



Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!



Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832

Email & Skype: info@chipsmall.com Web: www.chipsmall.com

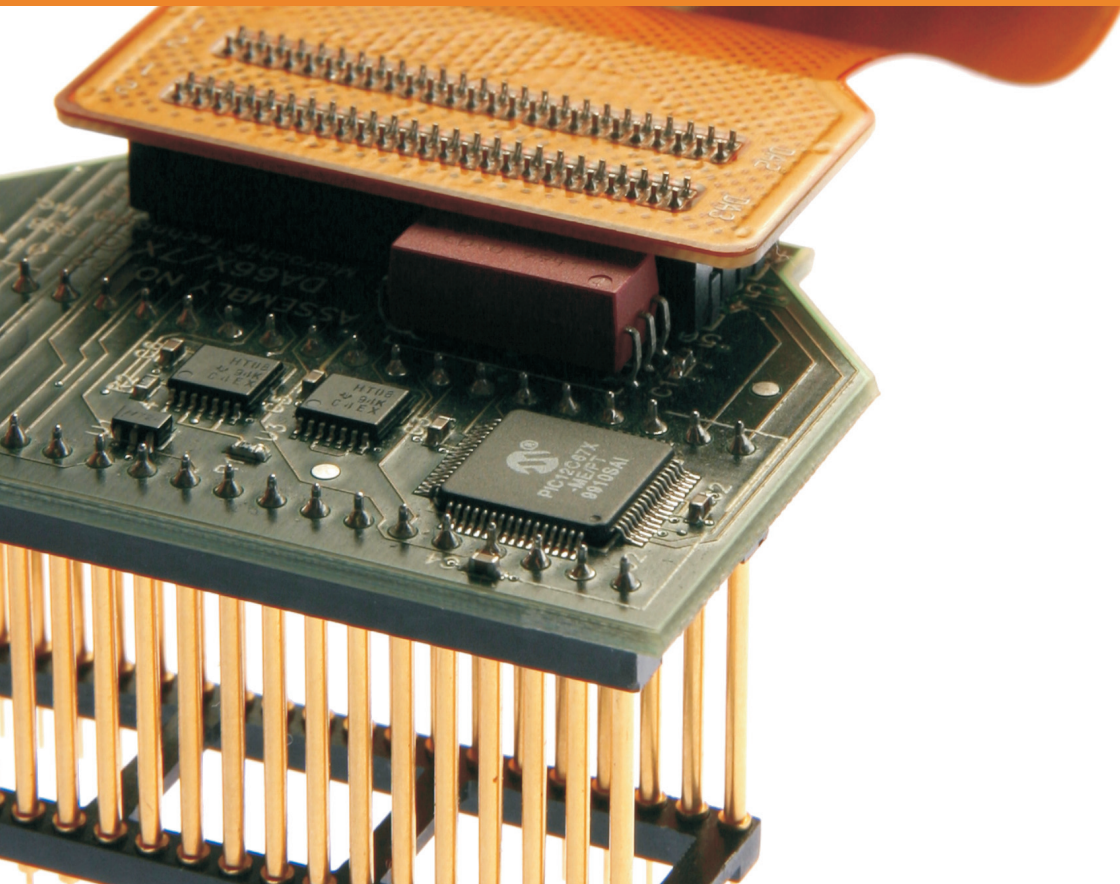
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China



HI-TECH
S O F T W A R E

PICC™ STD

A N S I C C O M P I L E R





HI-TECH PICC STD Compiler

HI-TECH Software.

Copyright (C) 2008 HI-TECH Software.
All Rights Reserved. Printed in Australia.
PICC is licensed exclusively to HI-TECH Software
by Microchip Technology Inc.
Produced on: February 21, 2008

HI-TECH Software Pty. Ltd.
ACN 002 724 549
45 Colebard Street West
Acacia Ridge QLD 4110
Australia

email: hitech@htsoft.com
web: <http://www.htsoft.com>
ftp: <ftp://www.htsoft.com>

Contents

Table of Contents	iii
List of Tables	xiii
1 Introduction	1
1.1 Typographic conventions	1
2 PICC Command-line Driver	3
2.1 Long Command Lines	4
2.2 Default Libraries	4
2.3 Standard Runtime Code	4
2.4 PICC Compiler Options	5
2.4.1 -C: Compile to Object File	7
2.4.2 -D <i>macro</i> : Define Macro	8
2.4.3 -E <i>file</i> : Redirect Compiler Errors to a File	8
2.4.4 -G <i>file</i> : Generate Source-level Symbol File	9
2.4.5 -I <i>path</i> : Include Search Path	9
2.4.6 -L <i>library</i> : Scan Library	10
2.4.7 -L- <i>option</i> : Adjust Linker Options Directly	10
2.4.8 -M <i>file</i> : Generate Map File	12
2.4.9 -N <i>size</i> : Identifier Length	12
2.4.10 -O <i>file</i> : Specify Output File	12
2.4.11 -P: Preprocess Assembly Files	13
2.4.12 -Q: Quiet Mode	13
2.4.13 -S: Compile to Assembler Code	13
2.4.14 -U <i>macro</i> : Undefine a Macro	13
2.4.15 -V: Verbose Compile	14
2.4.16 -X: Strip Local Symbols	14

2.4.17	--ASMLIST: Generate Assembler .LST Files	14
2.4.18	--CHAR= <i>type</i> : Make Char Type Signed or Unsigned	14
2.4.19	--CHECKSUM= <i>start-end@destination<, specs></i> : Calculate a check- sum	14
2.4.20	--CHIP= <i>processor</i> : Define Processor	15
2.4.21	--CHIPINFO: Display List of Supported Processors	15
2.4.22	--CODEOFFSET: Offset Program Code to Address	15
2.4.23	--CR= <i>file</i> : Generate Cross Reference Listing	15
2.4.24	--DEBUGGER= <i>type</i> : Select Debugger Type	16
2.4.25	--DOUBLE= <i>type</i> : Select kind of Double Types	16
2.4.26	--ECHO: Echo compiler command line	16
2.4.27	--ERRFORMAT= <i>format</i> : Define Format for Compiler Messages	16
	2.4.27.1 Using the Format Options	17
	2.4.27.2 Modifying the Standard Format	17
2.4.28	--ERRORS= <i>number</i> : Maximum Number of Errors	18
2.4.29	--FILL= <i>opcode</i> : Fill Unused Program Memory	18
2.4.30	--GETOPTION= <i>app, file</i> : Get Command-line Options	19
2.4.31	--HELP[= <i>option</i>]: Display Help	19
2.4.32	--IDE= <i>type</i> : Specify the IDE being used	19
2.4.33	--LANG= <i>language</i> : Specify the Language for Messages	19
2.4.34	--MEMMAP= <i>file</i> : Display Memory Map	20
2.4.35	--MSGDISABLE= <i>messagelist</i> : Disable Warning Messages	20
2.4.36	--MSGFORMAT= <i>format</i> : Set Advisory Message Format	20
2.4.37	--NODEL: Do not remove temporary files	20
2.4.38	--NOEXEC: Don't Execute Compiler	20
2.4.39	--OPT[= <i>type</i>]: Invoke Compiler Optimizations	20
2.4.40	--OUTDIR: Specify a directory for output files	21
2.4.41	--OUTPUT= <i>type</i> : Specify Output File Type	21
2.4.42	--PRE: Produce Preprocessed Source Code	21
2.4.43	--PROTO: Generate Prototypes	21
2.4.44	--RAM= <i>lo-hi, <lo-hi, . . .></i> : Specify Additional RAM Ranges	23
2.4.45	--ROM= <i>lo-hi, <lo-hi, . . .> tag</i> : Specify Additional ROM Ranges	24
2.4.46	--RUNTIME= <i>type</i> : Specify Runtime Environment	24
2.4.47	--SCANDEP: Scan for Dependencies	24
2.4.48	--SERIAL= <i>hexcode@address</i> : Store a Value at this Program Memory Address	26
2.4.49	--SETOPTION= <i>app, file</i> : Set The Command-line Options for Application	26
2.4.50	--SETUP= <i>dir</i> : Setup the product	26
2.4.51	--STRICT: Strict ANSI Conformance	26

2.4.52	--SUMMARY= <i>type</i> : Select Memory Summary Output Type	27
2.4.53	--TIME: Report time taken for each phase of build process	27
2.4.54	--VER: Display The Compiler's Version Information	27
2.4.55	--WARN= <i>level</i> : Set Warning Level	28
2.4.56	--WARNFORMAT= <i>format</i> : Set Warning Message Format	28
3	C Language Features	29
3.1	ANSI Standard Issues	29
3.1.1	Implementation-defined behaviour	29
3.2	Processor-related Features	29
3.2.1	Stack	29
3.2.2	Configuration Fuses	30
3.2.3	ID Locations	30
3.2.4	Bit Instructions	31
3.2.5	EEPROM Access	31
3.2.5.1	The <i>eeeprom</i> variable qualifier	31
3.2.5.2	The <code>__EEPROM_DATA()</code> macro	32
3.2.5.3	EEPROM Access Functions	33
3.2.5.4	EEPROM Access Macros	33
3.2.6	Flash Runtime Access	34
3.2.6.1	Flash Access Macros	34
3.2.6.2	Flash Access Functions	35
3.2.7	Baseline PIC special instructions	35
3.2.7.1	The OPTION instruction	35
3.2.7.2	The TRIS instructions	35
3.2.7.3	Calibration Space	36
3.2.7.4	Oscillator calibration constants	36
3.3	Files	37
3.3.1	Source Files	37
3.3.2	Symbol Files	37
3.3.3	Standard Libraries	37
3.3.4	Runtime startup Modules	38
3.3.4.1	Initialization of Data psects	39
3.3.4.2	Clearing the Bss Psects	40
3.3.4.3	Linking in the C Libraries	40
3.3.4.4	The powerup Routine	40
3.4	Supported Data Types and Variables	41
3.4.1	Radix Specifiers and Constants	41
3.4.2	Bit Data Types and Variables	42

3.4.3	8-Bit Integer Data Types and Variables	44
3.4.4	16-Bit Integer Data Types	44
3.4.5	32-Bit Integer Data Types and Variables	44
3.4.6	Floating Point Types and Variables	45
3.4.7	Structures and Unions	46
3.4.7.1	Bit-fields in Structures	46
3.4.7.2	Structure and Union Qualifiers	47
3.4.8	Standard Type Qualifiers	48
3.4.8.1	Const and Volatile Type Qualifiers	48
3.4.9	Special Type Qualifiers	48
3.4.9.1	Persistent Type Qualifier	49
3.4.9.2	Bank1, Bank2 and Bank3 Type Qualifiers	49
3.4.10	Eeprom Type Qualifier	50
3.4.11	Pointer Types	50
3.4.11.1	Baseline Pointers	50
3.4.11.2	Midrange Pointers	50
3.4.11.3	Highend Pointers	51
3.4.11.4	Qualifiers and Pointers	51
3.5	Storage Class and Object Placement	52
3.5.1	Local Variables	52
3.5.1.1	Auto Variables	53
3.5.1.2	Static Variables	53
3.5.2	Absolute Variables	53
3.6	Functions	54
3.6.1	Function Argument Passing	54
3.6.2	Function Return Values	55
3.6.2.1	Structure Return Values	56
3.7	Function Calling Convention	56
3.8	Operators	57
3.8.1	Integral Promotion	57
3.8.2	Shifts applied to integral types	59
3.8.3	Division and modulus with integral types	59
3.9	Psects	59
3.9.1	Compiler-generated Psects	60
3.10	Interrupt Handling in C	62
3.10.1	Interrupt Functions	62
3.10.1.1	Midrange Interrupt Functions	62
3.10.1.2	Highend Interrupt Functions	63
3.10.1.3	Context Saving on Interrupts	63

3.10.1.4	MidRange Context Saving	63
3.10.1.5	High-End Context Saving	64
3.10.1.6	Context Restoration	64
3.10.1.7	Interrupt Levels	64
3.10.1.8	Multiple Interrupts on High-End PICs	65
3.10.1.9	Enabling Interrupts	66
3.11	Mixing C and Assembler Code	66
3.11.1	External Assembly Language Functions	66
3.11.2	#asm, #endasm and asm()	68
3.11.3	Accessing C objects from within Assembly Code	69
3.11.3.1	Equivalent Assembly Symbols	69
3.11.3.2	Accessing special function register names from assembler	69
3.12	Preprocessing	70
3.12.1	Preprocessor Directives	70
3.12.2	Predefined Macros	70
3.12.3	Pragma Directives	70
3.12.3.1	The #pragma inline Directive	70
3.12.3.2	The #pragma jis and nojis Directives	73
3.12.3.3	The #pragma pack Directive	73
3.12.3.4	The #pragma printf_check Directive	74
3.12.3.5	The #pragma psect Directive	74
3.12.3.6	The #pragma regsused Directive	75
3.12.3.7	The #pragma switch Directive	76
3.13	Linking Programs	77
3.13.1	Replacing Library Modules	77
3.13.2	Signature Checking	78
3.13.3	Linker-Defined Symbols	79
3.14	Standard I/O Functions and Serial I/O	79
4	Macro Assembler	81
4.1	Assembler Usage	81
4.2	Assembler Options	82
4.3	HI-TECH C Assembly Language	84
4.3.1	Statement Formats	84
4.3.2	Characters	85
4.3.2.1	Delimiters	85
4.3.2.2	Special Characters	85
4.3.3	Comments	85
4.3.3.1	Special Comment Strings	85

4.3.4	Constants	86
4.3.4.1	Numeric Constants	86
4.3.4.2	Character Constants and Strings	86
4.3.5	Identifiers	86
4.3.5.1	Significance of Identifiers	87
4.3.5.2	Assembler-Generated Identifiers	87
4.3.5.3	Location Counter	87
4.3.5.4	Register Symbols	88
4.3.5.5	Symbolic Labels	88
4.3.6	Expressions	89
4.3.7	Program Sections	89
4.3.8	Assembler Directives	91
4.3.8.1	GLOBAL	91
4.3.8.2	END	91
4.3.8.3	PSECT	93
4.3.8.4	ORG	95
4.3.8.5	EQU	95
4.3.8.6	SET	95
4.3.8.7	DB	96
4.3.8.8	DW	96
4.3.8.9	DS	96
4.3.8.10	FNADDR	96
4.3.8.11	FNARG	96
4.3.8.12	FNBREAK	97
4.3.8.13	FNCALL	97
4.3.8.14	FNCONF	97
4.3.8.15	FNINDIR	98
4.3.8.16	FNSIZE	98
4.3.8.17	FNROOT	98
4.3.8.18	IF, ELSIF, ELSE and ENDIF	99
4.3.8.19	MACRO and ENDM	99
4.3.8.20	LOCAL	100
4.3.8.21	ALIGN	101
4.3.8.22	REPT	101
4.3.8.23	IRP and IRPC	102
4.3.8.24	PROCESSOR	103
4.3.8.25	SIGNAT	103
4.3.9	Assembler Controls	103
4.3.9.1	COND	103

4.3.9.2	EXPAND	103
4.3.9.3	INCLUDE	104
4.3.9.4	LIST	104
4.3.9.5	NOCOND	104
4.3.9.6	NOEXPAND	105
4.3.9.7	NOLIST	105
4.3.9.8	NOXREF	105
4.3.9.9	PAGE	105
4.3.9.10	SPACE	105
4.3.9.11	SUBTITLE	105
4.3.9.12	TITLE	106
4.3.9.13	XREF	106
5	Linker and Utilities	107
5.1	Introduction	107
5.2	Relocation and Psects	107
5.3	Program Sections	108
5.4	Local Psects	108
5.5	Global Symbols	108
5.6	Link and load addresses	109
5.7	Compiled Stack Operation	109
5.7.1	Parameters involving Function Calls	110
5.7.2	Implicit Calls to Assembler Library Routines	112
5.8	Map Files	112
5.8.1	Generation	112
5.8.2	Contents	113
5.8.2.1	General Information	113
5.8.2.2	Call Graph Information	115
5.8.2.3	Psect Information listed by Module	120
5.8.2.4	Psect Information listed by Class	122
5.8.2.5	Segment Listing	123
5.8.2.6	Unused Address Ranges	123
5.8.2.7	Symbol Table	123
5.9	Operation	124
5.9.1	Numbers in linker options	126
5.9.2	-Aclass=low-high,...	126
5.9.3	-Cx	126
5.9.4	-Cpsect=class	126
5.9.5	-Dclass=delta	127

5.9.6	-Dsymfile	127
5.9.7	-Eerrfile	127
5.9.8	-F	127
5.9.9	-Gspec	127
5.9.10	-Hsymfile	128
5.9.11	-H+symfile	128
5.9.12	-Jerrcount	128
5.9.13	-K	128
5.9.14	-I	129
5.9.15	-L	129
5.9.16	-LM	129
5.9.17	-Mmapfile	129
5.9.18	-N, -Ns and -Nc	129
5.9.19	-Ooutfile	129
5.9.20	-Pspec	129
5.9.21	-Qprocessor	131
5.9.22	-S	131
5.9.23	-Sclass=limit[, bound]	131
5.9.24	-Usymbol	132
5.9.25	-Vavmap	132
5.9.26	-Wnum	132
5.9.27	-X	132
5.9.28	-Z	132
5.10	Invoking the Linker	133
5.11	Map Files	133
5.11.1	Call Graph Information	134
5.12	Librarian	136
5.12.1	The Library Format	137
5.12.2	Using the Librarian	137
5.12.3	Examples	138
5.12.4	Supplying Arguments	138
5.12.5	Listing Format	139
5.12.6	Ordering of Libraries	139
5.12.7	Error Messages	139
5.13	Objtohex	139
5.13.1	Checksum Specifications	141
5.14	Cref	141
5.14.1	-Fprefix	142
5.14.2	-Hheading	142

5.14.3	-Llen	142
5.14.4	-Ooutfile	142
5.14.5	-Pwidth	143
5.14.6	-Sstoplist	143
5.14.7	-Xprefix	143
5.15	Cromwell	143
5.15.1	-Pname[,architecture]	143
5.15.2	-N	145
5.15.3	-D	145
5.15.4	-C	145
5.15.5	-F	145
5.15.6	-Okey	146
5.15.7	-Ikey	146
5.15.8	-L	146
5.15.9	-E	146
5.15.10	-B	146
5.15.11	-M	146
5.15.12	-V	146
5.16	Hexmate	146
5.16.1	Hexmate Command Line Options	147
5.16.1.1	specifications,filename.hex	149
5.16.1.2	+ Prefix	149
5.16.1.3	-ADDRESSING	150
5.16.1.4	-BREAK	150
5.16.1.5	-CK	150
5.16.1.6	-FILL	151
5.16.1.7	-FIND	152
5.16.1.8	-FIND...,DELETE	153
5.16.1.9	-FIND...,REPLACE	153
5.16.1.10	-FORMAT	153
5.16.1.11	-HELP	154
5.16.1.12	-LOGFILE	155
5.16.1.13	-Ofile	155
5.16.1.14	-SERIAL	155
5.16.1.15	-SIZE	156
5.16.1.16	-STRING	156
5.16.1.17	-STRPACK	156

B Error and Warning Messages	275
C Chip Information	403
Index	409

List of Tables

2.1	PICC file types	3
2.3	Default values for filling unprogrammed code space	15
2.4	Supported Double Types	16
2.5	Error format specifiers	17
2.6	Supported IDEs	19
2.7	Supported languages	19
2.8	Optimization Options	21
2.9	Output file formats	22
2.10	Runtime environment suboptions	25
2.11	Memory Summary Suboptions	27
3.1	Basic data types	41
3.2	Radix formats	42
3.3	Floating-point formats	45
3.4	Floating-point format example IEEE 754	46
3.5	Integral division	59
3.6	Preprocessor directives	71
3.7	Predefined macros	72
3.9	Pragma directives	73
3.10	Valid Register Names	76
3.11	switch types	76
3.12	Supported standard I/O functions	79
4.1	ASPIC command-line options	82
4.2	ASPICstatement formats	85
4.3	ASPIC numbers and bases	86
4.4	ASPIC operators	90

4.5	ASPIC assembler directives	92
4.6	PSECT flags	93
4.7	ASPIC assembler controls	104
4.8	LIST control options	105
5.1	Linker command-line options	125
5.2	Librarian command-line options	137
5.3	Librarian key letter commands	137
5.4	OBJTOHEX command-line options	140
5.5	CREF command-line options	142
5.6	CROMWELL format types	144
5.7	CROMWELL command-line options	144
5.8	-P option architecture arguments for COFF file output.	145
5.9	Hexmate command-line options	148
5.10	Hexmate Checksum Algorithm Selection	151
5.11	INHX types used in -FORMAT option	154
C.1	Devices supported by HI-TECH PICC STD	403
C.1	Devices supported by HI-TECH PICC STD	404
C.1	Devices supported by HI-TECH PICC STD	405
C.1	Devices supported by HI-TECH PICC STD	406
C.1	Devices supported by HI-TECH PICC STD	407
C.1	Devices supported by HI-TECH PICC STD	408

Chapter 1

Introduction

1.1 Typographic conventions

Different fonts and styles are used throughout this manual to indicate special words or text. Computer prompts, responses and filenames will be printed in `constant-spaced` type. When the filename is the name of a standard header file, the name will be enclosed in angle brackets, e.g. `<stdio.h>`. These header files can be found in the `INCLUDE` directory of your distribution.

Samples of code, C keywords or types, assembler instructions and labels will also be printed in a `constant-space` type. Assembler code is printed in a font similar to that used by C code.

Particularly useful points and new terms will be emphasized using *italicized type*. When part of a term requires substitution, that part should be printed in the appropriate font, but in *italics*. For example: `#include <filename.h>`.

Chapter 2

PICC Command-line Driver

PICC is the driver invoked from the command line to compile and/or link C programs. PICC has the following basic command format:

```
PICC [options] files [libraries]
```

It is conventional to supply the options (identified by a leading *dash* “-” or *double dash* “-”) before the filenames.

The options are discussed below. The files may be a mixture of source files (C or assembler) and object files. The order of the files is not important, except that it will affect the order in which code or data appears in memory. *Libraries* are a list of library names, or -L options, see Section 2.4.6. Source files, object files and library files are distinguished by PICC solely by the *file type* or *extension*. Recognized file types are listed in Table 2.1. This means, for example, that an assembler file must always have a .as extension (alphabetic case is not important).

PICC will check each file argument and perform appropriate actions. C files will be compiled; assembler files will be assembled. At the end, unless suppressed by one of the options discussed later,

Table 2.1: PICC file types

File Type	Meaning
.c	C source file
.as	Assembler source file
.obj	Relocatable object code file
.lib	Relocatable object library file
.hex	Intel HEX file

all object files resulting from compilation or assembly, or those listed explicitly on the command line, will be linked together with the standard runtime code and libraries and any user-specified libraries. Functions in libraries will be linked into the resulting output file only if referenced in the source code.

Invoking `PICC` with only object files specified as the file arguments (i.e. no source files) will mean only the link stage is performed. It is typical in Makefiles to use `PICC` with a `-C` option to compile several source files to object files, then to create the final program by invoking `PICC` again with only the generated object files and appropriate libraries (and appropriate options). If a `.lib` output file type is selected, the object files will be stored in a library instead of going through to the final link.

When a HEX file is given on the command line, `PICC` will invoke the `HEXMATE` utility and will merge the named hex file with the hex file currently being generated. This feature can be useful when, for example, a single hex file is desired which contains a bootloader and application program.

2.1 Long Command Lines

The `PICC` driver is capable of processing command lines exceeding any operating system limitation. To do this, the driver may be passed options via a command file. The command file is read by using the `@` symbol. For example:

```
PICC @xyz.cmd
```

2.2 Default Libraries

`PICC` will search the appropriate standard C library by default for symbol definitions. This will always be done last, after any user-specified libraries. The particular library used will be dependent on the processor selected.

2.3 Standard Runtime Code

`PICC` will automatically generate standard runtime start-up code appropriate for the processor and options selected unless you have specified the to disable this via the `--RUNTIME` option. If you require any special powerup initialization, you should use the *powerup* routine feature (see Section 3.3.4.4).

2.4 PICC Compiler Options

Most aspects of the compilation can be controlled using the command-line driver, PICC. The driver will configure and execute all required applications, such as the code generator, assembler and linker.

PICC recognizes the compiler options listed in the table below. The case of the options is not important, however command shells in UNIX based operating systems are case sensitive when it comes to names of files.

PICC Command-line Options

Option	Meaning
-C	Compile to object files only
-D <i>macro</i>	Define preprocessor macro
-E+ <i>file</i>	Redirect and optionally append errors to a file
-G <i>file</i>	Generate source-level debugging information
-I <i>path</i>	Specify a directory pathname for include files
-L <i>library</i>	Specify a library to be scanned by the linker
-L- <i>option</i>	Specify - <i>option</i> to be passed directly to the linker
-M <i>file</i>	Request generation of a MAP file
-N <i>size</i>	Specify identifier length
-O <i>file</i>	Set basename for output files
-P	Preprocess assembler files
-Q	Specify quiet mode
-S	Compile to assembler source files only
-U <i>symbol</i>	Undefine a predefined preprocessor symbol
-V	Verbose: display compiler pass command lines
-X	Eliminate local symbols from symbol table
--ASMLIST	Generate assembler .LST file for each compilation
--CHAR= <i>type</i>	Make the default char signed or unsigned
--CHECKSUM= <i>start-end@dest</i>	Calculate a checksum over an address range
--CHIP= <i>processor</i>	Selects which processor to compile for
--CHIPINFO	Displays a list of supported processors
--CODEOFFSET= <i>address</i>	Offset program code to address
--CR= <i>file</i>	Generate cross-reference listing
--DEBUGGER= <i>type</i>	Select the debugger that will be used
--DOUBLE= <i>type</i>	Selects size/kind of double types
--ECHO	Echo the PICC command line
--ERRFORMAT<= <i>format</i> >	Format error message strings to the given style
<i>continued...</i>	

PICC Command-line Options

Option	Meaning
--ERRORS= <i>number</i>	Sets the maximum number of errors displayed
--FILL= <i>opcode</i>	Fill unused program locations with this hexadecimal code
--GETOPTION= <i>app, file</i>	Get the command line options for the named application
--HELP<= <i>option</i> >	Display the compiler's command line options
--IDE= <i>ide</i>	Configure the compiler for use by the named IDE
--LANG= <i>language</i>	Specify language for compiler messages
--MEMMAP= <i>file</i>	Display memory summary information for the map file
--MSGFORMAT<= <i>format</i> >	Format general message strings to the given style
--MSGDISABLE<= <i>numbers</i> >	Disable these warning or advisory messages
--NODEL	Do not remove temporary files generated by the compiler
--NOEXEC	Go through the motions of compiling without actually compiling
--OPT<= <i>type</i> >	Enable general compiler optimizations
--OUTDIR	Specify output files directory
--OUTPUT= <i>type</i>	Generate output file type
--PRE	Produce preprocessed source files
--PROTO	Generate function prototype information
--RAM=lo-hi<,lo-hi,...>	Specify and/or reserve RAM ranges
--ROM=lo-hi<,lo-hi,...>	Specify and/or reserve ROM ranges
--RUNTIME= <i>type</i>	Configure the C runtime libraries to the specified type
--SCANDEP	Generate file dependency ".DEP files"
--SERIAL= <i>code@address</i>	Store this hexadecimal code at an address in program memory
--SETOPTION= <i>app, file</i>	Set the command line options for the named application
--SETUP= <i>argument</i>	Setup the product
--STRICT	Enable strict ANSI keyword conformance
--SUMMARY= <i>type</i>	Selects the type of memory summary output
--TIME	Show execution time in each stage of build process
--VER	Display the compiler's version number
--WARN= <i>level</i>	Set the compiler's warning level
<i>continued...</i>	

PICC Command-line Options

Option	Meaning
--WARNFORMAT= <i>format</i>	Format warning message strings to given style

All single letter options are identified by a leading *dash* character, “-”, e.g. -C. Some single letter options specify an additional data field which follows the option name immediately and without any whitespace, e.g. -Ddebug.

Multi-letter, or word, options have two leading *dash* characters, e.g. --ASMLIST. (Because of the double *dash*, you can determine that the option --ASMLIST, for example, is not a -A option followed by the argument SMLIST.) Some of these options define suboptions which typically appear as a *comma*-separated list following an *equal* character, =, e.g. --OUTPUT=intel,cof. The exact format of the options varies and are described in detail in the following sections.

Some commonly used suboptions include *default*, which represent the default specification that would be used if this option was absent altogether; *all*, which indicates that all the available suboptions should be enabled as if they had each been listed; and *none*, which indicates that all suboptions should be disabled. Some suboptions may be prefixed with a plus character, +, to indicate that they are in addition to the other suboptions present, or a minus character “-”, to indicate that they should be excluded. In the following sections, *angle brackets*, <>, are used to indicate optional parts of the command.

2.4.1 -C: Compile to Object File

The -C option is used to halt compilation after generating a relocatable object file. This option is frequently used when compiling multiple source files using a “make” utility. If multiple source files are specified to the compiler each will be compiled to a separate .obj file. The object files will be placed in the directory in which PICC was invoked, to handle situations where source files are located in read-only directories. To compile three source files main.c, module1.c and asmcode.as to object files you could use a command similar to:

```
PICC --CHIP=16F877A -C main.c module1.c asmcode.as
```

The compiler will produce three object files main.obj, module1.obj and asmcode.obj which could then be linked to produce an *Intel* HEX file using the command:

```
PICC --CHIP=16F877A main.obj module1.obj asmcode.obj
```

2.4.2 **-Dmacro: Define Macro**

The `-D` option is used to define a preprocessor macro on the command line, exactly as if it had been defined using a `#define` directive in the source code. This option may take one of two forms, `-Dmacro` which is equivalent to:

```
#define macro 1
```

placed at the top of each module compiled using this option, or `-Dmacro=text` which is equivalent to:

```
#define macro text
```

where *text* is the textual substitution required. Thus, the command:

```
PICC --CHIP=16F877A -Ddebug -Dbuffers=10 test.c
```

will compile `test.c` with macros defined exactly as if the C source code had included the directives:

```
#define debug 1
#define buffers 10
```

2.4.3 **-Efile: Redirect Compiler Errors to a File**

This option has two purposes. The first is to change the format of displayed messages. The second is to optionally allow messages to be directed to a file as some editors do not allow the standard command line redirection facilities to be used when invoking the compiler.

The general form of messages produced with the `-E` option in force is:

```
filename line_number: (message number) message string (message type)
```

If a filename is specified immediately after `-E`, it is treated as the name of a file to which all messages (errors, warnings etc) will be printed. For example, to compile `x.c` and redirect all errors to `x.err`, use the command:

```
PICC --CHIP=16F877A -Ex.err x.c
```

The `-E` option also allows errors to be appended to an existing file by specifying an *addition* character, `+`, at the start of the error filename, for example:

```
PICC --CHIP=16F877A -E+x.err y.c
```


If you wish to compile several files and combine all of the errors generated into a single text file, use the `-E` option to create the file then use `-E+` when compiling all the other source files. For example, to compile a number of files with all errors combined into a file called `project.err`, you could use the `-E` option as follows:

```
PICC --CHIP=16F877A -Eproject.err -O -C main.c
PICC --CHIP=16F877A -E+project.err -O -C part1.c
PICC --CHIP=16F877A -E+project.err -C asmcode.as
```

Section [2.4.27.1](#) has more information regarding this option as well as an overview of the messaging system and other related driver options.

2.4.4 `-Gfile`: Generate Source-level Symbol File

The `-G` option generates a *source-level symbol file* (i.e. a file which allows tools to determine which line of source code is associated with machine code instructions, and determine which source-level variable names correspond with areas of memory, etc.) for use with supported debuggers and simulators such as HI-TIDE™ and MPLAB®. If no filename is given, the symbol file will have the same base name as the project name, and an extension of `.sym`. For example the option `-Gtest.sym` generates a symbol file called `test.sym`. Symbol files generated using the `-G` option include source-level information for use with source-level debuggers.

Note that all source files for which source-level debugging is required should be compiled with the `-G` option. The option is also required at the link stage, if this is performed separately. For example:

```
PICC --CHIP=16F877A -G -C test.c modules1.c
PICC --CHIP=16F877A -Gtest.sym test.pl module1.pl
```

The `--IDE` option, see Section [2.4.32](#) will typically enable the `-G` option.

2.4.5 `-Ipath`: Include Search Path

Use `-I` to specify an additional directory to use when searching for header files which have been included using the `#include` directive. The `-I` option can be used more than once if multiple directories are to be searched.

The default include directory containing all standard header files are always searched even if no `-I` option is present. The default search path is searched after any user-specified directories have been searched. For example:

```
PICC --CHIP=16F877A -C -Ic:\include -Id:\myapp\include test.c
```

will search the directories `c:\include` and `d:\myapp\include` for any header files included into the source code, then search the default include directory (the include directory where the compiler was installed).

This option has no effect for files that are included into assembly source using the `INCLUDE` directive. See Section [4.3.9.3](#).

2.4.6 `-Llibrary`: Scan Library

The `-L` option is used to specify additional libraries which are to be scanned by the linker. Libraries specified using the `-L` option are scanned before the standard C library, allowing additional versions of standard library functions to be accessed.

The argument to `-L` is a library keyword to which the prefix `pic`; numbers representing the processor range, number of ROM pages and the number of RAM banks; and the suffix `.lib` are added. Thus the option `-LL` when compiling for a 16F877 will, for example, scan the library `pic42c-1.lib` and the option `-Lxx` will scan a library called `pic42c-xx.lib`. All libraries must be located in the `LIB` subdirectory of the compiler installation directory. As indicated, the argument to the `-L` option is *not* a complete library filename.

If you wish the linker to scan libraries whose names do not follow the above naming convention or whose locations are not in the `LIB` subdirectory, simply include the libraries' names on the command line along with your source files. Alternatively, the linker may be invoked directly allowing the user to manually specify all the libraries to be scanned.

2.4.7 `-L-option`: Adjust Linker Options Directly

The `-L` driver option can also be used to specify an option which will be passed directly to the linker. If `-L` is followed immediately by text starting with a *dash* character “-”, the text will be passed directly to the linker without being interpreted by PICC. For example, if the option `-L-FOO` is specified, the `-FOO` option will be passed on to the linker. The linker will then process this option, when, and if, it is invoked, and perform the appropriate function, or issue an error if the option is invalid.

Take care with command-line options. The linker cannot interpret driver options; similarly the command-line driver cannot interpret linker options. In most situations, it is always the command-line driver, PICC, that is being executed. If you need to add alternate settings in the linker tab in an MPLAB `Build options...` dialogue, these are the *driver* options (not linker options), but which are used by the driver to generate the appropriate linker options during the linking process.

The `-L` option is especially useful when linking code which contains non-standard program sections (or psects), as may be the case if the program contains C code which makes use of the `#pragma psect` directive or assembly code which contains user-defined psects. See Section 3.12.3.5 for more information. Without this `-L` option, it would be necessary to invoke the linker manually to allow the linker options to be adjusted.

One commonly used linker option is `-N`, which sorts the symbol table in the map file by address, rather than by name. This would be passed to PICC as the option `-L-N`.

This option can also be used to replace default linker options: If the string starting from the first character after the `-L` up to the first `=` character matches first part of a default linker option, then that default linker option is replaced by the option specified by the `-L`.

TUTORIAL

REPLACING DEFAULT LINKER OPTIONS In a particular project, the `psect` entry is used, but the programmer needs to ensure that this `psect` is positioned above the address 800h. This can be achieved by adjusting the default linker option that positions this `psect`. First, a map file is generated to determine how this `psect` is normally allocated memory. The `Linker command line:` in the map file indicates that this `psect` is normally linked using the linker option:

```
-pentry=CODE
```

Which places `entry` anywhere in the memory defined by the `CODE` class. The programmer then re-links the project, but now using the driver option:

```
-L-pentry=CODE+800h
```

to ensure that the `psect` is placed above 800h. Another map file is generated and the `Linker command line:` section is checked to ensure that the option was received and executed by the linker. Next, the address of the `psect entry` is noted in the `psect lists` that appear later in the map file. See Section 5.11 for more information on the contents of the map file.

If there are no characters following the first `=` character in the `-L` option, then any matching default linker option will be deleted. For example: `-L-pfirst=` will remove any default linker option that begins with the string `-pfirst=`. No warning is generated if such a default linker option cannot be found.

TUTORIAL

ADDING AND DELETING DEFAULT LINKER OPTIONS The default linker options for for a project links several psects in the following fashion.

```
-pone=600h,two,three
```

which links `one` at 600h, then follows this with `two`, then `three`. It has been decided