Chipsmall
深圳市津利帝科技有限公司
Geelead Technology Limited
Chipsmall
采购部

CCS C Compiler Manual

PCB / PCM / PCH

October 2016

Table of Contents

Overview 1
PCB, PCM and PCH Overview 1
Installation 1
Technical Support 2
Directories 2
File Formats 3
Invoking the Command Line Compiler 4
PCW Overview 6
Menu 6
Editor Tabs 7
Slide Out Windows 7
Editor 7
Debugging Windows 7
Status Bar 8
Output Messages 8
Program Syntax 9
Overall Structure 9
Comment 9
Trigraph Sequences 11
Multiple Project Files 11
Multiple Compilation Units 12
Full Example Program 12
Statements 15
Statements 15
if 16
while 16
do-while 17
for 17
switch 18
return 18
goto 19
label 19
break 19
continue 20
expr 20
; 20
stmt 21
Expressions 23
Constants 23
Identifiers 24
Operators 24
Operator Precedence 25
Data Definitions 27
Data Definitions 27
Type Specifiers 28

Type Qualifiers 29
Enumerated Types 30
Structures and Unions 30
typedef 31
Non-RAM Data Definitions 32
Using Program Memory for Data 34
Named Registers 35
Function Definition 37
Function Definition 37
Overloaded Functions 38
Reference Parameters 38
Default Parameters 39
Variable Argument Lists 39
Functional Overview 41
I2C 41
ADC 42
Analog Comparator 43
CAN Bus 44
CCP 46
Code Profile 47
Configuration Memory 48
DAC 49
Data Eeprom 50
Data Signal Modulator 51
External Memory 52
General Purpose I/O 52
Internal LCD 54
Internal Oscillator 55
Interrupts 56
Low Voltage Detect 57
PMP/EPMP 58
Power PWM 59
Program Eeprom 61
PSP 62
QEI 63
RS232 I/O 64
RTOS 66
SPI 68
Timer0 69
Timer1 70
Timer2 71
Timer3 72
Timer4 72
Timer5 72
TimerA 73
TimerB 74
USB 75
Voltage Reference 78

WDT or Watch Dog Timer 79
interrupt_enabled() 80
Stream I/O 80
PreProcessor 83
PRE-PROCESSOR DIRECTORY 83
__address__ 83
#asm #endasm #asm asis 85
#bit 87
__buildcount__ 87
#build 88
#byte 89
#case 89
date 90
#define 90
definedinc 92
#device 92
device 95
#if expr #else #elif #endif 96
#error 96
#export (options) 97
__file__ 99
__filename__ 99
#fill_rom 99
#fuses 100
#hexcomment 101
#id 102
#ifdef #ifndef #else #elif #endif 102
#ignore_warnings 103
#import (options) 104
#include 105
#inline 106
#int_xxxx 106
#INT_DEFAULT 110
#int_global 111
__line__ 111
#list 112
#line 112
#locate 113
#module 113
#nolist 114
#ocs 115
#opt 115
#org 116
#pin_select 117
__pcb__ 120
__pcm__ 120
__pch__ 121
#pragma 121

#priority 121
#profile 122
#reserve 123
#rom 123
#separate 124
#serialize 125
#task 127
__time__ 128
#type 128
#undef 130
_unicode 130
#use capture 131
#use delay 133
#use dynamic_memory 134
#use fast_io 134
#use fixed_io 135
#use i2c 135
#use profile() 137
#use pwm() 138
#use rs232 140
#use rtos 145
#use spi 146
#use standard_io 148
#use timer 149
#use touchpad 150
#warning 152
#word 152
#zero_ram 153
Built-in Functions 155
BUILT-IN FUNCTIONS 155
abs( ) 155
sin( ) cos( ) tan( ) asin( ) acos() atan() sinh() cosh() tanh() atan2() 156
adc_done( ) 157
adc_read() 158
adc_status() 159
adc_write() 159
assert( ) 160
atoe 160
atof( ) 161
pin_select() 162
atoi( ) atol( ) atoi32( ) 163
at_clear_interrupts( ) 164
at_disable_interrupts( ) 165
at_enable_interrupts( ) 165
at_get_capture( ) 166
at_get_missing_pulse_delay( ) 167
at_get_period( ) 167
at_get_phase_counter( ) 168

at_get_resolution( ) 169
at_get_set_point( ) 169
at_get_set_point_error( )..170
at_get_status( ) 171
at_interrupt_active( ) 171
at_set_compare_time( ) 172
at_set_missing_pulse_delay( )..173
at_set_resolution( )..174
at_set_set_point( )..174
at_setup_cc( ) 175
bit_clear( ) 176
bit_set( ) 177
bit_test( ) 177
brownout_enable( ) 178
bsearch( ) 178
calloc( )..179
ceil( ) 180
clc1_setup_gate() clc2_setup_gate() clc3_setup_gate() clc4_setup_gate() 180
clear_interrupt( ) 181
clear_pwm1_interrupt( ) clear_pwm2_interrupt( ) clear_pwm3_interrupt( )
clear_pwm4_interrupt( ) clear_pwm5_interrupt( ) clear_pwm6_interrupt( ) 182
cog_status( ) 183
cog_restart( ) 183
crc_calc( ) crc_calc8( ) crc_calc16( )..184
crc_init(mode) 185
cwg_status( ) 185
cwg_restart( ) 186
dac_write( ) 186
delay_cycles( ) 187
delay_ms( ) 187
delay_us( ) 188
disable_interrupts( )..189
disable_pwm1_interrupt( ) disable_pwm2_interrupt( ) disable_pwm3_interrupt(
) disable_pwm4_interrupt( ) disable_pwm5_interrupt( ) disable_pwm6_interrupt( ) 190
div( ) ldiv( ) 191
enable_interrupts( ) 192
enable_pwm1_interrupt( ) enable_pwm2_interrupt( ) enable_pwm3_interrupt( )
enable_pwm4_interrupt( ) enable_pwm5_interrupt( ) enable_pwm6_interrupt( ) 193
erase_eeprom( )...194
erase_program_eeprom( ) 194
exp( ) 195
ext_int_edge( ) 196
fabs( ) 196
floor( ) 197
fmod( ) 197
free( )..198
frexp( ) 199
scanf( ) 199

get_capture( ) 202
get_capture_event() 203
get_capture_time() 203
get_capture32() 204
get_hspwm_capture( ) 205
get_nco_accumulator( ) 205
get_nco_inc_value( ) 206
get_ticks( ) 206
get_timerA( ) 207
get_timerB( ) 207
get_timerx( ) 208
get_tris_x( ) 209
getc( ) getch( ) getchar( ) fgetc( ) 209
getenv( ) 210
gets( ) fgets( ) 215
goto_address( ) 216
high_speed_adc_done( ) 216
i2c_init( ) 217
i2c_isr_state( ) 218
i2c_poll( ) 219
i2c_read( ) 219
i2c_slaveaddr( ) 220
i2c_speed( ) 221
i2c_start( ) 221
i2c_stop( ) 222
i2c_write( ) 223
input( ) 224
input_change_x( ) 225
input_state( ) 225
input_x( ) 226
interrupt_active( ) 227
isalnum(char) isalpha(char) iscntrl(x) isdigit(char) isgraph(x) islower(char) isspace(char) isupper(char) isxdigit(char) isprint(x) ispunct(x) 227
isamong( ) 229
itoa( ) 229
jump_to_isr( ) 230
kbhit( ) 231
label_address( ) 232
labs( ) 232
lcd_contrast( ) 233
lcd_load( ) 233
lcd_symbol( ) 234
ldexp( ) 235
log( ) 235
log10( ) 236
longjmp( ) 237
make8( ) 237
make16( ) 238

make32( ) 238
malloc( ) 239
memcpy( ) memmove( ) 240
memset( ) 240
modf( ) 241
_mul( ) 242
nargs( ) 242
offsetof( ) offsetofbit( ) 243
output_x( ) 244
output_bit( ) 245
output_drive( ) 246
output_float( ) 246
output_high( ) 247
output_low( ) 248
output_toggle( ) 249
perror( ) 249
pid_busy( ) 250
pid_get_result( ) 250
pid_read( ) 251
pid_write( ) 252
pll_locked( ) 253
port_x_pullups ( ) 254
pow( ) pwr( ) 255
printf( ) fprintf( ) 255
profileout() 257
psmc_blanking( ) 258
psmc_deadband( ) 260
psmc_duty( ) 260
psmc_freq_adjust( ) 261
psmc_modulation( ) 262
psmc_pins( ) 263
psmc_shutdown( ) 264
psmc_sync( ) 266
psp_output_full( ) psp_input_full( ) psp_overflow( ) 267
putc( ) putchar( ) fputc( ) 267
putc_send( ); fputc_send( ); 268
puts( ) fputs( ) 269
pwm_off() 270
pwm_on() 270
pwm_set_duty() 271
pwm_set_duty_percent 272
pwm_set_frequency 272
pwm1_interrupt_active( ) pwm2_interrupt_active( ) pwm3_interrupt_active( )
pwm4_interrupt_active( ) pwm5_interrupt_active( ) pwm6_interrupt_active( ) 273
qei_get_count( ) 274
qei_set_count( ) 274
qei_status( ) 275
qsort( ) 275

rand( )....276
rcv_buffer_bytes( ) 277
rcv_buffer_full( ) 277
read_adc( )....278
read_bank( )....279
read_calibration( ) 280
read_configuration_memory( ) 281
read_eeprom( ) 281
read_extended_ram( )....282
read_program_memory( ) read_external_memory( ) 282
read_program_eeprom( ) 283
read_rom_memory( )....284
read_sd_adc( ) 284
realloc( ) 285
release_io() 286
reset_cpu( ) 286
restart_cause( ) 287
restart_wdt( ) 287
rotate_left( )....288
rotate_right( )....289
rtc_alarm_read( )....290
rtc_alarm_write( ) 290
rtc_read( )....291
rtc_write( ) 291
rtos_await( ) 292
rtos_disable( ) 293
rtos_enable( ) 293
rtos_msg_poll( ) 294
rtos_overrun( )....294
rtos_run( )....295
rtos_signal( ) 295
rtos_stats( ) 296
rtos_terminate( ) 297
rtos_wait( ) 297
rtos_yield( ) 298
set_adc_channel( )....299
set_adc_trigger() 299
set_analog_pins( )....300
scanf( ) 301
set_cog_blanking( )....304
set_cog_dead_band( ) 305
set_cog_phase( ) 305
set_compare_time( ) 306
set_dedicated_adc_channel( ) 307
set_input_level_x() 308
set_nco_inc_value( ) 308
set_power_pwm_override( )....310
set_power_pwmx_duty( ) 310

set_pwm1_duty( ) set_pwm2_duty( ) set_pwm3_duty( ) set_pwm4_duty( ) set_pwm5_duty( ) 311
set_pwm1_offset( ) set_pwm2_offset( ) set_pwm3_offset( ) set_pwm4_offset( ) set_pwm5_offset( ) set_pwm6_offset( ) 312
set_pwm1_period( ) set_pwm2_period( ) set_pwm3_period( ) set_pwm4_period( ) set_pwm5_period( ) set_pwm6_period( ) 313
set_pwm1_phase( ) set_pwm2_phase( ) set_pwm3_phase( ) set_pwm4_phase( ) set_pwm5_phase( ) set_pwm6_phase( ) 314
set_open_drain_x() 315
set_rtcc( ) set_timer0( ) set_timer1( ) set_timer2( ) set_timer3( ) set_timer4( ) set_timer5( ) 315
set_ticks( ) 316
setup_sd_adc_calibration( ) 317
set_sd_adc_channel( ) 318
set_timerA( ) 318
set_timerB( ) 319
set_tris_x( ) 320
set_uart_speed( ) 320
setjmp( ) 321
setup_adc(mode) 322
setup_adc_ports( ) 323
setup_adc_reference( ) 324
setup_at( ) 325
setup_ccp1( ) setup_ccp2( ) setup_ccp3( ) setup_ccp4( ) setup_ccp5( ) setup_ccp6( ) 326
setup_clc1() setup_clc2() setup_clc3() setup_clc4() 328
setup_comparator( ) 329
setup_counters( ) 329
setup_cog( ) 330
setup_crc( ) 332
setup_cwg( ) 332
setup_dac( ) 333
setup_external_memory( ) 334
setup_lcd( ) 334
setup_low_volt_detect( ) 336
setup_nco( ) 336
setup_opamp1( ) setup_opamp2( ) setup_opamp3() 337
setup_pid( ) 338
setup_pmp(option,address_mask) 339
setup_psmc( ) 340
setup_power_pwm( ) 342
setup_power_pwm_pins( ) 343
setup_psp(option,address_mask) 344
setup_pwm1( ) setup_pwm2( ) setup_pwm3( ) setup_pwm4( ) 345
setup_qei( ) 345
setup_rtc( ) 346
setup_rtc_alarm( ) 347
setup_smtx( ) 347

setup_spi( ) setup_spi2( ) 348
setup_timer_A( ) 349
setup_timer_B( ) 349
setup_timer_0( ) 350
setup_timer_1( ) 351
setup_timer_2( ) 351
setup_timer_3( ) 352
setup_timer_4( ) 353
setup_timer_5( ) 353
setup_uart( ) 354
setup_vref( ) 355
setup_wdt( ) 356
setup_zdc( ) 356
shift_left( ) 357
shift_right( ) 357
sleep( ) 358
smtx_read( ) 360
smtx_reset_timer( ) 360
smtx_start( ) 361
smtx_status( ) 361
smtx_stop( ) 362
smtx_write( ) 362
smtx_update( ) 363
spi_data_is_in( ) spi_data_is_in2( ) 363
spi_init() 364
spi_prewrite(data); 365
spi_read( ) spi_read2( ) 365
spi_read2_16() spi_read3_16() spi_read4_16() 366
spi_speed 367
spi_write( ) spi_write2( ) 367
spi_xfer( ) 368
SPI_XFER_IN() 369
sprintf( ) 369
sqrt( ) 370
srand( ) 371
STANDARD STRING FUNCTIONS( ) memchr( ) memcmp( ) strcat( )
strchr( ) strcmp( ) strcoll( ) strcspn( ) strerror( ) stricmp( ) strlen( ) strlwr( )
strncat( ) strncmp( ) strncpy( ) strpbrk( ) strrchr( ) strspn( ) strstr( ) strxfrm( ) 372
strcpy( ) strcopy( ) 373
strtod( ) 374
strtok( ) 375
strtol( ) 376
strtoul( ) 376
swap( ) 377
tolower( ) toupper( ) 378
touchpad_getc( ) 379
touchpad_hit( ) 379
touchpad_state( ) 380

tx_buffer_available()....381
tx_buffer_bytes()....382
tx_buffer_full( )....383
va_arg( )....383
va_end( )....384
va_start....385
write_bank( )....386
write_configuration_memory( )....386
write_eeprom( )....387
write_external_memory( )....388
write_extended_ram( )....389
write_program_eeprom( )....389
write_program_memory( )....390
zdc_status( )....391
Standard C Include Files....393
errno.h....393
float.h....393
limits.h....394
locale.h....395
setjmp.h....395
stddef.h....395
stdio.h....395
stdlib.h....396
Software License Agreement....397

OVERVIEW

PCB, PCM and PCH Overview

The PCB, PCM, and PCH are separate compilers. PCB is for 12-bit opcodes, PCM is for 14-bit opcodes, and PCH is for 16-bit opcode PIC® microcontrollers. Due to many similarities, all three compilers are covered in this reference manual. Features and limitations that apply to only specific microcontrollers are indicated within. These compilers are specifically designed to meet the unique needs of the PIC® microcontroller. This allows developers to quickly design applications software in a more readable, high-level language.

IDE Compilers (PCW, PCWH and PCWHD) have the exclusive C Aware integrated development environment for compiling, analyzing and debugging in real-time. Other features and integrated tools can be viewed in the help file.

When compared to a more traditional C compiler, PCB, PCM, and PCH have some limitations. As an example of the limitations, function recursion is not allowed. This is due to the fact that the PIC® has no stack to push variables onto, and also because of the way the compilers optimize the code. The compilers can efficiently implement normal C constructs, input/output operations, and bit twiddling operations. All normal C data types are supported along with pointers to constant arrays, fixed point decimal, and arrays of bits.

Installation

1. Insert the CD ROM, select each of the programs you wish to install and follow the on-screen instructions.

2. If the CD does not auto start run the setup program in the root directory.

3. For help answering the version questions see the "Directories" Help topic.

4. Key Questions that may come up:

   - Keep Settings- Unless you are having trouble select this
   - Link Compiler Extensions- If you select this the file extensions like .c will start the compiler IDE when you double click on files with that extension. .hex files start

the CCSLOAD program. This selection can be change in the IDE.

- Install MP LAB Plug In- If you plan to use MPLAB and you don't select this you will need to download and manually install the Plug-In.
- Install ICD2, ICD3...drivers-select if you use these microchip ICD units.
- Delete Demo Files- Always a good idea
- Install WIN8 APP- Allows you to start the IDE from the WIN8 Start Menu.

Technical Support

Compiler, software, and driver updates are available to download at:
http://www.ccsinfo.com/download

Compilers come with 30 or 60 days of download rights with the initial purchase. One year maintenance plans may be purchased for access to updates as released.

The intent of new releases is to provide up-to-date support with greater ease of use and minimal, if any, transition difficulty.

To ensure any problem that may occur is corrected quickly and diligently, it is recommended to send an email to: support@ccsinfo.com or use the Technical Support Wizard in PCW. Include the version of the compiler, an outline of the problem and attach any files with the email request. CCS strives to answer technical support timely and thoroughly.

Technical Support is available by phone during business hours for urgent needs or if email responses are not adequate. Please call 262-522-6500 x32.

Directories

The compiler will search the following directories for Include files.

- Directories listed on the command line
- Directories specified in the .CCSPJT file
- The same directory as the source.directories in the ccsc.ini file

By default, the compiler files are put in C:\Program Files\PICC and the example programs are in \PICC\EXAMPLES. The include files are in PICC\drivers. The device header files are in PICC\devices.

The compiler itself is a DLL file. The DLL files are in a DLL directory by default in \PICC\DLL.

It is sometimes helpful to maintain multiple compiler versions. For example, a project was tested with a specific version, but newer projects use a newer version. When installing the compiler you are prompted for what version to keep on the PC. IDE users can change versions using Help>about and clicking "other versions." Command Line users use start>all programs>PIC-C>compiler version.

Two directories are used outside the PICC tree. Both can be reached with start>all programs>PIC-C.

1.) A project directory as a default location for your projects. By default put in "My
Documents." This is a good place for VISTA and up.

2.) User configuration settings and PCWH loaded files are kept in %APPDATA%\PICC

File Formats

<table>
<tr><td><b>.c</b></td><td><b>This is the source file containing user C source code.</b></td></tr>
<tr><td><b>.h</b></td><td>These are standard or custom header files used to define pins, register, register bits, functions and preprocessor directives.</td></tr>
<tr><td><b>.pjt</b></td><td>This is the older pre- Version 5 project file which contains information related to the project.</td></tr>
<tr><td><b>.ccspjt</b></td><td>This is the project file which contains information related to the project.</td></tr>
<tr><td><b>.lst</b></td><td>This is the listing file which shows each C source line and the associated assembly code generated for that line.
The elements in the .LST file may be selected in PCW under Options>Project>Output Files
<table>
<tr><th>CCS Basic</th><th>Standard assembly instructions</th></tr>
<tr><td><b>with Opcodes</b></td><td>Includes the HEX opcode for each instruction</td></tr>
<tr><td><b>Old Standard</b></td><td></td></tr>
<tr><td><b>Symbolic</b></td><td>Shows variable names instead of addresses</td></tr>
</table></td></tr>
<tr><td><b>.sym</b></td><td>This is the symbol map which shows each register location and what program variables are stored in each location.</td></tr>
<tr><td><b>.sta</b></td><td>The statistics file shows the RAM, ROM, and STACK usage. It provides information on the source codes structural and textual complexities using Halstead and McCabe metrics.</td></tr>
</table>

.tre	The tree file shows the call tree. It details each function and what functions it calls along with the ROM and RAM usage for each function.
.hex	The compiler generates standard HEX files that are compatible with all programmers. The compiler can output 8-bet hex, 16-bit hex, and binary files.
.cof	This is a binary containing machine code and debugging information. The debug files may be output as Microchip .COD file for MPLAB 1-5, Advanced Transdata .MAP file, expanded .COD file for CCS debugging or MPLAB 6 and up .xx .COF file. All file formats and extensions may be selected via Options File Associations option in Windows IDE.
.cod	This is a binary file containing debug information.
.rtf	The output of the Documentation Generator is exported in a Rich Text File format which can be viewed using the RTF editor or Wordpad.
.rvf	The Rich View Format is used by the RTF Editor within the IDE to view the Rich Text File.
.dgr	The .DGR file is the output of the flowchart maker.
.esym **.xsym**	These files are generated for the IDE users. The file contains Identifiers and Comment information. This data can be used for automatic documentation generation and for the IDE helpers.
.o	Relocatable object file
.osym	This file is generated when the compiler is set to export a relocatable object file. This file is a .sym file for just the one unit.
.err	Compiler error file
.ccsload	used to link Windows 8 apps to CCSLoad
.ccssiow	used to link Windows 8 apps to Serial Port Monitor

Invoking the Command Line Compiler

The command line compiler is invoked with the following command:

```
CCSC   [options]   [cfilename]
```

Valid options:

+FB	**Select PCB (12 bit)**	**-D**	**Do not create debug file**
+FM	Select PCM (14 bit)	+DS	Standard .COD format debug file

+FH	Select PCH (PIC18XXX)	+DM	.MAP format debug file
+Yx	Optimization level x (0-9)	+DC	Expanded .COD format debug file
		+DF	Enables the output of an COFF debug file.
+FS	Select SXC (SX)	+EO	Old error file format
+ES	Standard error file	-T	Do not generate a tree file
+T	Create call tree (.TRE)	-A	Do not create stats file (.STA)
+A	Create stats file (.STA)	-EW	Suppress warnings (use with +EA)
+EW	Show warning messages	-E	Only show first error
+EA	Show all error messages and all warnings	+EX	Error/warning message format uses GCC's "brief format" (compatible with GCC editor environments)

The xxx in the following are optional. If included it sets the file extension:

+LNxxx	**Normal list file**	**+O8xxx**	**8-bit Intel HEX output file**
+LSxxx	MPASM format list file	+OWxxx	16-bit Intel HEX output file
+LOxxx	Old MPASM list file	+OBxxx	Binary output file
+LYxxx	Symbolic list file	-O	Do not create object file
-L	Do not create list file		
+P	Keep compile status window up after compile		
+Pxx	Keep status window up for xx seconds after compile		
+PN	Keep status window up only if there are no errors		
+PE	Keep status window up only if there are errors		
+Z	Keep scratch files on disk after compile		
+DF	COFF Debug file		
I+="..."	Same as I="..." Except the path list is appended to the current list		
I="..."	Set include directory search path, for example: I="c:\picc\examples;c:\picc\myincludes" If no I= appears on the command line the .PJT file will be used to supply the include file paths.		
-P	Close compile window after compile is complete		
+M	Generate a symbol file (.SYM)		
-M	Do not create symbol file		
+J	Create a project file (.PJT)		
-J	Do not create PJT file		
+ICD	Compile for use with an ICD		
#xxx="yyy"	Set a global #define for id xxx with a value of yyy, example: #debug="true"		
+Gxxx="yyy"	Same as #xxx="yyy"		
+?	Brings up a help file		
-?	Same as +?		

+STDOUT	Outputs errors to STDOUT (for use with third party editors)
+SETUP	Install CCSC into MPLAB (no compile is done)
sourceline=	Allows a source line to be injected at the start of the source file. Example: CCSC +FM myfile.c sourceline="#include <16F887.h>"
+V	Show compiler version (no compile is done)
+Q	Show all valid devices in database (no compile is done)

A / character may be used in place of a + character. The default options are as follows:
+FM +ES +J +DC +Y9 -T -A +M +LNlst +O8hex -P -Z

If @filename appears on the CCSC command line, command line options will be read from the specified file. Parameters may appear on multiple lines in the file.

If the file CCSC.INI exists in the same directory as CCSC.EXE, then command line parameters are read from that file before they are processed on the command line.

Examples:

```
CCSC +FM C:\PICSTUFF\TEST.C
 CCSC +FM +P +T TEST.C
```

PCW Overview

The PCW IDE provides the user an easy to use editor and environment for developing microcontroller applications. The IDE comprises of many components, which are summarized below. For more information and details, use the Help>PCW in the compiler..

Many of these windows can be re-arranged and docked into different positions.

Menu

All of the IDE's functions are on the main menu. The main menu is divided into separate sections, click on a section title ('Edit', 'Search', etc) to change the section. Double clicking on the section, or clicking on the chevron on the right, will cause the menu to minimize and take less space.

Editor Tabs

All of the open files are listed here. The active file, which is the file currently being edited, is given a different highlight than the other files. Clicking on the X on the right closes the active file. Right clicking on a tab gives a menu of useful actions for that file.

Slide Out Windows

'Files' shows all the active files in the current project. 'Projects' shows all the recent projects worked on. 'Identifiers' shows all the variables, definitions, prototypes and identifiers in your current project.

Editor

The editor is the main work area of the IDE and the place where the user enters and edits source code. Right clicking in this area gives a menu of useful actions for the code being edited.

Debugging Windows

Debugger control is done in the debugging windows. These windows allow you set breakpoints, single step, watch variables and more.

Status Bar

The status bar gives the user helpful information like the cursor position, project open and file being edited.

Output Messages

Output messages are displayed here. This includes messages from the compiler during a build, messages from the programmer tool during programming or the results from find and searching.

PROGRAM SYNTAX

Overall Structure

A program is made up of the following four elements in a file:
Comment
Pre-Processor Directive
Data Definition
Function Definition
  Statements
  Expressions

Every C program must contain a main function which is the starting point of the program execution. The program can be split into multiple functions according to the their purpose and the functions could be called from main or the sub-functions. In a large project functions can also be placed in different C files or header files that can be included in the main C file to group the related functions by their category. CCS C also requires to include the appropriate device file using #include directive to include the device specific functionality. There are also some preprocessor directives like #fuses to specify the fuses for the chip and #use delay to specify the clock speed. The functions contain the data declarations,definitions,statements and expressions. The compiler also provides a large number of standard C libraries as well as other device drivers that can be included and used in the programs. CCS also provides a large number of built-in functions to access the various peripherals included in the PIC microcontroller.

Comment

Comments – Standard Comments
A comment may appear anywhere within a file except within a quoted string. Characters between /* and */ are ignored. Characters after a // up to the end of the line are ignored.

Comments for Documentation Generator
The compiler recognizes comments in the source code based on certain markups. The compiler recognizes these special types of comments that can be later exported for use in the documentation generator. The documentation generator utility uses a user selectable template to export these comments and create a formatted output document in Rich Text File Format. This utility is only available in the IDE version of the compiler. The source code markups are as follows.

Global Comments

These are named comments that appear at the top of your source code. The comment names are case sensitive and they must match the case used in the documentation template.
For example:

```
//*PURPOSE This program implements a Bootloader.
//*AUTHOR John Doe
```

A '//' followed by an * will tell the compiler that the keyword which follows it will be the named comment. The actual comment that follows it will be exported as a paragraph to the documentation generator.
Multiple line comments can be specified by adding a : after the *, so the compiler will not concatenate the comments that follow. For example:

```
/**:CHANGES
     05/16/06   Added PWM loop
     05/27.06   Fixed Flashing problem
*/
```

Variable Comments

A variable comment is a comment that appears immediately after a variable declaration.
For example:

```
int seconds; // Number of seconds since last entry
long day,     // Current day of the month,   /* Current Month */
long year;      // Year
```

Function Comments

A function comment is a comment that appears just before a function declaration. For example:

```
// The following function initializes outputs
void function_foo()
{
        init_outputs();
}
```

Function Named Comments

The named comments can be used for functions in a similar manner to the Global Comments. These comments appear before the function, and the names are exported as-is to the documentation generator.

For example:

```
//*PURPOSE This function displays data in BCD format
void display_BCD( byte n)
{
	display_routine();
 }
```

Trigraph Sequences

The compiler accepts three character sequences instead of some special characters not available on all keyboards as follows:

Sequence	Same as
??=	#
??(	[
??/	\
??)	]
??'	^
??<	{
??!	|
??>	}
??-	~

Multiple Project Files

When there are multiple files in a project they can all be included using the #include in the main file or the sub-files to use the automatic linker included in the compiler. All the header files, standard libraries and driver files can be included using this method to automatically link them.

For example: if you have main.c, x.c, x.h, y.c,y.h and z.c and z.h files in your project, you can say in:

main.c	**#include <device header file>**
	#include<x.c>
	#include<y.c>
	#include <z.c>

x.c	#include <x.h>
y.c	#include <y.h>
z.c	#include <z.h>

In this example there are 8 files and one compilation unit. Main.c is the only file compiled.

Note that the #module directive can be used in any include file to limit the visibility of the symbol in that file.

To separately compile your files see the section "multiple compilation units".

Multiple Compilation Units

Multiple Compilation Units are only supported in the IDE compilers, PCW, PCWH, PCHWD and PCDIDE.   When using multiple compilation units, care must be given that pre-processor commands that control the compilation are compatible across all units. It is recommended that directives such as #FUSES, #USE and the device header file all put in an include file included by all units. When a unit is compiled it will output a relocatable object file (*.o) and symbol file (*.osym).

There are several ways to accomplish this with the CCS C Compiler. All of these methods and example projects are included in the MCU.zip in the examples directory of the compiler.

Full Example Program

Here is a sample program with explanation using CCS C to read adc samples over rs232:

```
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
/////                       EX_ADMM.C                               ////
////                                                                ////
////  This program displays the min and max of 30 A/D samples over  ////
////  the RS-232 interface.  The process is repeated forever.       ////
////                                                                ////
```

```
////  If required configure the CCS prototype card as follows:       ////
////     Insert jumper from output of POT to pin A5                  ////
////     Use a 10K POT to vary the voltage.                          ////
////                                                                 ////
////  Jumpers:                                                       ////
////     PCM,PCH    pin C7 to RS232 RX, pin C6 to RS232 TX           ////
////     PCD        none                                             ////
////                                                                 ////
////  This example will work with the PCM, PCH, and PCD compilers.   ////
////  The following conditional compilation lines are used to        ////
////  include a valid device for each compiler.  Change the device,  ////
////  clock and RS232 pins for your hardware if needed.              ////
/////////////////////////////////////////////////////////////////////////
////        (C) Copyright 1996,2007 Custom Computer Services         ////
//// This source code may only be used by licensed users of the CCS  ////
//// C compiler.  This source code may only be distributed to other  ////
//// licensed users of the CCS C compiler.  No other use,            ////
//// reproduction or distribution is permitted without written       ////
//// permission.  Derivative programs created using this software    ////
//// in object code form are not restricted in any way.              ////
/////////////////////////////////////////////////////////////////////////
#if defined(__PCM__)                           // Preprocessor directive
that chooses
                                                    // the compiler
#include <16F877.h>                           // Preprocessor directive
that selects
                                                    // the chip
#fuses HS,NOWDT,NOPROTECT,NOLVP                // Preprocessor directive
that defines
                                                    // the chip fuses
#use delay(clock=20000000)                     // Preprocessor directive
that                                                                //
specifies clock speed

#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7) // Preprocessor directive
that includes
                                                    // RS232 libraries

#elif defined(__PCH__)
#include <18F452.h>
#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)
#fuses HS,NOWDT
#device ADC=8
#use delay(clock=20000000)
#use rs232(baud=9600, UART1A)
#endif

void main() {
   unsigned int8 i, value, min, max;
   printf("Sampling:");                        // Printf function included
in RS232
                                                    // library
   setup_adc_ports(AN0);
```