



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





MICROCHIP

**MPLAB[®] XC16 C Compiler
User's Guide**

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, PIC³² logo, rfPIC and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


FilterLab, Hampshire, HI-TECH C, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, chipKIT, chipKIT logo, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, HI-TIDE, In-Circuit Serial Programming, ICSP, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, Omniscient Code Generation, PICC, PICC-18, PICDEM, PICDEM.net, PCKit, PICtail, REAL ICE, rfLAB, Select Mode, Total Endurance, TSHARC, UniWinDriver, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2012, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

ISBN: 978-1-62076-467-1

**QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
= ISO/TS 16949 =**

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC[®] MCUs and dsPIC[®] DSCs, KEELOQ[®] code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Table of Contents

Preface 11

Chapter 1. Compiler Overview

 1.1 Introduction 17

 1.2 Device Description 17

 1.3 Compiler Description and Documentation 17

 1.3.1 ANSI C Standard 17

 1.3.2 Optimization 17

 1.3.3 ANSI Standard Library Support 18

 1.3.4 Flexible Memory Models 18

 1.3.5 Attributes and Qualifiers 18

 1.3.6 Compiler Driver 18

 1.3.7 Documentation 18

 1.4 Compiler and Other Development Tools 19

Chapter 2. Common C Interface

 2.1 Introduction 21

 2.2 Background – The Desire for Portable Code 21

 2.2.1 The ANSI Standard 22

 2.2.2 The Common C Interface 23

 2.3 Using the CCI 24

 2.4 ANSI Standard Refinement 25

 2.4.1 Source File Encoding 25

 2.4.2 The Prototype for main 25

 2.4.3 Header File Specification 25

 2.4.4 Include Search Paths 26

 2.4.5 The number of Significant Initial Characters in an Identifier 27

 2.4.6 Sizes of Types 27

 2.4.7 Plain char Types 28

 2.4.8 Signed Integer Representation 28

 2.4.9 Integer conversion 29

 2.4.10 Bit-wise Operations on Signed Values 29

 2.4.11 Right-shifting Signed Values 29

 2.4.12 Conversion of Union Member Accessed Using Member
 With Different Type 30

 2.4.13 Default Bit-field int Type 30

 2.4.14 Bit-fields Straddling a Storage Unit Boundary 31

 2.4.15 The Allocation Order of Bits-field 31

 2.4.16 The NULL macro 32

 2.4.17 Floating-point sizes 32

2.5 ANSI Standard Extensions	33
2.5.1 Generic Header File	33
2.5.2 Absolute addressing	33
2.5.3 Far Objects and Functions	34
2.5.4 Near Objects	35
2.5.5 Persistent Objects	36
2.5.6 X and Y Data Objects	37
2.5.7 Banked Data Objects	37
2.5.8 Alignment of Objects	38
2.5.9 EEPROM Objects	39
2.5.10 Interrupt Functions	39
2.5.11 Packing Objects	42
2.5.12 Indicating Antiquated Objects	43
2.5.13 Assigning Objects to Sections	43
2.5.14 Specifying Configuration Bits	45
2.5.15 Manifest Macros	46
2.5.16 In-line Assembly	47
2.6 Compiler Features	48
2.6.1 Enabling the CCI	48

Chapter 3. Compiler Command-Line Driver

3.1 Introduction	49
3.2 Invoking the Compiler	50
3.2.1 Drive Command-Line Format	50
3.2.2 Environment Variables	50
3.2.3 Input File Types	51
3.3 The Compilation Sequence	52
3.3.1 The Compiler Applications	52
3.3.2 Single-Step Compilation	54
3.3.3 Multi-Step Compilation	55
3.3.4 Assembly Compilation	56
3.4 Runtime Files	57
3.4.1 Library Files	57
3.4.2 Startup and Initialization	57
3.5 Compiler Output	58
3.5.1 Output Files	58
3.5.2 Diagnostic Files	58
3.6 Compiler Messages	59

3.7 Driver Option Descriptions	60
3.7.1 Options Specific to 16-Bit Devices	60
3.7.2 Options for Controlling the Kind of Output	62
3.7.3 Options for Controlling the C Dialect	63
3.7.4 Options for Controlling Warnings and Errors	64
3.7.5 Options for Debugging	70
3.7.6 Options for Controlling Optimization	71
3.7.7 Options for Controlling the Preprocessor	76
3.7.8 Options for Assembling	79
3.7.9 Options for Linking	79
3.7.10 Options for Directory Search	82
3.7.11 Options for Code Generation Conventions	82
3.8 MPLAB IDE Toolsuite Equivalents	83
Chapter 4. Device-Related Features	
4.1 Introduction	85
4.2 Device Support	85
4.3 Device Header Files	85
4.3.1 Register Definition Files	85
4.4 Stack	86
4.5 Configuration Bit Access	86
4.6 Using SFRs	87
4.7 Bit-Reversed and Modulo Addressing	89
Chapter 5. Differences Between MPLAB XC16 and ANSI C	
5.1 Divergence from the ANSI C Standard	91
5.2 Extensions to the ANSI C Standard	91
5.2.1 Keyword Differences	91
5.2.2 Expression Differences	91
5.3 Implementation-Defined Behavior	91
Chapter 6. Supported Data Types and Variables	
6.1 Introduction	93
6.2 Identifiers	93
6.3 Integer Data Types	94
6.3.1 Double-Word Integers	94
6.3.2 char Types	95
6.4 Floating-Point Data Types	95
6.5 Structures and Unions	96
6.5.1 Structure and Union Qualifiers	96
6.5.2 Bit-Fields in Structures	96
6.6 Pointer Types	98
6.6.1 Combining Type Qualifiers and Pointers	98
6.6.2 Data Pointers	99
6.6.3 Function Pointers	99
6.6.4 Special Pointer Targets	99
6.7 Complex Data Types	100
6.8 Literal Constant Types and Formats	101

6.9 Standard Type Qualifiers	103
6.9.1 Const Type Qualifier	103
6.9.2 Volatile Type Qualifier	103
6.10 Compiler-Specific type Qualifiers	104
6.10.1 __psv__ Type Qualifier	104
6.10.2 __prog__ Type Qualifier	104
6.10.3 __eds__ Type Qualifier	105
6.10.4 __pack_upper_byte Type Qualifier	105
6.10.5 __pmp__ Type Qualifier	105
6.10.6 __external__ Type Qualifier	106
6.11 Variable Attributes	107

Chapter 7. Memory Allocation and Access

7.1 Introduction	117
7.2 Address Spaces	117
7.3 Variables in Data Space Memory	118
7.3.1 Non-Auto Variable Allocation and Access	118
7.3.2 Auto Variable Allocation and Access	121
7.3.3 Changing Auto Variable Allocation	124
7.4 Variables in Program Space	125
7.4.1 Allocation and Access of Program Memory Objects	125
7.4.2 Access of objects in Program Memory	127
7.4.3 Size Limitations of Program Memory Variables	128
7.4.4 Changing Program Memory Variable Allocation	129
7.5 Parallel Master Port Access	130
7.5.1 Initialize PMP	130
7.5.2 Declare a New Memory Space	131
7.5.3 Define Variables within PMP Space	131
7.6 External Memory Access	132
7.6.1 Declare a New Memory Space	132
7.6.2 Define Variables Within an External Space	132
7.6.3 Define How to Access Memory Spaces	133
7.6.4 An External Example	135
7.7 Extended Data Space Access	136
7.8 Packing Data Stored in Flash	137
7.8.1 Packed Example	137
7.8.2 Usage Considerations	137
7.8.3 Addressing Information	138
7.9 Allocation of Variables to Registers	139
7.10 Variables in EEPROM	139
7.10.1 Accessing EEDATA via User Managed PSV	139
7.10.2 Accessing EEDATA Using TBLRDx Instructions	140
7.10.3 Accessing EEDATA Using Managed Access	141
7.10.4 Additional Sources of Information	141
7.11 Dynamic Memory Allocation	141
7.12 Memory Models	142
7.12.1 Near or Far Data	143

Chapter 8. Operators and Statements

8.1 Introduction	145
8.2 Built-In Functions	145
8.3 Integral Promotion	145

Chapter 9. Register Usage

9.1 Introduction	147
9.2 Register Variables	147
9.3 Changing Register Contents	148

Chapter 10. Functions

10.1 Introduction	149
10.2 Writing Functions	149
10.2.1 Function Specifiers	149
10.2.2 Function Attributes	149
10.3 Function Size Limits	156
10.4 Allocation of Function Code	156
10.5 Changing the Default Function Allocation	157
10.6 Inline Functions	158
10.7 Memory Models	159
10.7.1 Near or Far Code	160
10.8 Function Call Conventions	161
10.8.1 Function Parameters	161
10.8.2 Return Value	163
10.8.3 Preserving Registers Across Function Calls	163

Chapter 11. Interrupts

11.1 Introduction	165
11.2 Interrupt Operation	166
11.3 Writing an Interrupt Service Routine	166
11.3.1 Guidelines for Writing ISRs	166
11.3.2 Syntax for Writing ISRs	167
11.3.3 Coding ISRs	167
11.3.4 Using Macros to Declare Simple ISRs	168
11.4 Specifying the Interrupt Vector	169
11.4.1 Non-SMPS dsPIC30F DSCs Interrupt Vectors	170
11.4.2 SMPS dsPIC30F DSCs Interrupt Vectors	172
11.4.3 PIC24F MCUs Interrupt Vectors	174
11.4.4 dsPIC33F DSCs/PIC24H MCUs Interrupt Vectors	177
11.5 Interrupt Service Routine Context Saving	180
11.6 Nesting Interrupts	180
11.7 Enabling/Disabling Interrupts	181
11.8 ISR Considerations	182
11.8.1 Sharing Memory with Mainline Code	182
11.8.2 PSV Usage with Interrupt Service Routines	186
11.8.3 Latency	186

MPLAB[®] XC16 C Compiler User's Guide

Chapter 12. Main, Runtime Startup and Reset

12.1 Introduction	187
12.2 The main Function	187
12.3 Runtime Startup and Initialization	187

Chapter 13. Mixing C and Assembly Code

13.1 Introduction	189
13.2 Mixing Assembly Language and C Variables and Functions	189
13.3 Using Inline Assembly Language	192
13.4 Predefined Assembly Macros	196

Chapter 14. Library Routines

Chapter 15. Optimizations

15.1 Introduction	199
-------------------------	-----

Chapter 16. Preprocessing

16.1 Introduction	201
16.2 C Language Comments	201
16.3 Preprocessing Directives	201
16.4 Predefined Macro Names	202
16.4.1 Compiler Version Macro	202
16.4.2 Output Types Macros	203
16.4.3 Target Device Macros	203
16.4.4 Device Features Macros	203
16.4.5 Other Macros	204
16.5 Pragmas vs. Attributes	205

Chapter 17. Linking Programs

17.1 Introduction	207
17.2 Default Memory Spaces	207
17.3 Replacing Library Symbols	209
17.4 Linker-Defined Symbols	209
17.5 Default Linker Script	210

Appendix A. Implementation-Defined Behavior

A.1 Introduction	211
A.2 Translation	212
A.3 Environment	212
A.4 Identifiers	213
A.5 Characters	213
A.6 Integers	214
A.7 Floating Point	215
A.8 Arrays and Pointers	215
A.9 Registers	216
A.10 Structures, Unions, Enumerations and Bit Fields	216
A.11 Qualifiers	216
A.12 Declarators	216
A.13 Statements	217
A.14 Preprocessing Directives	217
A.15 Library Functions	218
A.16 Signals	219
A.17 Streams and Files	220
A.18 tmpfile	221
A.19 errno	221
A.20 Memory	221
A.21 abort	221
A.22 exit	221
A.23 getenv	221
A.24 system	221
A.25 strerror	222

Appendix B. Diagnostics

B.1 Introduction	223
B.2 Errors	223
B.3 Warnings	242

Appendix C. GNU Free Documentation License

C.1 Preamble	263
C.2 Applicability and Definitions	263
C.3 Verbatim Copying	265
C.4 Copying in Quantity	265
C.5 Modifications	266
C.6 Combining Documents	267
C.7 Collections of Documents	267
C.8 Aggregation with Independent Works	267
C.9 Translation	268
C.10 Termination	268
C.11 Future Revisions of this License	268
C.12 Relicensing	269

Appendix D. ASCII Character Set 271

MPLAB[®] XC16 C Compiler User's Guide

Appendix E. Deprecated Features

E.1 Introduction	273
E.2 Predefined Constants	273
E.3 Variables in Specified Registers	274
E.3.1 Defining Global Register Variables	274
E.3.2 Specifying Registers for Local Variables	275
E.4 Changing Non-Auto Variable Allocation	275

Appendix F. Built-in Functions

F.1 Introduction	277
F.2 Built-In Function Descriptions	279

Appendix G. XC16 Configuration Settings

G.1 Introduction	305
------------------------	-----

Glossary	307
-----------------------	------------

Index	327
--------------------	------------

Worldwide Sales and Service	338
--	------------



MPLAB[®] XC16 C COMPILER USER'S GUIDE

Preface

NOTICE TO CUSTOMERS

All documentation becomes dated, and this manual is no exception. Microchip tools and documentation are constantly evolving to meet customer needs, so some actual dialogs and/or tool descriptions may differ from those in this document.

For the most up-to-date information on development tools, see the MPLAB[®] IDE or MPLAB X IDE Help. Select the Help menu and then "Topics" or "Help Contents" to open a list of available Help files.

For the most current PDFs, please refer to our web site (<http://www.microchip.com>). Documents are identified by "DSXXXXXA", where "XXXXX" is the document number and "A" is the revision level of the document. This number is located on the bottom of each page, in front of the page number.

INTRODUCTION

MPLAB XC16 C Compiler documentation and support information is discussed in the sections below:

- Document Layout
- Conventions Used
- Recommended Reading
- myMicrochip Personalized Notification Service
- The Microchip Web Site
- Microchip Forums
- Customer Support

DOCUMENT LAYOUT


This document describes how to use GNU language tools to write code for 16-bit applications. The document layout is as follows:

- **Chapter 1. “Compiler Overview”** – describes the compiler, development tools and feature set.
- **Chapter 3. “Compiler Command-Line Driver”** – describes how to use the compiler from the command line.
- **Chapter 5. “Differences Between MPLAB XC16 and ANSI C”** – describes the differences between the C language supported by the compiler syntax and the standard ANSI-89 C.
- **Chapter 4. “Device-Related Features”** – describes the compiler header and register definition files, as well as how to use with SFRs.
- **Chapter 6. “Supported Data Types and Variables”** – describes the compiler integer, floating point and pointer data types.
- **Chapter 7. “Memory Allocation and Access”** – describes the compiler run-time model, including information on sections, initialization, memory models, the software stack and much more.
- **Chapter 8. “Operators and Statements”** – discusses operators and statements.
- **Chapter 9. “Register Usage”** – explains how to access and use SFRs.
- **Chapter 10. “Functions”** – details available functions.
- **Chapter 11. “Interrupts”** – describes how to use interrupts.
- **Chapter 12. “Main, Runtime Startup and Reset”** – describes important elements of C code.
- **Chapter 14. “Library Routines”** – explains how to use libraries.
- **Chapter 13. “Mixing C and Assembly Code”** – provides guidelines to using the compiler with 16-bit assembly language modules.
- **Chapter 15. “Optimizations”** – describes optimization options.
- **Chapter 16. “Preprocessing”** – details preprocessing operation.
- **Chapter 17. “Linking Programs”** – explains how linking works.
- **Appendix A. “Implementation-Defined Behavior”** – details compiler-specific parameters described as implementation-defined in the ANSI standard.
- **Appendix B. “Diagnostics”** – lists error and warning messages generated by the compiler.
- **Appendix C. “MPLAB XC16 vs. MPLAB XC8”** – contains the ASCII character set.
- **Appendix C. “GNU Free Documentation License”** – usage license for the Free Software Foundation.
- **Appendix E. “Deprecated Features”** – details features that are considered obsolete.
- **Appendix F. “Built-in Functions”** – lists the built-in functions of the C compiler.
- **Appendix G. “XC16 Configuration Settings”** – information about configuration settings macros.

CONVENTIONS USED

The following conventions may appear in this documentation:

DOCUMENTATION CONVENTIONS

Description	Represents	Examples
Arial font:		
Italic characters	Referenced books	<i>MPLAB[®] IDE User's Guide</i>
	Emphasized text	...is the <i>only</i> compiler...
Initial caps	A window	the Output window
	A dialog	the Settings dialog
	A menu selection	select Enable Programmer
Quotes	A field name in a window or dialog	"Save project before build"
Underlined, italic text with right angle bracket	A menu path	<u><i>File>Save</i></u>
Bold characters	A dialog button	Click OK
	A tab	Click the Power tab
Text in angle brackets < >	A key on the keyboard	Press <Enter>, <F1>
Courier font:		
Plain Courier	Sample source code	#define START
	File names	autoexec.bat
	File paths	c:\mcc18\h
	Keywords	_asm, _endasm, static
	Command-line options	-Opa+, -Opa-
	Bit values	0, 1
	Constants	0xFF, 'A'
Italic Courier	A variable argument	<i>file.c</i> , where <i>file</i> can be any valid file name
Square brackets []	Optional arguments	mpasmwin [options] <i>file</i> [options]
Curly brackets and pipe character: { }	Choice of mutually exclusive arguments; an OR selection	errorlevel {0 1}
Ellipses...	Replaces repeated text	var_name [, var_name...]
	Represents code supplied by user	void main (void) { ... }
Sidebar Text		
	Device Dependent. This feature is not supported on all devices. Devices supported will be listed in the title or text.	xmemory attribute

RECOMMENDED READING

This documentation describes how to use the MPLAB XC16 C Compiler. Other useful documents are listed below. The following Microchip documents are available and recommended as supplemental reference resources.

Release Notes (Readme Files)

For the latest information on Microchip tools, read the associated Release Notes (HTML files) included with the software.

MPLAB[®] Assembler, Linker and Utilities for PIC24 MCUs and dsPIC[®] DSCs User's Guide (DS51317)

A guide to using the 16-bit assembler, object linker, object archiver/librarian and various utilities.

16-Bit Language Tools Libraries (DS51456)

A descriptive listing of libraries available for Microchip 16-bit devices. This includes standard (including math) libraries and C compiler built-in functions. DSP and 16-bit peripheral libraries are described in Release Notes provided with each peripheral library type.

Device-Specific Documentation

The Microchip website contains many documents that describe 16-bit device functions and features. Among these are:

- Individual and family data sheets
- Family reference manuals
- Programmer's reference manuals

C Standards Information

American National Standard for Information Systems – *Programming Language – C*. American National Standards Institute (ANSI), 11 West 42nd. Street, New York, New York, 10036.

This standard specifies the form and establishes the interpretation of programs expressed in the programming language C. Its purpose is to promote portability, reliability, maintainability and efficient execution of C language programs on a variety of computing systems.

C Reference Manuals

Harbison, Samuel P. and Steele, Guy L., *C A Reference Manual*, Fourth Edition, Prentice-Hall, Englewood Cliffs, N.J. 07632.

Kernighan, Brian W. and Ritchie, Dennis M., *The C Programming Language*, Second Edition. Prentice Hall, Englewood Cliffs, N.J. 07632.

Kochan, Steven G., *Programming In ANSI C*, Revised Edition. Hayden Books, Indianapolis, Indiana 46268.

Plauger, P.J., *The Standard C Library*, Prentice-Hall, Englewood Cliffs, N.J. 07632.

Van Sickle, Ted., *Programming Microcontrollers in C*, First Edition. LLH Technology Publishing, Eagle Rock, Virginia 24085.

myMICROCHIP PERSONALIZED NOTIFICATION SERVICE

Microchip's personal notification service helps keep customers current on their Microchip products of interest. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool.

Please visit <http://www.microchip.com/pcn> to begin the registration process and select your preferences to receive personalized notifications. A FAQ and registration details are available on the page, which can be opened by selecting the link above.

When you are selecting your preferences, choosing "Development Systems" will populate the list with available development tools. The main categories of tools are listed below:

- **Compilers** – The latest information on Microchip C compilers, assemblers, linkers and other language tools. These include all MPLAB C compilers; all MPLAB assemblers (including MPASM™ assembler); all MPLAB linkers (including MPLINK™ object linker); and all MPLAB librarians (including MPLIB™ object librarian).
- **Emulators** – The latest information on Microchip in-circuit emulators. These include the MPLAB REAL ICE™ and MPLAB ICE 2000 in-circuit emulators
- **In-Circuit Debuggers** – The latest information on Microchip in-circuit debuggers. These include the MPLAB ICD 2 and 3 in-circuit debuggers and PICKit™ 2 and 3 debug express.
- **MPLAB® IDE/MPLAB X IDE** – The latest information on Microchip MPLAB IDE, the Windows® Integrated Development Environment, or MPLAB X IDE, the open source, cross-platform Integrated Development Environment. These lists focus on the IDE, Project Manager, Editor and Simulator, as well as general editing and debugging features.
- **Programmers** – The latest information on Microchip programmers. These include the device (production) programmers MPLAB REAL ICE in-circuit emulator, MPLAB ICD 3 in-circuit debugger, MPLAB PM3 and development (nonproduction) programmers MPLAB ICD 2 in-circuit debugger, PICSTART® Plus and PICKit 2 and 3.
- **Starter/Demo Boards** – These include MPLAB Starter Kit boards, PICDEM demo boards, and various other evaluation boards.

THE MICROCHIP WEB SITE

Microchip provides online support via our web site at <http://www.microchip.com>. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

MPLAB[®] XC16 C Compiler User's Guide

MICROCHIP FORUMS

Microchip provides additional online support via our web forums at <http://www.microchip.com/forums>. Currently available forums are:

- Development Tools
- 8-bit PIC MCUs
- 16-bit PIC MCUs
- 32-bit PIC MCUs

CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or field application engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document. See our web site for a complete, up-to-date listing of sales offices.

Technical support is available through the web site at <http://support.microchip.com>.

Documentation errors or comments may be emailed to docerrors@microchip.com.

DOCUMENT REVISION HISTORY

Revision A (April 2012)

Initial release of this document.

Revision B (July 2012)

- **Chapter 2. "Common C Interface"** was added.
- Figure 3-1 "Software Development Tools Data Flow" was updated.
- Table 3-16 "Linking Options" now includes the -fill option.
- Added the -pack_upper_byte qualifier information in **Section 6.10.4 "___pack_upper_byte Type Qualifier"** and **Section 7.8 "Packing Data Stored in Flash"**.
- Added DBRPAG/PSVPAG preservation bullet under **Section 10.8 "Function Call Conventions"**
- Fixed code syntax in **Section 11.4 "Specifying the Interrupt Vector"**.
- Fixed Eval Edition description under **Chapter 15. "Optimizations"**.
- Added "volatile" to SFR registers in **Appendix F. "Built-in Functions"**.
- Added built-in functions __builtin_write_CRYOTP and __builtin_write_NVM_secure in **Appendix F. "Built-in Functions"**.

Chapter 1. Compiler Overview

1.1 INTRODUCTION

The MPLAB XC16 C compiler is defined and described in the following topics:

- Device Description
- Compiler Description and Documentation
- Compiler and Other Development Tools

1.2 DEVICE DESCRIPTION

The MPLAB XC16 C compiler fully supports all Microchip 16-bit devices:

- The dsPIC[®] family of digital signal controllers combines the high performance required in digital signal processor (DSP) applications with standard microcontroller (MCU) features needed for embedded applications.
- The PIC24 family of MCUs are identical to the dsPIC DSCs with the exception that they do not have the digital signal controller module or that subset of instructions. They are a subset, and are high-performance MCUs intended for applications that do not require the power of the DSC capabilities.

1.3 COMPILER DESCRIPTION AND DOCUMENTATION

The MPLAB XC16 C compiler is a full-featured, optimizing compiler that translates standard ANSI C programs into 16-bit device assembly language source. The compiler also supports many command-line options and language extensions that allow full access to the 16-bit device hardware capabilities, and affords fine control of the compiler code generator.

The compiler is a port of the GNU Compiler Collection (GCC) compiler from the Free Software Foundation

This key features of the compiler are discussed in the following sections.

1.3.1 ANSI C Standard

The compiler is a fully validated compiler that conforms to the ANSI C standard as defined by the ANSI specification (ANSI x3.159-1989) and described in Kernighan and Ritchie's *The C Programming Language* (second edition). The ANSI standard includes extensions to the original C definition that are now standard features of the language. These extensions enhance portability and offer increased capability. In addition, language extensions for dsPIC DSC embedded-control applications are included.

1.3.2 Optimization

The compiler uses a set of sophisticated optimization passes that employ many advanced techniques for generating efficient, compact code from C source. The optimization passes include high-level optimizations that are applicable to any C code, as well as 16-bit device-specific optimizations that take advantage of the particular features of the device architecture.

For more on optimizations, see **Chapter 15. "Optimizations"**.

1.3.3 ANSI Standard Library Support

The compiler is distributed with a complete ANSI C standard library. All library functions have been validated, and conform to the ANSI C library standard. The library includes functions for string manipulation, dynamic memory allocation, data conversion, time-keeping and math functions (trigonometric, exponential and hyperbolic). The standard I/O functions for file handling are also included, and, as distributed, they support full access to the host file system using the command-line simulator. The fully functional source code for the low-level file I/O functions is provided in the compiler distribution, and may be used as a starting point for applications that require this capability.

1.3.4 Flexible Memory Models

The compiler supports both large and small code and data models. The small code model takes advantage of more efficient forms of call and branch instructions, while the small data model supports the use of compact instructions for accessing data in SFR space.

The compiler supports two models for accessing constant data. The “constants in data” model uses data memory, which is initialized by the run-time library. The “constants in code” model uses program memory, which is accessed through the Program Space Visibility (PSV) window.

1.3.5 Attributes and Qualifiers

The compiler keyword `__attribute__` allows you to specify special attributes of variables, structure fields or functions. This keyword is followed by an attribute specification inside double parentheses, as in:

```
int last_mode __attribute__ ((persistent));
```

In other compilers, qualifiers are used to create qualified types:

```
persistent int last_mode;
```

The MPLAB XC16 C Compiler does have some non-standard qualifiers described in **Section 6.10 “Compiler-Specific type Qualifiers”**.

Generally speaking, qualifiers indicate how an object should be accessed, whereas attributes indicate where objects are to be located. Attributes also have many other purposes.

1.3.6 Compiler Driver

The compiler includes a powerful command-line driver program. Using the driver program, application programs can be compiled, assembled and linked in a single step.

1.3.7 Documentation

The compiler is supported under both the MPLAB IDE v8.xx and above, and the MPLAB X IDE. For simplicity, both IDEs are referred to throughout the book as simply MPLAB IDE.

Features that are unique to specific devices, and therefore specific compilers, are noted with a “DD” icon next to the section and text that identifies the specific devices to which the information applies (see the **Preface**).

1.4 COMPILER AND OTHER DEVELOPMENT TOOLS

The compiler works with many other Microchip tools including:

- MPLAB XC16 Assembler and Linker - see the *MPLAB Assembler, Linker and Utilities for PIC24 MCUs and dsPIC DSCs User's Guide (DS51317)*.
- MPLAB IDE v8.xx and MPLAB X IDE
- The MPLAB SIM Simulator and MPLAB X Simulator
- All Microchip debug tools and programmers
- Demonstration boards and Starter kits that support 16-bit devices

MPLAB[®] XC16 C Compiler User's Guide

NOTES:

Chapter 2. Common C Interface

2.1 INTRODUCTION

The Common C Interface (CCI) is available with all MPLAB XC C compilers and is designed to enhance code portability between these compilers. For example, CCI-conforming code would make it easier to port from a PIC18 MCU using the MPLAB XC8 C compiler to a PIC24 MCU using the MPLAB XC16 C compiler.

The CCI assumes that your source code already conforms to the ANSI Standard. If you intend to use the CCI, it is your responsibility to write code that conforms. Legacy projects will need to be migrated to achieve conformance. A compiler option must also be set to ensure that the operation of the compiler is consistent with the interface when the project is built.

The following topics are examined in this chapter of the MPLAB XC16 C Compiler User's Guide:

- ANSI Standard Extensions
- Using the CCI
- ANSI Standard Refinement
- ANSI Standard Extensions

2.2 BACKGROUND – THE DESIRE FOR PORTABLE CODE

All programmers want to write portable source code.

Portability means that the same source code can be compiled and run in a different execution environment than that for which it was written. Rarely can code be one hundred percent portable, but the more tolerant it is to change, the less time and effort it takes to have it running in a new environment.

Embedded engineers typically think of code portability as being across target devices, but this is only part of the situation. The same code could be compiled for the same target but with a different compiler. Differences between those compilers might lead to the code failing at compile time or runtime, so this must be considered as well.

You may only write code for one target device and only use one brand of compiler, but if there is no regulation of the compiler's operation, simply updating your compiler version may change your code's behavior.

Code must be portable across targets, tools, and time to be truly flexible.

Clearly, this portability cannot be achieved by the programmer alone, since the compiler vendors can base their products on different technologies, implement different features and code syntax, or improve the way their product works. Many a great compiler optimization has broken many an unsuspecting project.

Standards for the C language have been developed to ensure that change is managed and code is more portable. The American National Standards Institute (ANSI) publishes standards for many disciplines, including programming languages. The ANSI C Standard is a universally adopted standard for the C programming language.

2.2.1 The ANSI Standard

The ANSI C Standard has to reconcile two opposing goals: freedom for compilers vendors to target new devices and improve code generation, with the known functional operation of source code for programmers. If both goals can be met, source code can be made portable.

The standard is implemented as a set of rules which detail not only the syntax that a conforming C program must follow, but the semantic rules by which that program will be interpreted. Thus, for a compiler to conform to the standard, it must ensure that a conforming C program functions as described by the standard.

The standard describes *implementation*, the set of tools and the runtime environment on which the code will run. If any of these change, e.g., you build for, and run on, a different target device, or if you update the version of the compiler you use to build, then you are using a different implementation.

The standard uses the term *behavior* to mean the external appearance or action of the program. It has nothing to do with how a program is encoded.

Since the standard is trying to achieve goals that could be construed as conflicting, some specifications appear somewhat vague. For example, the standard states that an `int` type must be able to hold at least a 16-bit value, but it does not go as far as saying what the size of an `int` actually is; and the action of right-shifting a signed integer can produce different results on different implementations; yet, these different results are still ANSI C compliant.

If the standard is too strict, device architectures may not allow the compiler to conform.¹ But, if it is too weak, programmers would see wildly differing results within different compilers and architectures, and the standard would lose its effectiveness.

The standard organizes source code whose behavior is not fully defined into groups that include the following behaviors:

Implementation-defined behavior

This is unspecified behavior where each implementation documents how the choice is made.

Unspecified behavior

The standard provides two or more possibilities and imposes no further requirements on which possibility is chosen in any particular instance.

Undefined behavior

This is behavior for which the standard imposes no requirements.

Code that strictly conforms to the standard does not produce output that is dependent on any unspecified, undefined, or implementation-defined behavior. The size of an `int`, which we used as an example earlier, falls into the category of behavior that is defined by implementation. That is to say, the size of an `int` is defined by which compiler is being used, how that compiler is being used, and the device that is being targeted.

All the MPLAB XC compilers conform to the ANSI X3.159-1989 Standard for programming languages (with the exception of the XC8 compiler's inability to allow recursion, as mentioned in the footnote). This is commonly called the C89 Standard. Some features from the later standard, C99, are also supported.

1. Case in point: The mid-range PIC[®] microcontrollers do not have a data stack. Because a compiler targeting this device cannot implement recursion, it (strictly speaking) cannot conform to the ANSI C Standard. This example illustrates a situation in which the standard is too strict for mid-range devices and tools.

For freestanding implementations – or for what we typically call embedded applications – the standard allows non-standard extensions to the language, but obviously does not enforce how they are specified or how they work. When working so closely to the device hardware, a programmer needs a means of specifying device setup and interrupts, as well as utilizing the often complex world of small-device memory architectures. This cannot be offered by the standard in a consistent way.

While the ANSI C Standard provides a mutual understanding for programmers and compiler vendors, programmers need to consider the implementation-defined behavior of their tools and the probability that they may need to use extensions to the C language that are non-standard. Both of these circumstances can have an impact on code portability.

2.2.2 The Common C Interface

The Common C Interface (CCI) supplements the ANSI C Standard and makes it easier for programmers to achieve consistent outcomes on all Microchip devices when using any of the MPLAB XC C compilers.

It delivers the following improvements, all designed with portability in mind.

Refinement of the ANSI C Standard

The CCI documents specific behavior for some code in which actions are implementation-defined behavior under the ANSI C Standard. For example, the result of right-shifting a signed integer is fully defined by the CCI. Note that many implementation-defined items that closely couple with device characteristics, such as the size of an `int`, are not defined by the CCI.

Consistent syntax for non-standard extensions

The CCI non-standard extensions are mostly implemented using keywords with a uniform syntax. They replace keywords, macros and attributes that are the native compiler implementation. The interpretation of the keyword may differ across each compiler, and any arguments to the keywords may be device specific.

Coding guidelines

The CCI may indicate advice on how code should be written so that it can be ported to other devices or compilers. While you may choose not to follow the advice, it will not conform to the CCI.

2.3 USING THE CCI

The CCI allows enhanced portability by refining implementation-defined behavior and standardizing the syntax for extensions to the language.

The CCI is something you choose to follow and put into effect, thus it is relevant for new projects, although you may choose to modify existing projects so they conform.

For your project to conform to the CCI, you must do the following things.

Enable the CCI

Select the MPLAB IDE widget *Use CCI Syntax* in your project, or use the command-line option that is equivalent.

Include <xc.h> in every module

Some CCI features are only enabled if this header is seen by the compiler.

Ensure ANSI compliance

Code that does not conform to the ANSI C Standard does not conform to the CCI.

Observe refinements to ANSI by the CCI

Some ANSI implementation-defined behavior is defined explicitly by the CCI.

Use the CCI extensions to the language

Use the CCI extensions rather than the native language extensions

The next sections detail specific items associated with the CCI. These items are segregated into those that refine the standard, those that deal with the ANSI C Standard extensions, and other miscellaneous compiler options and usage. Guidelines are indicated with these items.

If any implementation-defined behavior or any non-standard extension is not discussed in this document, then it is not part of the CCI. For example, GCC case ranges, label addresses and 24-bit `short long` types are not part of the CCI. Programs which use these features do not conform to the CCI. The compiler may issue a warning or error to indicate when you use a non-CCI feature and the CCI is enabled.

2.4 ANSI STANDARD REFINEMENT

The following topics describe how the CCI refines the implementation-defined behaviors outlined in the ANSI C Standard.

2.4.1 Source File Encoding

Under the CCI, a source file must be written using characters from the 7-bit ASCII set. Lines may be terminated using a *line feed* ('\n') or *carriage return* ('\r') that is immediately followed by a *line feed*. Escaped characters may be used in character constants or string literals to represent extended characters not in the basic character set.

2.4.1.1 EXAMPLE

The following shows a string constant being defined that uses escaped characters.

```
const char myName[] = "Bj\370rk\n";
```

2.4.1.2 DIFFERENCES

All compilers have used this character set.

2.4.1.3 MIGRATION TO THE CCI

No action required.

2.4.2 The Prototype for `main`

The prototype for the `main()` function is

```
int main(void);
```

2.4.2.1 EXAMPLE

The following shows an example of how `main()` might be defined

```
int main(void)
{
    while(1)
        process();
}
```

2.4.2.2 DIFFERENCES

The 8-bit compilers used a `void` return type for this function.

2.4.2.3 MIGRATION TO THE CCI

Each program has one definition for the `main()` function. Confirm the return type for `main()` in all projects previously compiled for 8-bit targets.

2.4.3 Header File Specification

Header file specifications that use directory separators do not conform to the CCI.

2.4.3.1 EXAMPLE

The following example shows two conforming include directives.

```
#include <usb_main.h>
#include "global.h"
```