPATH or in the project directory.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.86 L1806: ELF File <Filename> Appears to be Corrupted

Type: Error

Description: The specified binary file is not a valid ELF binary file.

Tips:

- Ensure that the correct binary file is used.
- Check if paths are defined correctly. The binary files must be located in one of the paths listed in the environment variables OBJPATH or GENPATH or in the project directory.

6.3.87 L1808: String Overflow in <Function Name>, Contact Vendor

Type: Error

Description: A section name detected in a section table is longer than 100 characters.

Tip: Ensure all section names are smaller than 100 characters.

6.3.88 L1809: Section <Section Name> Located in a Segment with Invalid Qualifier

Type: Error

Description: Attributes associated with a section and used in several binary files are not compatible. In one file, the section contains variables; in the other, it contains constants, variables, or code.

Tip: Check usage of the different sections in all binary files. A specific section should contain the same type of information throughout the project.

6.3.89 L1811: Symbol <Symbol Number> - < Symbol Name> Duplicated in <First Filename> and <Second Filename>

Type: Error

Description: The specified global symbol is defined in two different binary files.

Tip: Rename the symbol defined in one of the specified files.

Freescale Semiconductor, Inc.

6.3.90 L1818: Symbol <Symbol Number> - < Symbol Name> Duplicated in <First Filename> and <Second Filename>

Type: Error

Description: The specified global symbol is defined in two different binary files.

Tip: Rename the symbol defined in one of the specified files.

6.3.91 L1820: Weak Symbol <Symbol Name> Duplicated in <First Filename> and <Second Filename>

Type: Warning

Description: The specified weak symbol is defined in two different binary files.

Tip: Rename the symbol defined in one of the specified files.

6.3.92 L1821: Symbol <id1> Conflicts with <id2> in File <File> (Same Code)

Type: Error

Description: A static symbol is defined twice in the same module.

Tip: Rename one of the symbols in the module.

6.3.93 L1822: Symbol <Symbol Name> in File <Filename> is Undefined

Type: Error

Description: The specified symbol is referenced in the file, but not defined anywhere in the application.

Tips:

- Check if an object file is missing in the NAMES block and if the correct binary file is used.
- Check if paths are defined correctly. The binary files must be located in one of the paths listed in the environment variables OBJPATH or GENPATH or in the project directory.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.94 L1823: External Object <Symbol Name> in <Filename> Created by Default

Type: Warning

Description: The specified symbol is referenced in the file, but not defined in the application. However, an external declaration for this object is available in at least one of the binary files. The object should be defined in the first binary file where it is externally defined.

This is only valid for ANSI C applications.

In this case, an external definition for a variable `var` is: `extern int var;`

The definition of the corresponding variable is: `int var;`

Tip: Define the specified symbol in one of the files building the application.

6.3.95 L1824: Invalid Mark Type for <Ident>

Type: Error

Description: Internal error. The object file is corrupt.

Tip: Recompile sources and contact a Motorola representative if error occurs again.

6.3.96 L1826: Can't Read File. <Filename> is not an ELF Library Containing ELF Objects (ELF Objects Expected)

Type: Error

Description: The specified file is not a valid library. The linker is only able to link uniform binary files together.

Tip: Recompile the source file to ELF object file format.

6.3.97 L1902: <Cmd> Command not Supported

Type: Error

Description: There are command keywords in the linker parameter file that are not yet implemented.

Tip: Only use commands specified in the linker manual.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

6.3.98 L1903: Unexpected Symbol in Link Parameter File

Type: Error

Description: Syntax error in linker parameter file. An illegal character appeared.

Tip: Ensure that the linker parameter file is specified as the file argument on the command line and not an executable file.

If the file is a link parameter file, edit it and replace the invalid character or symbol.

6.3.99 L1905: Invalid Section Attribute for Program Header

Type: Error

Description: Illegal object file

Tip: Recompile source files. Contact a Motorola representative if error continues to appear.

6.3.100 L1906: Fixup Out of Buffer (<Obj> Referenced at Offset <Address>)

Type: Error

Description: An illegal relocation of an object is detected in the object file <Object> at address <Address>.

Tip: Check the relocation at that address. The offset may be out of range for this relocation type. If not, it may be caused by a corrupt object file.

6.3.101 L1907: Fixup Overflow in <Object>, Type <objType> at Offset <Address>

Type: Error

Description: An illegal relocation of an object is detected in the object file <Object> at address <Address>. The type of object is given in <objType>.

Tip: Check the relocation at that address. The offset may be out of range for this relocation type. If not it may be caused by a corrupt object file.

Freescale Semiconductor, Inc.

6.3.102 L1908: Fixup Error in <Object>, Type <objType> at Offset <Address>

Type: Error

Description: An illegal relocation of an object is detected in the object file <Object> at address <Address>. The type of object is given in <objType>.

Tip: Check the relocation at that address. The offset may be out of range for this relocation type. If not, it may be caused by a corrupt object file.

6.3.103 L1910: Invalid Section Attribute for Program Header

Type: Error

Description: A program header needs specific section attributes that should not be changed.

Tip: This is an internal error and may be caused by a corrupt object file.

6.3.104 L1911: Program Header End is not Aligned on the End of a Section

Type: Warning

Description: The program has to be aligned with the end of a section.

6.3.105 L1912: Object <obj> Overlaps with Another (last addr: <addr>, Object Address: <objadr>

Description: The object <obj> overlaps with another object at address <addr>. The address of the object is given in <objadr>.

Tip: Place one of the objects somewhere else.

6.3.106 L1913: Object Filler Overlaps with Something Else

Type: Error

Description: An object filler overlaps with another object. This is not allowed.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.107 L1914: Invalid Object: <Object>

Type: Error

Description: An object of unknown type is detected in an object file. This is an internal error and may be caused by a corrupt object file or incompatible object formats.

Tip: Recompile sources and try to link again. Contact a Motorola representative if this error continues to appear.

6.3.108 L1915: Gap in <Ident> at <address> before <Object> is too Big

Type: Error

Description: Gaps more than 32 bytes are not allowed between succeeding objects in a section. Only gaps caused by alignment are allowed.

Tip: This is an internal error. Contact a Motorola representative if this error continues to appear.

6.3.109 L1916: Section Name <Section> is too Long. Name is Cut to 90 Characters Length

Type: Warning

Description: The length of a name is limited to 90 characters.

Tip: Rename the section and recompile.

6.3.110 L1919: Duplicate Definition of <Object> in Library File(s) <File1> and/or <File2> Discarded

Type: Warning

Description: An object definition is duplicated in a library.

Tip: Rename one of the objects and recompile.

Freescale Semiconductor, Inc.

6.3.111 L1921: Marking: Too Many Nested Procedure Calls

Type: Error

Description: The object file is corrupt.

Tip: Recompile and try to link again.

6.3.112 L1922: File <filename> Has DWARF Data of Different Version, DWARF Data may not be Generated

Type: Warning

Description: The linked files have different versions of debug information sections (*ELF/DWARF*). When linking object files from different vendors, this message might occur if the linker does not recognize the debug info in all object files.

This message may also appear if some object files do not have debug information. The generated absolute file may have some correct debug information, but probably not for all modules.

Tip: Recompile with one version for output.

6.3.113 L1927: Fixups for DWARF Section <sectionname> not Correctly Generated

Type: Error

Description: The linker has problems in generating fixups for a specific *DWARF* debug section.

Tip: The problem may occur when linking object files from different vendors. The debug information may be better than no debug.

6.3.114 L1928: Limitation: Code Size <num>

Type: Error

Description: This message appears in the demo version of the MCUez linker. The size of linked code is limited to 1 Kbyte.

Tip: Contact a Motorola representative.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.3.115 L1929: Limitation: Too many Mections (<num>)

Type: Error

Description: This message appears in the demo version of the MCUez linker. The number of sections is limited.

Tip: Contact a Motorola representative.

6.3.116 L1930: Unknown Fixup Type in <ident>, Type <type>, at Offset <offset>

Type: Error

Description: The object file is corrupt or linker version does not support compiler instructions.

Tip: Recompile sources and link again.

6.3.117 L1931: Program Header Begin is not Aligned on the Beginning of a Section

Type: Warning

Description: The program has to be aligned with the start of a section.

6.3.118 L1932: Program Header Overflow in <name> at <index>

Type: Error

Description: Overflow of the internal data structures. This may be caused by corrupt input files. The limit is defined for all imaginable cases and raised constantly with the amount of resources available on a modern PC.

Tip: Recompile sources. If this occurs again, then too many sections have been defined.

6.3.119 L1933: ELF: <details> Warning

Type: Warning

Description: Data in the file is not complete or consistent. The <details> specify the cause of the warning. Possible causes are listed in message L1934.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

6.3.120 L1934: ELF: <details> Error

Type: Error

Description: Error while reading an *ELF* object file. The <details> specify the cause of the error. Possible causes are:

- Cannot open <File> — see **6.3.60 L1309 : Cannot Open <File>**
- Read error in <File>
- Out of memory in <File> — see **6.3.84 L1803: Out of Memory in <Function Name>**
- No *ELF* Section Header Table found in <File> — see **6.3.85 L1804: No ELF Section Header Table Found in <Filename>**
- *ELF* file <File> is corrupted — see **6.3.86 L1806: ELF File <Filename> Appears to be Corrupted**
- String in <File> is too long — see **6.3.87 L1808: String Overflow in <Function Name>, Contact Vendor**
- Section <File> located in a segment with invalid qualifier — see **6.3.88 L1809: Section <Section Name> Located in a Segment with Invalid Qualifier**
- Programming language incompatible
- Incompatible memory model: <m1> in previous files and <m2> in current file — see **6.3.62 L1401: Incompatible Memory Model: <Memory Model Name> in Previous Files and <Memory Model Name> in Current File**.
- Incompatible processor: <cpu1> in previous files and <cpu2> in current file — see **6.3.61 L1400: Incompatible Processor: <Processor Name> in Previous Files and <Processor Name> in Current File**
- String buffer overrun in <File>
- <File> is not a valid *ELF* file — see **6.3.57 L1303: <Filename> is not a Valid ELF File**
- <File> is a not an *ELF* object file — see **6.3.59 L1305: <Filename> is not an ELF Format Object File (ELF Object File Expected)**
- File <File> not found — see **6.3.56 L1302: File <Filename> not Found**

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

- Requested section not found
- Program header not found
- Currently no file open
- Request is not valid
- Object <name> has an unknown type
- Fixup error: <cause>
- File is not a valid library file
- File is not a valid *ELF* library file
- *ELF* file corrupted
- *DWARF* fixup incorrect: <cause>
- Internal

6.3.121 L1936: ELF Output: <details> Error

Type: Error

Description: The <details> specify the cause of the error. Possible causes are:

- Cannot open <File> — see **6.3.60 L1309 : Cannot Open <File>**
- Out of memory in <File> — see **6.3.84 L1803: Out of Memory in <Function Name>**
- Wrong file type for <action>
- Write error in <File>
- No *ELF* Section Header defined in <File>
- String buffer overrun in <File>
- Wrong section type
- Internal buffer overflow in <Function>
- All local symbols before the first global one
- Currently no file open
- Request is not valid
- Internal

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

6.3.122 L1938: Type Clash in Segment (Corrupt Object: <name>)

Type: Error

Description: The object file is corrupt.

Tip: Recompile sources and link again. Contact a Motorola representative if the error continues.

6.3.123 L4000: Could not Open Object File (<objFile>) in NAMES List

Type: Error

Description: The linker could not open any object file in the NAMES list. This message prints out the name of the last file found in the NAMES list (<objFile>).

Tip: Ensure that path settings are correct. Object files are searched for in the current directory and in the list of paths specified in the environment variables OBJPATH and GENPATH.

6.3.124 L4001: Link Parameter File <PRMFile> not Found

Type: Error

Description: The specified source file does not exist or the search paths are not correctly set.

Tip: Ensure that path settings are correct. Linker parameter files are searched for in the current directory and in the list of paths specified with the environment variable GENPATH.

6.3.125 L4002: NAMES Section was not Found in Linker Parameter File <PRM File>

Type: Error

Description: The NAMES section was not found in the linker parameter file.

Tip: Ensure that a correct parameter file is passed to the linker.

Freescale Semiconductor, Inc.

6.3.126 L4004: Linking <PRM File> as ELF/DWARF Format Link Parameter File

Type: Information

Description: If the first file in the NAMES section is an *ELF/DWARF* object file, this message is issued and the *ELF/DWARF* object file format is started.

6.3.127 L4005: Illegal File Format of Object File (<objFile>) in NAMES List

Type: Error

Description: No object file in the NAMES list contains a known file format. This message prints out the name of the last file in the NAMES list that was opened (<objFile>).

Tip: Ensure that path settings are correct.

6.3.128 L4006: Failed to Create Temporary File

Type: Error

Description: The linker creates a temporary file in the current directory when prescanning the linker parameter file. If this fails, the linker cannot continue.

Tip: Enable read access to linker files in the current directory.

6.3.129 L4007: Include File Nesting too Deep in Link Parameter File

Type: Error

Description: Include files can only be nested six deep.

6.3.130 L4008: Include File <includefile> not Found

Type: Error

Description: The include file <includefile> was not found.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Index

Symbols

.abs 19, 38
.copy 65, 85
.data. 65, 66
.map. 38
.prm 37
.rodata 65
.rodata1 65
.s1 38
.s2 38
.s3 38
.stack 65
.startData 65, 85
.sx 38
.text 65

A

Absolute File. 19, 38, 47, 52
ABSPATH 38, 44
Application
 Startup (also see Startup) 84
Assembly
 Application 47, 80
 Smart Linking 81

C

Command
 ENTRIES 45, 79, 80
 INIT. 47, 91
 LINK. 47, 93
 MAIN 49
 MAPFILE 49, 92
 NAMES. 44, 52
 PLACEMENT. 44, 61, 66
 SEGMENTS 44, 53

Freescale Semiconductor, Inc.

STACKSIZE . . . 68
STACKTOP . . . 69
VECTOR. . . . 70

D

Drag and Drop. . . . 35

E

-E option . . . 91
ENTRIES . . . 45, 79, 80
Environment Variable
ABSPATH . . . 38, 44, 47, 100
ERRORFILE. . . . 102
GENPATH . . . 37, 44, 52, 98, 99
LINKPTIONS . . . 90
OBJPATH. . . . 44, 52, 99
SRECORD . . . 38, 101
TEXTPATH . . . 38, 44, 48, 100
Error feedback. . . . 35

F

File
Absolute . . . 19, 38, 47, 52
Library. . . . 52
MAP . . . 38, 48, 49
Motorola S . . . 38
Object . . . 52
Parameter . . . 37
Parameter (Linker) . . . 37, 42
File Menu . . . 27

G

GENPATH . . . 44, 52

I

INIT. . . . 47, 91
Input . . . 31
Input File. . . . 34

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.

L

Library File 52
LINK 47, 93
Linker Menu 31

M

-M Option 92
MAIN 49
MAP File 38, 48, 49
  COPYDOWN 50
  FILE 50
  OBJECT ALLOCATION 50
  OBJECT DEPENDENCY 50
  SEGMENT ALLOCATION 50
  STARTUP 50
  STATISTICS 50
  TARGET 50
  UNUSED OBJECTS 50
MAPFILE 49, 92
Menu Bar 27
MESSAGE 31
Message
  ERROR 108
  FATAL 108
  WARNING 108
Motorola S File 38

N

NAMES 44, 52
NO_INIT 55

O

-O Option 93
Object File 52
OBJPATH 44, 52
Option
  -E 91
  -M 92
  -O 93
  -S 93

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

-V 91, 94
-W1 94
-W2 . . . . 94, 95, 96
Output 31

P

PAGED 55
Parameter
File (Linker) 37, 42
Parameter File 37
Path List 97
PLACEMENT 44, 61, 66
Program Startup (also see Startup) 84

Q

Qualifier 53, 55
NO_INIT 55
PAGED 55
READ_ONLY 55
READ_WRITE 55

R

READ_ONLY 55
READ_WRITE 55

S

-S Option 93
Section
.copy 65, 85
.data 65, 66
.rodata 65
.stack 65
.startData 65, 85
.text 65
Segment
Alignment 53, 56
Ffill pattern 53
Fill Pattern 59
Qualifier 53, 55
SEGMENTS 44, 53
Smart Linking 19, 79
STACKSIZE 68

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

STACKTOP 69
Startup
    Application 84
Startup Function 88
    User Defined 88
Startup Structure 84
    flags 85
    initBodies 87
    libInits 86, 87
    main 86
    nofInitBodies 87
    nofLibInits 86, 87
    nofZeroOuts 86
    pZeroOut 86
    stackOffset 86
    toCopyDownBeg 86
    User Defined 87
Startup.TXT 84
Status Bar 27

T

TEXTPATH 38, 44
Toolbar 25

V

-V Option 91, 94
VECTOR 70
Vector 20
View Menu 34

W

-W1 option 94
-W2 Option 94, 95, 96
Window 25

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

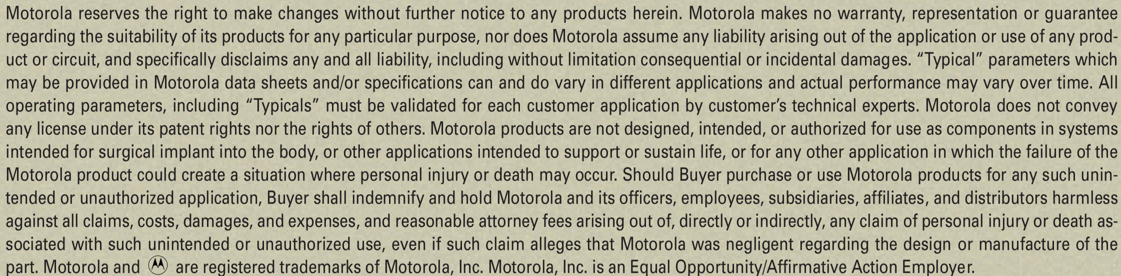
Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Need to know more? That's ez, too.

Technical support for MCUez development tools is available through your regional Motorola office or by contacting:

Motorola, Inc.
6501 William Cannon Drive West
MD:OE17
Austin, Texas 78735
Phone (800) 521-6274
Fax (602) 437-1858
CRC@CRC.email.sps.mot.com



Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Motorola data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and Ⓜ are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

How to reach us:

USA/EUROPE/Locations Not Listed: Motorola Literature Distribution; P.O. Box 5405, Denver, Colorado 80217, 1–800–441–2447 or 1-303-675-2140.
Customer Focus Center: 1–800–521–6274

JAPAN: Motorola Japan Ltd.; SPD, Strategic Planning Office, 141, 4–32–1, Nishi–Gotanda, Shinagawa–ku, Tokyo, Japan, 03–5487–8488

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd., Silicon Harbour Centre, 2 Dal King Street, Tai Po Industrial Estate, Tai Po, New Territories, Hong Kong, 852–26668334

Mfax™, Motorola Fax Back System: RMFAX0@email.sps.mot.com; http://sps.motorola.com/mfax/; TOUCHTONE, 1-602–244–8609; US & Canada ONLY, 1–800–774–1848

HOME PAGE: http://motorola.com/sps/

Mfax is a trademark of Motorola, Inc.

MOTORO
Semiconductor Produ

Freescale Semiconduct nc...