MPLAB X IDE User's Guide

MPLAB® X IDE User's Guide

Notice to Customers

Important:
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. Please refer to our website (www.microchip.com) to obtain the latest documentation available.

Documents are identified with a "DS" number. This number is located on the bottom of each page, in front of the page number. The numbering convention for the DS number is "DSXXXXXA", where "XXXXX" is the document number and "A" is the alphabetic revision level of the document.

For the most up-to-date information on development tools, see the MPLAB® IDE online help. Select the Help menu, and then Help Content to open a list of available online help files.

Table of Contents

1. What is MPLAB X IDE?

MPLAB® X IDE IS A SOFTWARE PROGRAM THAT IS USED TO DEVELOP APPLICATIONS FOR MICROCHIP MICROCONTROLLERS AND DIGITAL SIGNAL CONTROLLERS.

This development tool is called an Integrated Development Environment, or IDE, because it provides a single integrated "environment" to develop code for embedded microcontrollers. MPLAB X IDE incorporates powerful tools to help you discover, configure, develop, debug and qualify your embedded designs. MPLAB X IDE works seamlessly with the MPLAB development ecosystem of software and tools, many of which are completely free.

1.1 An Overview of Embedded Systems

An embedded system is typically a design that uses the power of a small microcontroller, like the Microchip PIC® or AVR® microcontroller (MCU). These microcontrollers combine a microprocessor unit (like the CPU in a personal computer) with some additional circuits called peripherals, plus some additional circuits, on the same chip to make a small control module requiring few other external devices. This single device can then be embedded into other electronic and mechanical devices for low-cost digital control.

1.1.1 Differences Between an Embedded Controller and a Personal Computer

The main difference between an embedded controller and a personal computer is that the embedded controller is dedicated to one specific task or set of tasks. A personal computer is designed to run many different types of programs and to connect to many different external devices. An embedded controller has a single program and, as a result, can be made cheaply to include just enough computing power and hardware to perform that dedicated task.

A personal computer has a relatively expensive generalized central processing unit (CPU) at its heart with many other external devices (memory, disk drives, video controllers, network interface circuits, etc.). An embedded system has a low-cost MCU for its intelligence, has many peripheral circuits on the same chip, and has relatively few external devices.

Often, an embedded system is an invisible part, or sub-module of another product, such as a cordless drill, refrigerator or garage door opener. The controller in these products does a tiny portion of the function of the whole device. The controller adds low-cost intelligence to some of the critical sub-systems in these devices.

An example of an embedded system is a smoke detector. Its function is to evaluate signals from a sensor and sound an alarm if the signals indicate the presence of smoke. A small program in the smoke detector either runs in an infinite loop, sampling the signal from the smoke sensor, or lies dormant in a low-power "Sleep" mode, being awakened by a signal from the sensor. The program then sounds the alarm. The program would possibly have a few other functions, such as a user test function, and a low battery alert.

While a personal computer with a sensor and audio output could be programmed to do the same function, it would not be a cost-effective solution (nor would it run on a nine-volt battery, unattended for years). Embedded designs use inexpensive microcontrollers to put intelligence into the everyday things in our environment, such as smoke detectors, cameras, cell phones, appliances, automobiles, smart cards and security systems.

1.1.2 Components of a Microcontroller

The microcontroller (MCU) has on-chip program memory (Figure 1-1 or Figure 1-2) for the firmware, or coded instructions, to run a program (Figure 1-3 or Figure 1-4). A Program Counter (PC) is used to address program memory, including Reset and interrupt addresses. A hardware stack is used with call and return instructions in code, so it works with, but is not part of, program memory. Device data sheets describe the details of program memory operation, vectors and the stack.

Figure 1-1. PIC® MCU Data Sheet - Program Memory and Stack

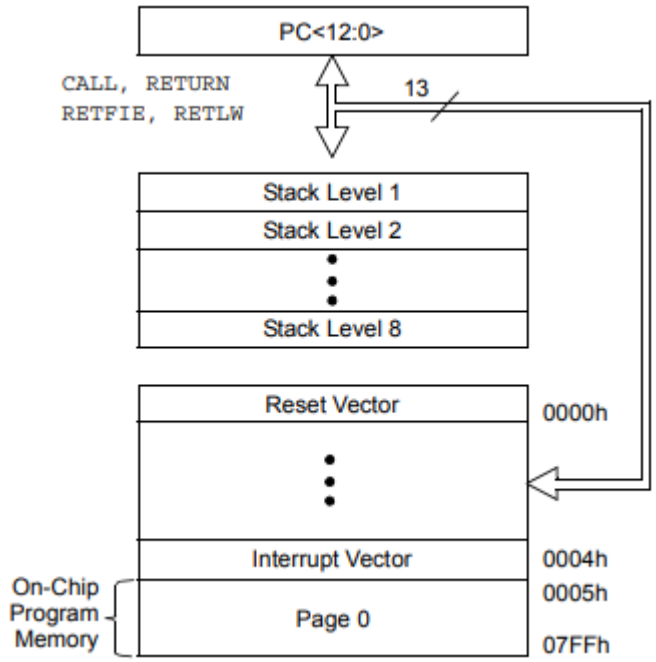


Figure 1-2. AVR® MCU Data Sheet - Program Memory

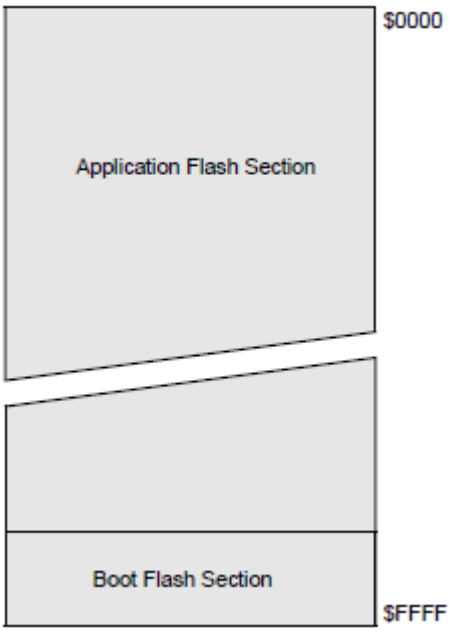


Figure 1-3. PIC® MCU Data Sheet – Instruction Set (Excerpt)

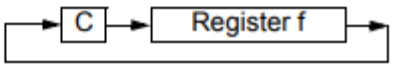
RRF	**Rotate Right f through Carry**
Syntax:	[*label*] RRF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	See description below
Status Affected:	C
Description:	The contents of register 'f' are rotated one bit to the right through the Carry flag. If 'd' is '0', the result is placed in the W register. If 'd' is '1', the result is placed back in register 'f'. 

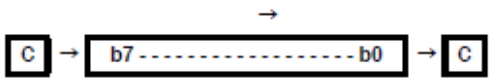
Figure 1-4. AVR® MCU Instruction Set (Excerpt)

ROR – Rotate Right through Carry

Description:

Shifts all bits in Rd one place to the right. The C Flag is shifted into bit 7 of Rd. Bit 0 is shifted into the C Flag. This operation, combined with ASR, effectively divides multi-byte signed values by two. Combined with LSR it effectively divides multi-byte unsigned values by two. The Carry Flag can be used to round the result.

Operation:



	Syntax:	Operands:	Program Counter:
(i)	ROR Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0111

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S: N ⊕ V, For signed tests.

V: N ⊕ C (For N and C after the shift)

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: Rd0
Set if, before the shift, the LSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
            lsr   r19          ; Divide r19:r18 by two
            ror   r18          ; r19:r18 is an unsigned two-byte integer
            brcc  zeroenc1     ; Branch if carry cleared
            asr   r17          ; Divide r17:r16 by two
            ror   r16          ; r17:r16 is a signed two-byte integer
            brcc  zeroenc2     ; Branch if carry cleared
            ...
zeroenc1:   nop                ; Branch destination (do nothing)
            ...

zeroenc1:   nop                ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1

The microcontroller also has data, or File Register, memory. This memory consists of Special Function Registers (SFRs) and General Purpose Registers (GPRs). SFRs are registers used by the CPU and peripheral functions for controlling the desired operation of the device. GPRs are for storage of variables that the program will need for computation or temporary storage. Some microcontrollers have additional data EEPROM memory. As with program memory, device data sheets describe the details of data memory use and operation.

Table 1-1. PIC® MCU Data Sheet – File Registers Example

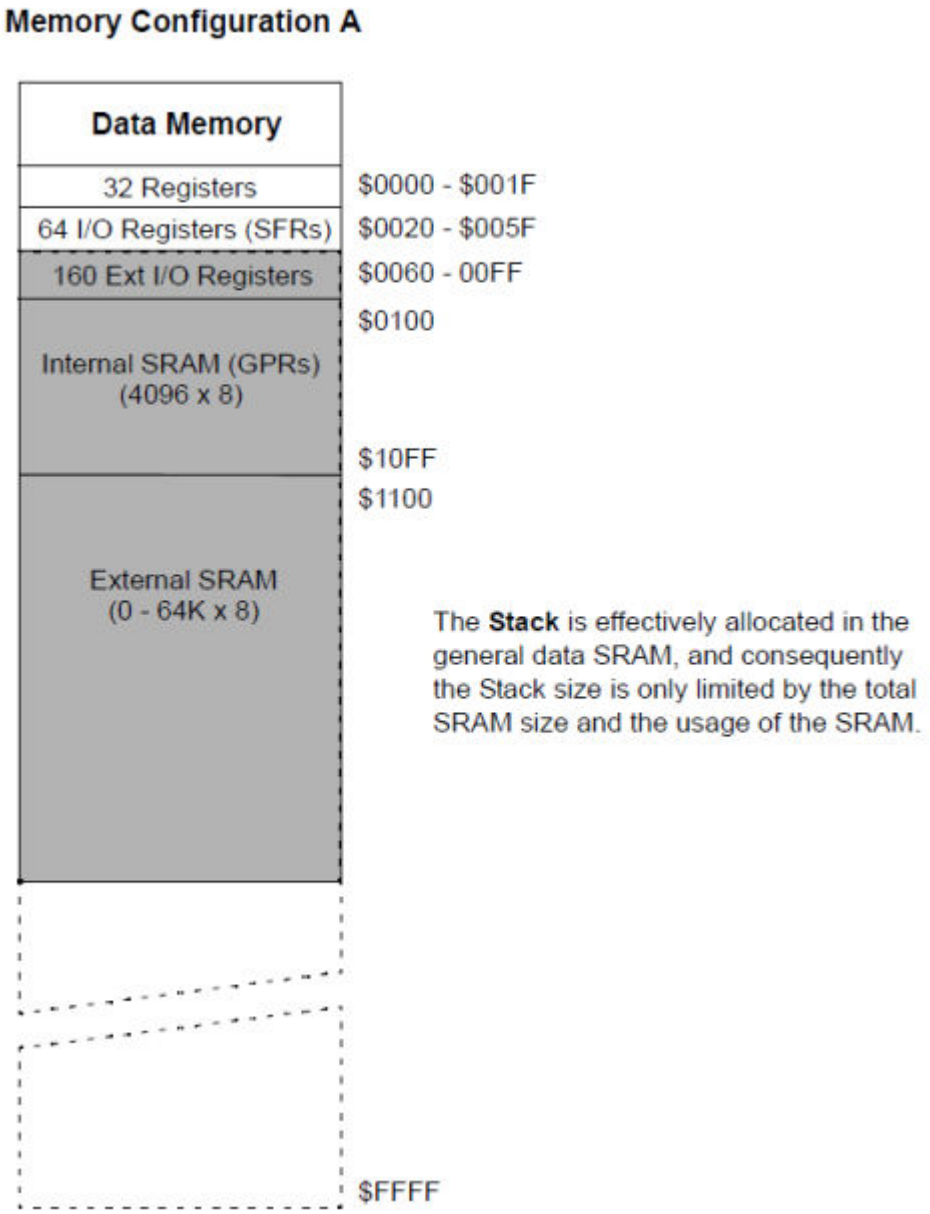
Bank 0	File Address	Bank 1	File Address	Bank 2	File Address	Bank 3	File Address
Indirect addr. (1)	00h	Indirect addr. (1)	80h	Indirect addr. (1)	100h	Indirect addr. (1)	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h	PORTA	105h	TRISA	185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h	PORTC	107h	TRISC	187h
	08h		88h		108h		188h
	09h		89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDAT	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2(1)	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh		18Eh
TMR1H	0Fh	OSCCON	8Fh	EEADRH	10Fh		18Fh
T1CON	10h	OSCTUNE	90h		110h		190h
TMR2	11h		91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD(2)	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h	WPUA	95h	WPUB	115h		195h
CCPR1H	16h	IOCA	96h	IOCB	116h		196h
CCP1CON	17h	WDTCON	97h		117h		197h
RCSTA	18h	TXSTA	98h	VRCON	118h		198h
TXREG	19h	SPBRG	99h	CM1CON0	119h		199h
RCREG	1Ah	SPBRGH	9Ah	CM2CON0	11Ah		19Ah
	1Bh	BAUDCTL	9Bh	CM2CON1	11Bh		19Bh
PWM1CON	1Ch		9Ch		11Ch		19Ch
ECCPAS	1Dh		9Dh		11Dh	PSTRCON	19Dh

.....continued

Bank 0	File Address	Bank 1	File Address	Bank 2	File Address	Bank 3	File Address
ADRESH	1Eh	ADRESL	9Eh	ANSEL	11Eh	SRCON	19Eh
ADCON0	1Fh	ADCON1	9Fh	ANSELH	11Fh		19Fh
General Purpose Register 96 Bytes	20h	General Purpose Register 80 Bytes	A0h	General Purpose Register 80 Bytes	120h		1A0h
			EFh		16Fh		
		accesses 70h-7Fh	F0h	accesses 70h-7Fh	170h	accesses 70h-7Fh	1F0h
	7Fh		FFh		17Fh		1FFh

Empty table cells: Unimplemented data memory locations, read as '0'.
Note 1: Not a physical register.
Note 2: Address 93h also accesses the SSP Mask (SSPMSK) register under certain conditions.

Figure 1-5. AVR® MCU Data Sheet – SRAM Data Memory and Stack



In addition to memory, the microcontroller has a number of peripheral device circuits on the same chip. Some peripheral devices are called input/output (I/O) ports. I/O ports are pins on the microcontroller that can be used as outputs and driven high or low to send signals, blink lights, drive speakers – just about anything that can be sent through a wire. Often these pins are bidirectional and can also be configured as inputs, allowing the program to respond to an external switch, a sensor, or to communicate with some external device.

Figure 1-6. PIC® MCU Data Sheet – Block Diagram (Excerpt)

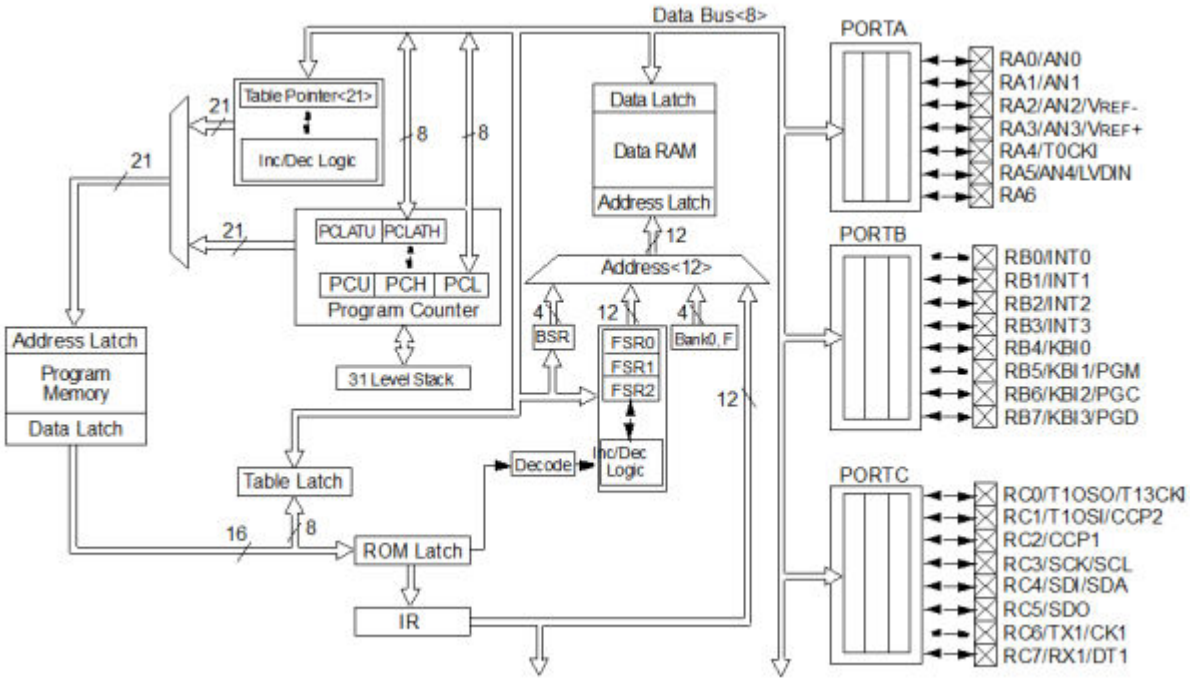
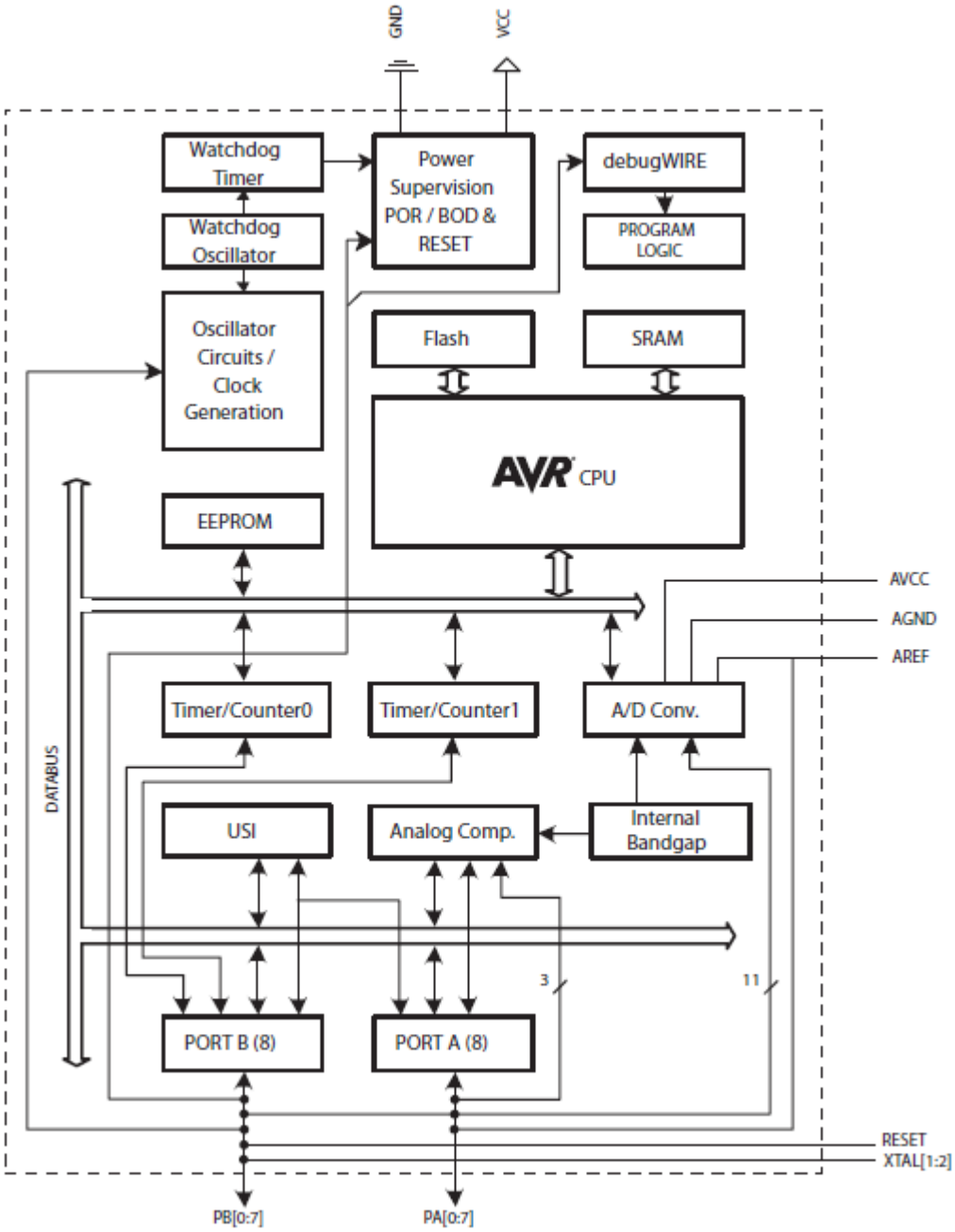


Figure 1-7. AVR® MCU Data Sheet – Block Diagram (Excerpt)



To design such a system, choose which peripherals are necessary for the application.

The following is a list of common peripherals:

- Analog-to-Digital Converters (ADCs) allow microcontrollers to connect to sensors and receive changing voltage levels.
- Serial communication peripherals that allow streaming communications over a few wires to another microcontroller, to a local network, or to the Internet.
- Peripherals on the PIC MCU called "timers" accurately measure signal events and generate and capture communications signals, produce precise waveforms, even automatically reset the microcontroller if it gets "hung" or lost due to a power glitch or hardware malfunction.

- Other peripherals detect when the external power is dipping to dangerous levels, so that the microcontroller can store critical information and safely shut down before power is completely lost.

The peripherals, and the amount of memory an application needs to run a program, largely determine which PIC MCU to use. Other factors could include the power consumed by the microcontroller and its "form factor," i.e., the size and characteristics of the physical package that must reside on the target design.

Figure 1-8. PIC® MCU Device Package Example

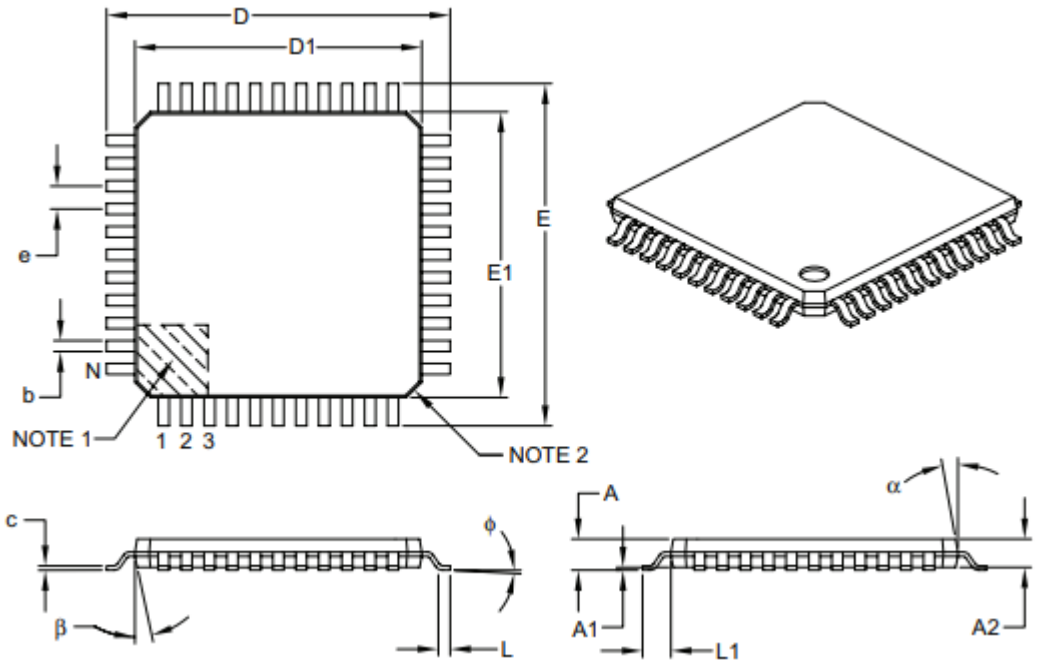
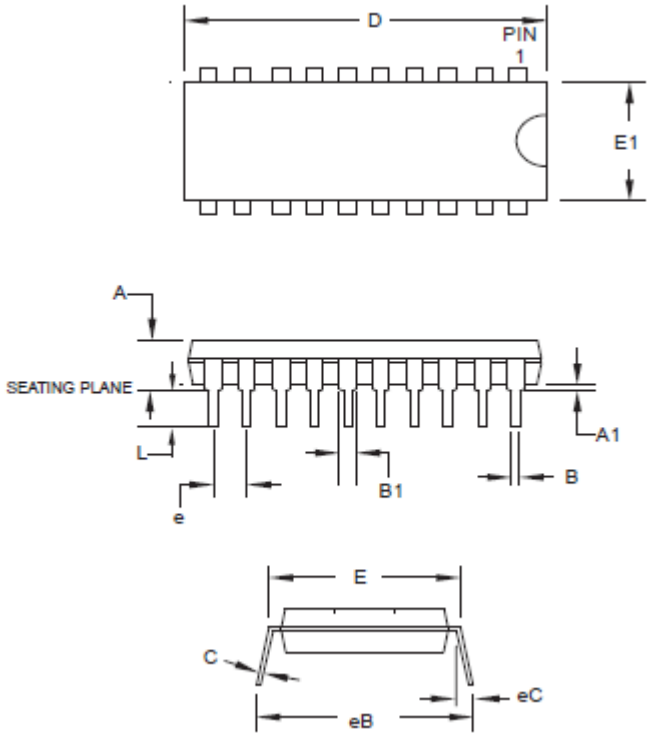


Figure 1-9. Example AVR® MCU Device Package



A microcontroller becomes active when power is applied and an oscillator begins generating a clocking signal. Depending on the microcontroller, there may be several internal and external oscillator operational modes.

Figure 1-10. PIC® MCU Data Sheet – Timing Diagram (Excerpt)

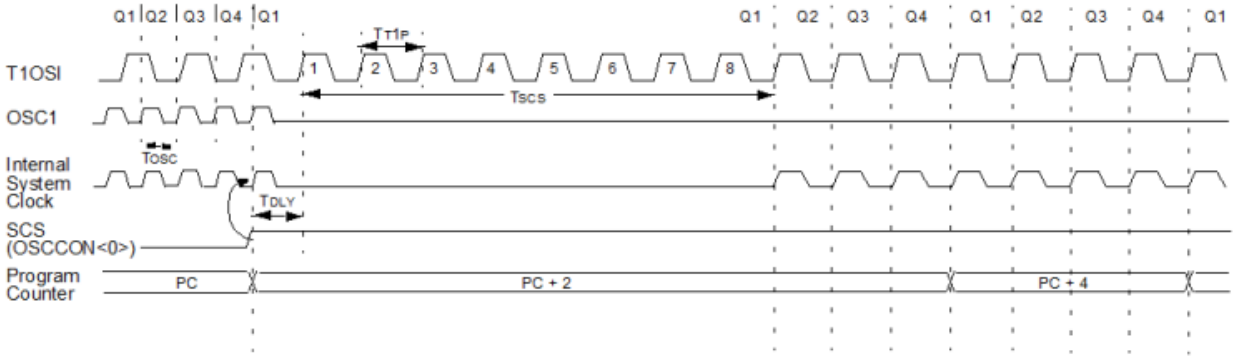
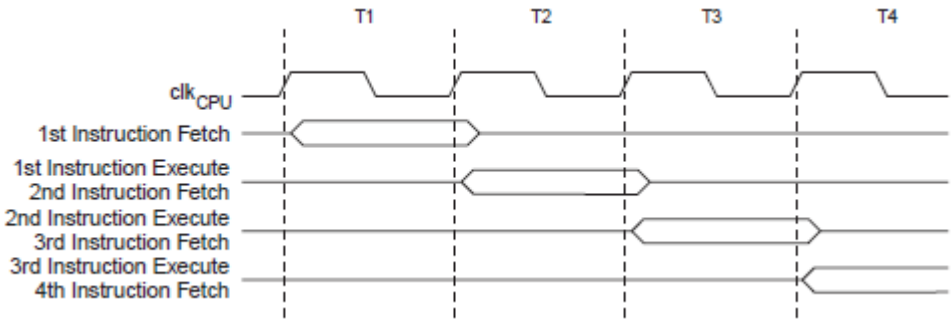


Figure 1-11. AVR® MCU Data Sheet – Timing (Excerpt)



1.1.3 Implementing an Embedded System Design with MPLAB X IDE

A development system for embedded controllers is a system of programs running on a computer that help to write, edit, debug and program code – which is the intelligence of embedded systems applications – into a microcontroller. MPLAB X IDE is such a system, it contains all the components needed to design and deploy embedded systems applications.

The typical tasks for developing an embedded controller application are:

1. Create the high level design. From the features and performance desired, decide which MCU is best suited to the application, then design the associated hardware circuitry. After determining which peripherals and pins control the hardware, write the firmware – the software that will control the hardware aspects of the embedded application. A language tool such as an assembler, which is directly translatable into machine code, or a compiler that allows a more natural language for creating programs, should be used to write and edit code. Assemblers and compilers help make the code understandable, allowing function labels to identify code routines with variables that have names associated with their use, and with constructs that help organize the code in a maintainable structure.
2. Compile, assemble and link the software using the assembler and/or compiler and linker to convert your code into "ones and zeros" – machine code for the MCUs. This machine code will eventually become the firmware (the code programmed into the microcontroller).
3. Test your code. Usually a complex program does not work exactly the way imagined, and "bugs" need to be removed from the design to get proper results. The debugger allows you to see the "ones and zeros" execute, related to the source code you wrote, with the symbols and function names from your program. Debugging allows you to experiment with your code to see the value of variables at various points in the program, and to do "what if" checks, changing variable values and stepping through routines.
4. "Burn" the code into a microcontroller and verify that it executes correctly in the finished application.

Of course, each of these steps can be quite complex. The important thing is to concentrate on the details of your own design, while relying upon MPLAB X IDE and its components to get through each step without continuously encountering new learning curves.

Step 1 is driven by the designer, although MPLAB X IDE can help in modeling circuits and code so that crucial design decisions can be made.

MPLAB X IDE really helps with steps 2 through 4. Its Programmer's Editor helps write correct code with the language tools of choice. The editor is aware of the assembler and compiler programming constructs and automatically "color-keys" the source code to help ensure it is syntactically correct. The Project Manager enables you to organize the various files used in your application: source files, processor description header files and library files. When the code is built, you can control how rigorously code will be optimized for size or speed by the compiler and where individual variables and program data will be programmed into the device. You can also specify a "memory model" in order to make the best use of the microcontroller's memory for your application. If the language tools run into errors when building the application, the offending line is shown and can be double clicked to go to the corresponding source file for immediate editing. After editing, you will rebuild and try your application again. Often this write-compile-fix loop is done many times for complex code as the sub-sections are written and tested. MPLAB X IDE goes through this loop with maximum speed, allowing you to get on to the next step.

When the code builds with no errors, it needs to be tested. MPLAB X IDE has components called "debuggers" and free software simulators for all MCUs to help test the code. Even if the hardware is not yet finished, you can begin testing the code with the simulator, a software program that simulates the execution of the microcontroller. The simulator can accept a simulated input (stimulus), in order to model how the firmware responds to external signals. The simulator can measure code execution time, single step through code to watch variables and peripherals, and trace the code to generate a detailed record of how the program ran.

When the hardware is in a prototype stage, a hardware debugger, such as an in-circuit emulator or an in-circuit debugger, can be used. These debug tools run the code in real time on your actual application by using special circuitry built into many devices with Flash program memory. They can "see into" the target microcontroller's program and data memory, and stop and start program execution, allowing you to test the code with the microcontroller in place on the application.

After the application is running correctly, you can program a microcontroller with one of Microchip's devices or development programmers. These programmers verify that the finished code is programmed correctly into the device.

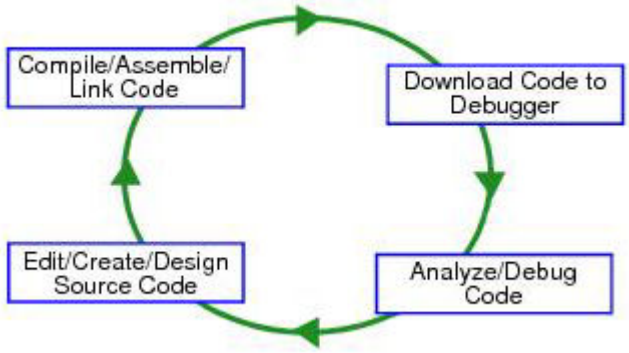
MPLAB X IDE supports most PIC MCUs, all dsPIC DSCs, and a growing number of AVR and SAM MCUs.

1.2 The Development Cycle

The process for writing an application is often described as a development cycle, since it is rare that all the steps from design to implementation can be done flawlessly the first time. More often code is written, tested and then modified to produce an application that performs correctly.

The Integrated Development Environment allows the embedded systems design engineer to progress through this cycle without the distraction of switching among an array of tools. By using MPLAB X IDE, all the functions are integrated, allowing the engineer to concentrate on completing the application without the interruption of separate tools and different modes of operation.

Figure 1-12. The Design Cycle



MPLAB X IDE is a "wrapper" that coordinates all the tools from a single graphical user interface, usually automatically. For instance, once code is written, it can be converted to executable instructions and downloaded into a microcontroller to see how it works. In this process multiple tools are needed: an editor to write the code, a project manager to organize files and settings, a compiler or assembler to convert the source code to machine code and some sort of hardware or software that either connects to a target microcontroller or simulates the operation of a microcontroller.

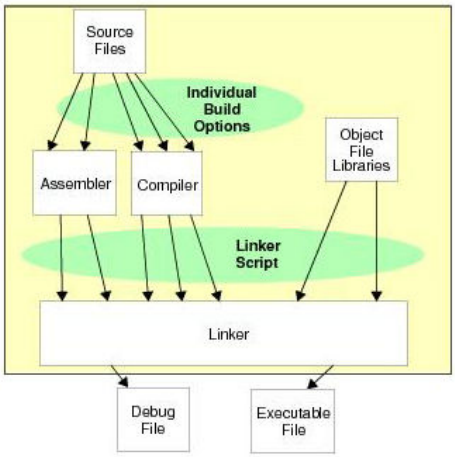
1.3 Project Manager

The project manager organizes the files to be edited and other associated files so they can be sent to the language tools for assembly or compilation, and ultimately to a linker.

The linker has the task of placing the object code fragments from the assembler, compiler and libraries into the proper memory areas of the embedded controller, and ensure that the modules function with each other (or are "linked").

This entire operation from assembly and compilation through the link process is called a project "build." Properties specified for the language tools can be invoked differently for each file, if desired, and a build process integrates all of the language tools' operations.

Figure 1-13. MPLAB® X IDE Project Manager



The source files are text files that are written conforming to the rules of the assembler or compiler. The assembler and compiler convert them into intermediate modules of machine code and placeholders for references to functions and data storage.

The linker resolves these placeholders and combines all the modules into a file of executable machine code. The linker also produces a debug file which allows MPLAB X IDE to relate the executing machine codes back to the source files.

A text editor is used to write the code. It recognizes the constructs in the text and uses color coding to identify various elements, such as instruction mnemonics, C language constructs and comments. The editor supports operations commonly used in writing source code. After the code is written, the editor works with the other tools to display code execution in the debugger. Breakpoints (which stop or "break" the execution of code) can be set in the editor and the values of variables can be inspected by hovering the mouse pointer over the variable name. Names of variables can be dragged from source text windows and then dropped into a Watches window where their changing values can be watched after each breakpoint or during code execution.

1.4 Language Tools

Language tools are programs such as cross-assemblers and cross-compilers. Most people are familiar with some of the language tools that run on a computer, e.g., Visual Basic or C compilers.

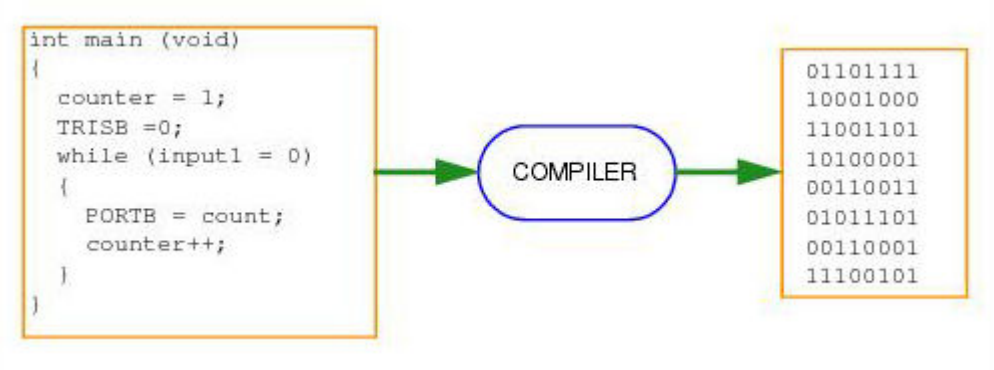
When using language tools for embedded systems, a "cross-assembler" or "cross-compiler" is used. These tools differ from typical compilers in that they run on a computer, but they produce code to run on another microprocessor (or microcontroller).

Language tools also produce a debug file that MPLAB X IDE uses to correlate the machine instructions and memory locations with the source code. This bit of integration allows the MPLAB X IDE editor to set breakpoints, allows Watches windows to view variable contents, and lets you single step through the source code, while watching the application execute.

Embedded system language tools also differ somewhat from compilers that run and execute on a computer because they must be very space conscious. The smaller the code produced, the better, because that provides the smallest possible memory usage for the target, which reduces cost. This means that techniques to optimize and enhance the code, using machine-specific knowledge, are desirable.

The size of programs for computers typically extends into the megabytes for moderately complex programs. The size of simple embedded systems programs may be as small as a thousand bytes or less. A medium size embedded system might need 32K or 64K of code for relatively complex functions. Some embedded systems use megabytes of storage for large tables, user text messages or data logging.

Figure 1-14. A Compiler Converts Source Code Into Machine Instructions



1.5 Target Debugging

In an integrated development environment (IDE), the execution of the code is tested on a debugger. The debugger can be a software program that simulates the operation of the microcontroller for testing, or it can be a hardware instrument to analyze the program as it executes in the application.

The IDE and Debugging

Debugging usually becomes urgent near the end of the project design cycle. As deadlines loom, getting the application to function as originally designed is the last step before going into deployment of the product and often has the most influence on producing delays in getting a product out. That's where an integrated development environment is most important. Doing fine "tweaks" to the code, recompiling, downloading and testing all require time. Using all tools within a single environment will reduce the time around the "cycle." These last steps, where critical bugs are worked out, are a test for the embedded systems designer. The right tool can save time. With MPLAB X IDE, many tools can be selected, but they all will have a similar interface and the learning curve from simulator to low-cost in-circuit debugger to powerful in-circuit emulator is small.

Software Debuggers

Simulators are built into MPLAB X IDE so a program can be tested without any additional hardware. A simulator is a software debugger, and the debugger functions for the simulator are almost identical to the hardware debuggers, allowing a new tool to be learned with ease. Usually a simulator runs somewhat slower than an actual microcontroller since the CPU in the computer is being used to simulate the operations of the microcontroller.

Hardware Debuggers

There are two types of hardware that can be used with MPLAB X IDE: programmers and hardware debuggers. A programmer simply burns the machine code from the PC into the internal memory of the target microcontroller. The microcontroller can then be plugged into the application and, hopefully, it will run as designed.

Usually, however, the code does not function exactly as anticipated, and the engineer is tasked with reviewing the code and its operation in the application to determine how to modify the original source code to make it execute as desired. This process is called debugging. As noted previously, the simulator can be used to test how the code will operate, but once a microcontroller is programmed with the firmware, many things outside the scope of the simulator come into play. Using just a programmer, the code could be changed, reprogrammed into the microcontroller and plugged into the target for retest, but this could be a long, laborious cycle if the code is complex, and it is difficult to understand exactly what is going wrong in the hardware.

This is where a hardware debugger is useful. Hardware debuggers can be in-circuit emulators or in-circuit debuggers, which use microcontrollers that have special built-in debugging features. A hardware debugger, like a simulator, allows the engineer to inspect variables at various points in the code, and single step to follow instructions as the hardware interacts with its specialized circuitry.

1.6 Device Programming

After the application has been debugged and is running in the development environment, it needs to be tested on its own. A device can be programmed with an in-circuit emulator, an in-circuit debugger, a development programmer, or a device programmer. MPLAB X IDE can be set to the programmer function, and the part can be "burned". The target application can now be observed in its nearly final state. Engineering prototype programmers allow quick prototypes to be made and evaluated. Some applications can be programmed after the device is soldered on the target PC board. Using In-Circuit Serial Programming™ (ICSP™) programming capability, the firmware can be programmed into the application at the time of manufacture, allowing updated revisions to be programmed into an embedded application later in its life cycle. Devices that support in-circuit debugging can even be plugged back into an in-circuit debugger after manufacturing for quality tests and development of next generation firmware.

Production programming can be accomplished using a production programmer and the MPLAB IPE, which is installed with MPLAB X IDE.

1.7 Components of MPLAB X IDE

MPLAB X IDE includes:

- a full-featured programmer's text editor that also serves as a window into the debugger.
- a project manager (visible as the Projects window) that provides integration and communication between the IDE and the language tools.
- a number of assembler/linker suites for the development of firmware for your project's device.
- a debugger engine that provides breakpoints, single stepping, Watches windows and all the features of a modern debugger. The debugger works in conjunction with debug tools, both software and hardware.
- a software simulator for all PIC and dsPIC devices as well as many AVR and SAM devices. The simulator is actually composed of several device-specific simulator executables. MPLAB X IDE decides which one to use based on your project's device.

Optional components can be acquired or purchased to work with the MPLAB X IDE.

Compiler Language Tools

MPLAB XC C compilers from Microchip provide fully integrated, optimized code for MCUs. Along with community-based compilers and third-party compilers, they are invoked by the MPLAB X IDE project manager to compile code that is automatically loaded into the target debugger for instant testing and verification.

Programming Environments/Frameworks

The MPLAB Code Configurator (MCC) is a free, graphical programming environment that generates seamless, easy-to-understand C code to be inserted into your project. Just install the MCC plugin into MPLAB X IDE by selecting *Tools>Plugins>Available Plugins*.

MPLAB® Harmony is a framework of system services, device drivers, and other libraries that are built upon a base of portable peripheral libraries to provide flexible, portable, and consistent software "building blocks" that you can use to develop your embedded PIC32 applications. The MPLAB Harmony Configurator (MHC), a time-saving hardware configuration utility for MPLAB Harmony, is available as a plugin under *Tools>Plugins>Available Plugins*.

Programmers

Production programmers, such as the MPLAB ICD 4 in-circuit debugger, are capable of the programming code into target devices in a production environment. Programmers may be used with MPLAB X IDE or MPLAB IPE to control programming of both code and data, as well as the Configuration bits to set the various operating modes of the target microcontrollers or digital signal controllers.

In-Circuit Debuggers and Emulators

In-circuit debuggers and emulators can be used to debug application code on target devices. By using some of the on-chip resources, these tools can download code into a target microcontroller inserted in the application, set breakpoints, single step and monitor registers and variables. Emulators include additional debug features, such as trace.

Plugin Tools

Several plugins are available to add to the capabilities for MPLAB X IDE. For example, the MPLAB Data Visualizer provides a mechanism to graphically view application output in real-time.

1.8 MPLAB X IDE Help

MPLAB X IDE is built upon the NetBeans platform. Therefore, many of the NetBeans functions are now MPLAB X IDE functions. For more on NetBeans Help topics, see:

https://docs.oracle.com/cd/E50453_01/doc.80/e50452/toc.htm

Help Contents

Please refer to all help files for a complete understanding of MPLAB X IDE behavior. To launch help, select *Help>Help Contents*. This merges all help files into one, so it may take a little more time to open.

Tool Help Contents

You can also view individual help files for selected tools under *Help>Tool Help Contents*. Launching an individual help file will be faster and provide a smaller search of topics.

Developer Help

Microchip Developer Help is a good place to look for tips on advanced features:

http://microchipdeveloper.com/

Enter text in the box next to "How do I?" to look up topics in Developer Help.

Figure 1-15. Microchip Developer Help

1.9 Other MPLAB X IDE Documentation

In addition to help, links to other documentation, videos, forums, and wikis are featured on the Start Page.

Figure 1-16. MPLAB X IDE Start Page

1.10 Microchip Website

Microchip Technology provides online support of Development Tools via our website at:

http://www.microchip.com/development-tools/

1.11 Microchip Store

Starting with MPLAB X IDE v3.xx, you may access the Microchip store for development tools software and hardware products via the IDE.

	Store Icon

- MPLAB X Store tab – next to the Start tab is this tab, which provides links to products.
- Help>MPLAB X Store – under the Help menu is a link to open the MPLAB X Store tab.
- MPLAB X Store Toolbar Icon – an icon on the MPLAB X Store toolbar opens the MPLAB X Store tab.

1.12 Programming Center

Add code to your device or order free samples pre-programmed using the Microchip Programming Center. Click the icon to go to the microchipDIRECT site where you can order this value-added service.

	Programming Center Icon

1.13 MPLAB X IDE Updates

MPLAB X IDE is an evolving program with thousands of users. Microchip Technology is continually designing new microcontrollers with new features. Many new MPLAB X IDE features come from customer requests and from internal usage. Continued new designs and the release of new microcontrollers ensure that MPLAB X IDE will continue to evolve.

MPLAB X IDE is scheduled for a version update approximately every few months to add new device support and new features.

For projects that are midway through development when a new version of MPLAB X IDE is released, it is considered "best practice" to not update to the new release unless there is a compelling reason to do so, such as a bug fix on a bug that inhibits the current efforts. The start of a new project is the best time to update to a new release.

Each new release of the MPLAB X IDE software has new features implemented, so the printed documentation will inevitably "lag" behind the online help. The online help is the best source for any questions about MPLAB X IDE.

To be notified of updates to MPLAB X IDE and its components, subscribe to the Development Tools section of myMICROCHIP Personalized Notification Service on:

http://www.microchip.com/pcn

1.14 Working Outside the IDE

MPLAB X IDE is designed to help you write, debug and release applications for embedded systems. However, your company may have requirements that make code development outside the IDE necessary.

To find out how you can build your code outside MPLAB X IDE, see:

http://microchipdeveloper.com/mplabx:work-outside

2. Before You Begin

AS YOU PREPARE TO USE MPLAB X IDE, DO THE FOLLOWING:

- Review the installation requirements in the readme/release notes
- Install MPLAB X IDE (and the JRE)
- Install USB drivers for hardware tools
- Connect to a target
- Install language tools (assemblers and/or compilers)
- Launch the IDE
- Shop the Microchip Store

Additionally, learn how to launch multiple instances of the IDE, multiple versions of the IDE, and use startup parameters.

2.1 Review Installation Requirements

Installation requirements, as well as other installation issues, are specified in the "Readme for MPLAB X IDE." Find a link to this HTML file on the desktop under *Learn & Discover>Getting Started>Users Guide & Release Notes*.

2.2 Install JRE and MPLAB X IDE

When you install MPLAB X IDE (based on the NetBeans platform), Java Runtime Environment (JRE) 8 is installed. This JRE is used solely with MPLAB X IDE and is not registered on your computer for general use.

For Windows XP, JRE 7 will need to be installed. The MPLAB X IDE installer will search for JRE 7 on your system and if **not** found it will recommend sites for download.

Mac OS X 10.8 (Mountain Lion) was the first version to support JRE8. Therefore Mac OS X 10.8 or greater is required for MPLAB X IDE.

2.3 Install the USB Device Drivers (for Hardware Tools)

The MPLAB X IDE installer contains a preinstaller that will place USB device drivers for supported hardware tools onto your system. After MPLAB X IDE installation, when a hardware tool is plugged into the computer, the operating system (OS) should associate the driver with the tool (i.e., finish the driver installation). However, additional steps may be needed for correct association, for some OSs and tools.

USB Driver Installation for macOS® or Linux® OS

If you are planning to install MPLAB X IDE on a Mac or Linux computer, no additional steps are needed.

USB Driver Installation for Windows® OS

If you are planning to install MPLAB X IDE on a personal computer that uses the Windows operating system, follow the instructions below.

Note: The USB hardware tool drivers for MPLAB IDE v8 are not the same as those for MPLAB X IDE.

The table below shows which tools are affected and which are not.

Instructions apply to these tools	You do not need to do anything for these tools
MPLAB REAL ICE in-circuit emulator	PICkit 2 or PICkit 3 in-circuit debugger
MPLAB ICD 3, MPLAB ICD 4, or MPLAB PICkit 4 in-circuit debugger	Other MPLAB Starter Kits
MPLAB PM3 device programmer	

.....continued	
Instructions apply to these tools	**You do not need to do anything for these tools**
PIC32 Starter Kit	

Complete the instructions in the following sections to determine your installation method.

2.3.1 Before You Install MPLAB X IDE

Be aware that if your Windows OS version of WinUSB drivers is older than the IDE preinstaller version, the system drivers will be replaced. If you want to keep your version of WinUSB drivers, rename these files before installing MPLAB X IDE.

The WinUSB driver files are located at the following locations:

32-bit OS

`C:\Windows\system32\WinUSB.dll` **(32-bit)**

`C:\Windows\system32\drivers\WinUSB.sys` **(64-bit)**

64-bit OS

`C:\Windows\SysWOW64\WinUSB.dll` **(32-bit)**

`C:\Windows\system32\WinUSB.dll` **(64-bit)**

`C:\Windows\system32\drivers\WinUSB.sys` **(64-bit)**

2.3.2 If MPLAB IDE v8.xx is NOT Installed on Your System

You do not need to do anything; the USB drivers will be preinstalled when MPLAB X IDE is installed. Once you plug your tool into a computer USB port, a "New Hardware Found" notification should appear. Then, either the install will proceed automatically or you will have to follow a wizard and choose to "Automatically select driver." However, if either procedure fails to install the drivers (associate the preinstalled driver with your tool), you will need to install/associate the drivers manually. For instructions and driver locations, see 2.3.3.4 If You Need to Manually Switch the Drivers.

2.3.3 If MPLAB IDE v8.xx is Installed on Your System

If MPLAB IDE v8.xx or earlier is already installed on your computer, you may need to run the MPLAB Device Driver Switcher to switch from MPLAB IDE v8 drivers to MPLAB X IDE drivers.

To determine which drivers are associated with the your tool:

1. Plug your desired tool into the USB connector on your personal computer.
2. Open the computer's Device Manager window.
   - 2.1. For MPLAB IDE v8.xx, the driver should be under "Custom USB Device>Microchip Custom USB Device."
   - 2.2. For MPLAB X IDE, the driver should be under "Microchip Tools>Microchip WinUSB Device."
3. If you have an MPLAB IDE v8.xx driver, you will need to use the Switcher to switch to the MPLAB X IDE drivers.

To switch to MPLAB X IDE preinstalled drivers:

1. If You Have Windows XP 64, Manually Switch
2. If You Have Windows 7 or 8, Use Administrator Mode
3. Switch USB Device Drivers (for Hardware Tools)

If the Switcher fails to switch the drivers, see:

1. If You Need to Manually Switch the Drivers
2. Tool Communication Issues

2.3.3.1 If You Have Windows XP 64, Manually Switch

Note: Microsoft Windows XP Professional SP3 is no longer supported. However, MPLAB X IDE may continue to function on this operating system.

If you are using Windows XP 64-bit OS on your computer, you will not be able to use the MPLAB Device Drive Switcher to switch from MPLAB IDE v8 drivers to MPLAB X IDE drivers; you will have to switch the drivers manually. See 2.3.3.4 If You Need to Manually Switch the Drivers.

2.3.3.2 If You Have Windows 7 or 8, Use Administrator Mode

To run the MPLAB Device Driver Switcher, you must be in Administrator mode.

To run the Switcher utility as administrator, right click on the executable and select 'Run as Administrator.' Executable locations are specified below:

- 32-bit OS: `C:\Program Files\Microchip\MPLABX\vx.xx\Switcher\32Bit\MPDDSwitch.exe`
- 64-bit OS: `C:\Program Files (x86)\Microchip\MPLABX\vx.xx\Switcher\64Bit\MPDDSwitch64.exe`

It is recommended that you use the Switcher GUI utility first to switch the drivers. If this is problematic, you may switch the drivers using command-line applications.

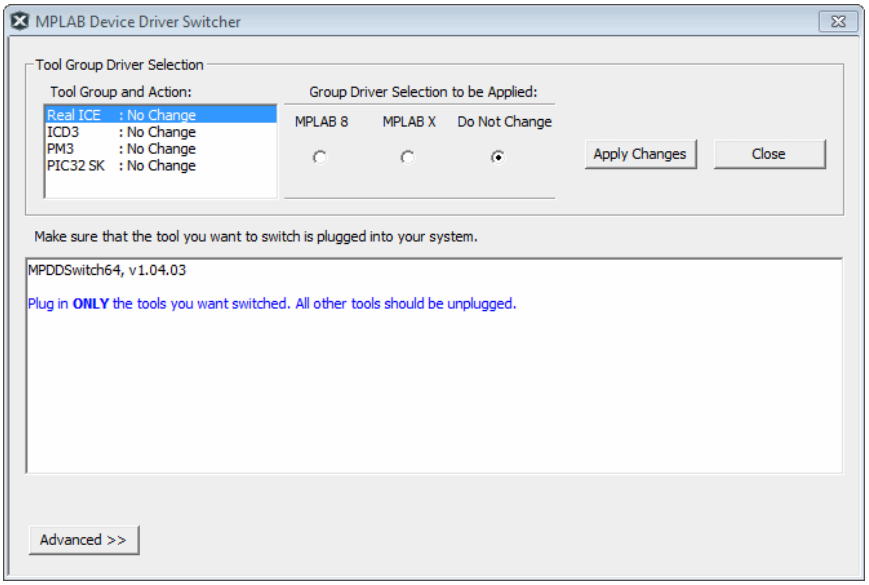
To run the command-line application – `mchpdds32.exe` or `mchpdds64.exe` – as administrator, first open the command prompt as administrator: *Start>All Programs>Accessories>Command Prompt*, right click and select 'Run as Administrator'. This will open up the Administrator: Command Prompt. After this, the instructions provided in the `ReadMe32.txt` or `ReadMe64.txt` file may be followed to complete the driver switching.

2.3.3.3 Switch USB Device Drivers (for Hardware Tools)

The MPLAB Device Driver Switcher is a GUI application that enables you to switch between MPLAB IDE v8 device drivers and MPLAB X IDE device drivers. Click on the desktop icon to launch the Switcher utility.

Figure 2-1. MPLAB Driver Switcher Icon

Figure 2-2. Switcher Utility



Switcher Operation

To switch USB device drivers:

1. Click to select the connected tool for which you wish to switch drivers under "Tool Group and Action."
   Note: If the tool(s) are not connected when Switcher is run, the driver(s) will not be installed or switched for those particular tools.

2. Click the radio button for either "MPLAB 8" or "MPLAB X."
3. Click the **Apply Changes** button. Switcher progress will be shown in the large text window. This may take some time.
4. Once the program/batch completes, view the name(s) of the driver(s) in the Device Manager window. For MPLAB X IE, the driver(s) should be under "Microchip Tools>Microchip WinUSB Device." For MPLAB IDE v8.xx, the driver(s) should be under "Custom USB Device>Microchip Custom USB Device."

Switcher Troubleshooting

1. If the GUI fails to switch the drivers, check the paths to the driver files by clicking the **Advanced** button. Run the Switcher again.
2. If the GUI still fails to switch the drivers, you will need to switch the drivers manually. For instructions and driver locations, see 2.3.3.4 If You Need to Manually Switch the Drivers.

Switcher Executable Location

You can find the Switcher executable file in the MPLAB X IDE install folder, Switcher subfolder, kept in the following location (by default):

- 32-bit OS: `C:\Program Files\Microchip\MPLABX\vx.xx\Switcher`
- 64-bit OS: `C:\Program Files (x86)\Microchip\MPLABX\vx.xx\Switcher`

where `vx.xx` is the MPLAB X IDE version.

2.3.3.4 If You Need to Manually Switch the Drivers

If you need to switch or associate the USB device drivers manually:

1. Open the Device Manager (under Control Panel). Look under "Microchip Tools" for your tool or, if you cannot find it there, under "Other Devices" for "Unknown Device."
2. Right click on your tool name or "Unknown Device" and select "Update Driver Software."
3. In the Update Driver Software dialog, select "Browse my computer for driver software."
   Note: DO NOT select "Search automatically for updated driver software." This will associate a Windows default driver with your tool, which will probably not work. If you accidentally select this, back out or exit and repeat these steps to switch to the correct driver.
4. Locate the correct device driver for your system.
   For MPLAB X IDE, the default locations for the device drivers are:

   ```
   C:\Program Files\Microchip\MPLABX\vx.xx\Switcher\32Bit\
   winusb\x86\MCHPWinUSBDevice.inf
   ```

   or

   ```
   C:\Program Files (x86)\Microchip\MPLABX\vx.xx\Switcher\64Bit\
   winusb\amd64\MCHPWinUSBDevice.inf
   ```

   where `vx.xx` is the version number of MPLAB X IDE.
5. If a Windows Security dialog pops up, select "Install this driver software anyway" to proceed to switch to the selected drivers.

2.3.3.5 Tool Communication Issues

1. If you are using a docking station or hub and have issues after plugging in the tool, you may need to plug the tool directly into a USB port on your computer.
2. If you need to switch to a device driver manually, you will need to point to the `INF` file in the `32Bit` or `64Bit` folder. For details, see 2.3.3.4 If You Need to Manually Switch the Drivers.

2.4 Connect to a Target (for Hardware Tools)

For in-circuit debuggers and emulators, refer to the following to determine how to connect your hardware tool to a target:

- the Development Tools Design Advisory
- the Header Specification (if you are using a header)

- your tool documentation

For dedicated programmers, refer to your tool documentation for connection information.

If you are using a Microchip demonstration board, evaluation kit or reference design as your target, please refer to the accompanying documentation for set up information.

2.5 Install the Language Tools

When you install MPLAB X IDE, the following language tools are installed: MPASM toolchain.

In addition, the following C compiler toolchains (compiler, assembler, linker, etc.) can be used with MPLAB X IDE. To select a compiler toolchain, consider which device you wish to use and then choose a toolchain that supports that device.

- MPLAB XC C Compilers
- AVR and Arm C Compilers
- AVRASM2 - installed with Atmel Studio

MPLAB XC compilers can be licensed as Free or PRO (full-featured, code-optimized) compilers. To install and license these compilers, view the document *Installing and Licensing MPLAB XC C Compilers* (DS50002059). MPLAB X IDE includes the option to license your installed compiler and roam in and out for network licenses. See *Tools>Licenses*.

AVR and Arm GNU C compilers (GCCs) are always free. Follow the installer screens to install on you computer.

Note: Ensure these compilers are installed where MPLAB X IDE can search for them, e.g., where MPLAB XC C compilers are installed under `C:\Program Files\Microchip`. Otherwise you will have to specify in MPLAB X IDE where to find them (see 4.7 Set Language Tool Locations).

2.6 Launch the IDE and View the Desktop

Double click on the MPLAB X IDE icon to launch the program.

Figure 2-3. MPLAB X IDE Icon

MPLAB X IDE is built upon the NetBeans platform. If you are familiar with the NetBeans IDE, then the MPLAB X IDE desktop will look familiar. However, the tabs listed below are specific to MPLAB X IDE.

- Start Page
- MPLAB X Store

On the Start Page, there are 3 tabs with links. If you don't want the Start Page displayed on start-up, uncheck "Show on Startup" beneath any tab.

Figure 2-4. MPLAB X IDE Desktop

2.7 Shop the MPLAB X Store

Click on the MPLAB X Store tab to purchase tools from the Microchip store. The tab is formatted to showcase some of the latest items for sale pertaining to MPLAB X IDE. You can also browse categories to find other items.

This tab may be accessed directly from the Help menu or from a toolbar icon.

Figure 2-5. MPLAB X Store

Additional features are coming. A microchipDirect login is planned from the Start Page, My MPLAB IDE. You will then be able to:

- See alerts on software (like HPA expiring)
- Access to your mySoftware account, where you can register and download licenses and renew HPA

2.8 Launch Multiple Instances of the IDE

Some set up is required before using hardware tools (MPLAB PICkit 4, etc.) with an instance of MPLAB X IDE. After any hardware tool setup, an instance of the IDE may be invoked from its own directory.

2.8.1 Setting Up Hardware Tools to Work with Multiple Instances

By default, you can work with up to five (5) instances of the IDE. If you want to have more instances, you will need to modify the "mchpdefport" file manually.

Use and Format of "mchpdefport" File

The "mchpdefport" file provides the information necessary for tool hot-plug use to both the IDE and to the low-level USB library (DLL, so, or dylib file). The format for this file is as follows:

```
localhost
30000
30002
30004
30006
30008
```

The first line indicates the host name on which the IDE is running.

The other lines represent port or socket numbers through which the low-level library communicates with the upper-level IDE. Each instance of the IDE will be assigned to a different port or socket. All communications between the instances of the IDE should be hidden from the user.

For up to five (5) instances of MPLAB X IDE, you do not need to alter this file. If you want more than five instances, you can edit the file to add more port or socket numbers.

Location of the "mchpdefport" File

Within a default installation of the IDE, the "mchpdefport" file can be found in the following place, depending on the operating system:

OS	Location

Windows (64-bit)	`C:\Windows\system32`
	and
	`C:\Windows\SysWOW64`
	Note: Both occurrences of "mchpdefport" must be modified.
	Note: If you are on a 64-bit system and use a 32-bit editor to edit "mchpdefport" in `C:\Windows\system32`, you will actually be modifying the "mchpdefport" file in `SysWOW64`. You must use a 64-bit editor to edit the file in `system32`.

Windows (32-bit)	`C:\Windows\system32`
Linux	`/etc/.mplab_ide`
Mac (OS X)	`/etc/.mplab_ide`

2.8.2 Invoking Instances of the IDE

MPLAB X IDE requires that each instance has its own user directory. Therefore, preferences that are set or plugins that are added to one instance will not be reflected in another instance.

In order to invoke multiple instances, launch the IDE with the `--userdir` option and specify a directory.

Window OS

Create a shortcut with the `--userdir` option. For example, on Windows 7 OS:

1. Right click on the desktop and select *New>Shortcut*.

2. Browse to the installed (default) MPLAB X IDE executable:
   `C:\Program Files (x86)\Microchip\MPLABX\vx.xx\mplab_platform\bin\mplab_ide.exe` where `vx.xx` is the MPLAB X IDE version.
3. At the end of that line, type the location of the user directory. You can either place your directory under the default location for this type of MPLAB X IDE information:
   `--userdir "C:\Users\MyFiles\AppData\Roaming\.mplab_ide\dev\anydir"`
   or you can place it where ever you like:
   `--userdir anydir`
4. Click OK.

Linux OS

The installed version, initiated without any parameters (clicking on the desktop icon), will run with a user directory of `$(HOME)/.mplab_ide`. To change the user directory, run the `$InstallationDir/mplab_platform/bin/mplab_ide` shell script, passing the argument `--userid anydir`. For example, to run MPLAB X IDE in two different instances:

```
$ /opt/microchip/mplabx/vx.xx/mplab_platform/bin/mplab_ide --userdir ~/.anydir1 &
$ /opt/microchip/mplabx/vx.xx/mplab_platform/bin/mplab_ide --userdir ~/.anydir2 &
```

where `vx.xx` is the MPLAB X IDE version.

You can create desktop icons that have the user ID embedded, too.

Mac OS

Open a Shell window and type the command line listed below to execute your installation of MPLAB X IDE in an alternate user directory. You can either place your directory under the default location for this type of MPLAB X IDE information:

```
$/bin/sh /Applications/microchip/mplabx/vx.xx/mplab_ide.app/Contents/
Resources/mplab_platform/bin/mplab_ide --userdir "${HOME}/Library/
Application Support/mplab_ide/dev/anydir"
```

or you can place it wherever you like:

```
$/bin/sh /Applications/microchip/mplabx/vx.xx/mplab_ide.app/Contents/
Resources/mplab_platform/bin/mplab_ide --userdir anydir
```

2.9 Launch Multiple Versions of the IDE

Before the release of MPLAB X IDE v3.00, you could have different versions of the IDE on your PC by installing to different directories. Beginning with MPLAB X IDE v3.00, this is done by default, for example:

```
C:\Program Files (x86)\Microchip\MPLABX\v3.00
```

You can launch and run different versions at the same time. Each version has its own user directory.

You can launch a version of MPLAB X IDE and a version of MPLAB IDE (v8 or earlier) on the same PC, but keep in mind of the USB hardware driver limitations mentioned in "**Install the USB Device Drivers (for Hardware Tools)**."

2.10 Launch using Startup Parameters

Many command-line options are from the NetBeans platform. See:

http://wiki.netbeans.org/FaqStartupParameters

In addition, a list of MPLAB® X IDE options may be found by using the `--help` option. For example, on a 64-bit Windows® 7 system:

```
>C:\Program Files (x86)\Microchip\MPLABX\vx.xx\mplab_platform\bin\mplab_ide --help
```

3. Tutorial

THIS TUTORIAL PROVIDES A GUIDED EXAMPLE FOR WORKING WITH AN MPLAB X IDE PROJECT.

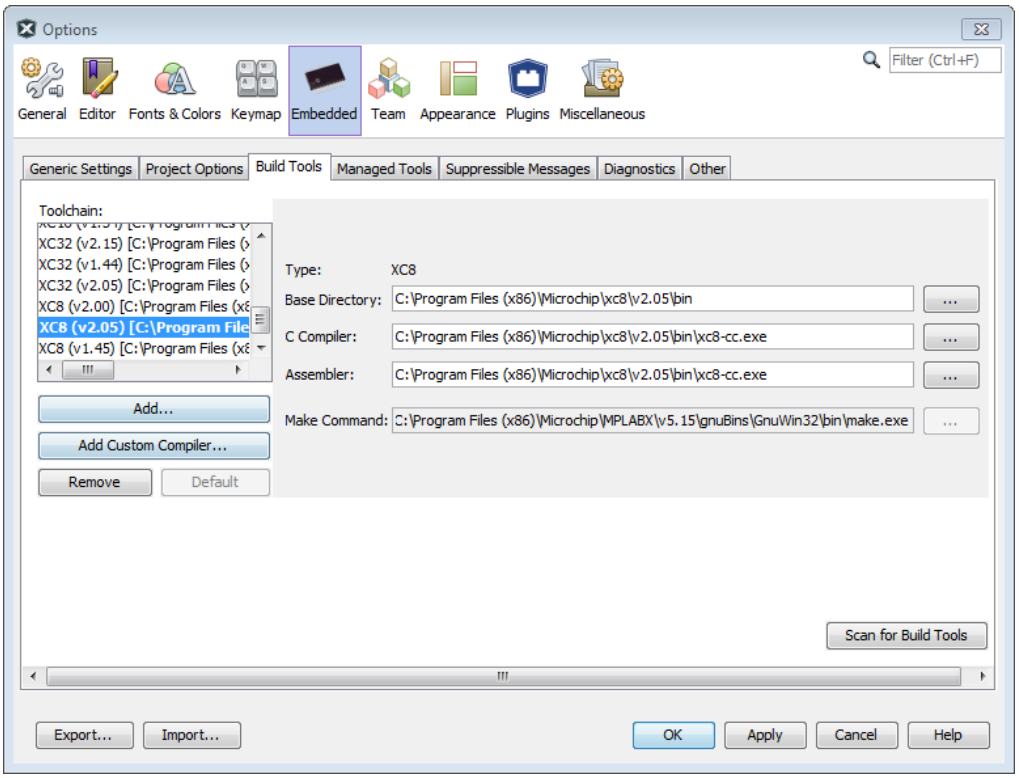
The example project will display potentiometer values on four LEDs. After downloading and opening the project in MPLAB X IDE, how to set/change project properties and run the project is explained. Debug features such as setting breakpoints, stepping through code, viewing variable values, and viewing/changing register and memory values are demonstrated.

3.1 Installing and Setting Up the Software

Download and install the software listed below. Refer to section 2. Before You Begin for installation details.

- Install MPLAB X IDE. Download the free IDE at http://www.microchip.com/mplabx.
- Install MPLAB XC8 C compiler. Download a free MPLAB XC compiler at http://www.microchip.com/xc.

Launch MPLAB X IDE. Ensure that the compiler is shown in the IDE window *Tools>Options>Embedded>Build Tools*. If not, click the **Add** button to find the compiler install location manually.

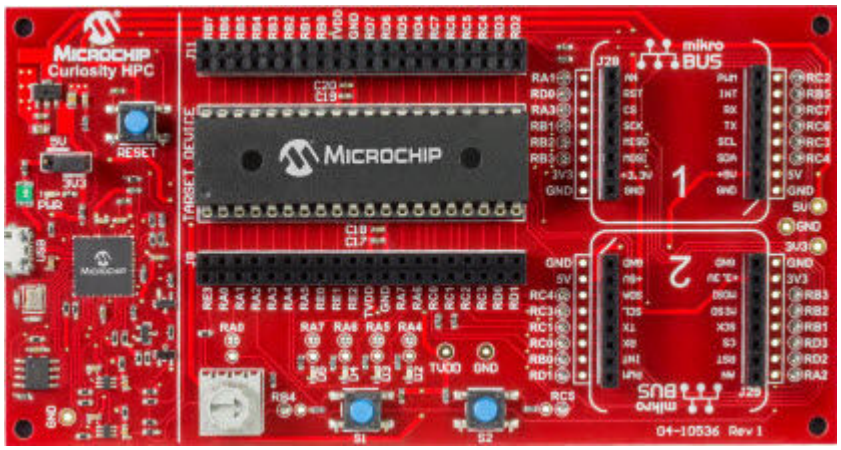


3.2 Connecting the Hardware

The Curiosity High Pin Count (HPC) Development Board (DM164136) with a PIC16F18875 will be used. Refer to the following product pages for more information:

- DM164136
- PIC16F18875

For a supported USB cable, plug the mini USB connector into the evaluation board and plug the USB connector into your computer. Drivers will install automatically.

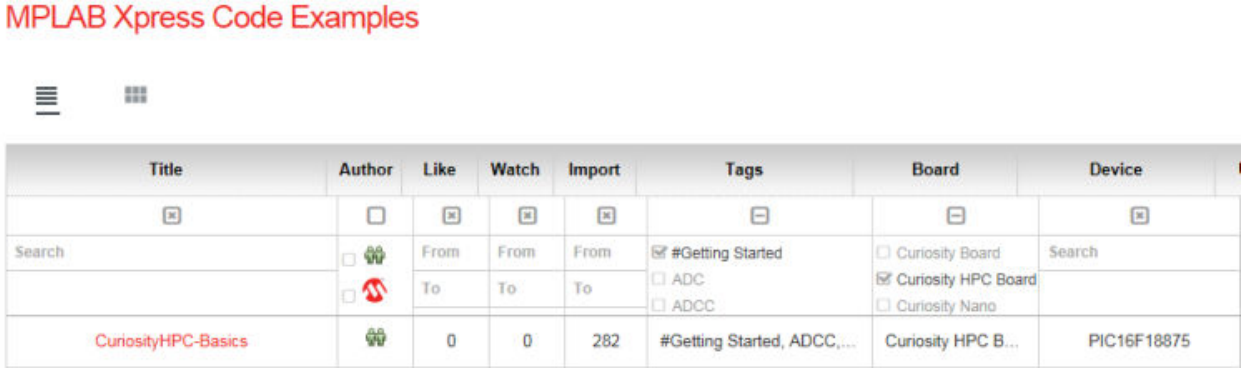


3.3 Downloading the Example Code

Go to MPLAB Xpress Code Examples.

Then complete the following selections:

- Under **Tags**, select the "#Getting Started" check box.
- Under **Board**, select the "Curiosity HPC Board" check box.
- Under **Title**, click on "CuriosityHPC-Basics."



On the "CuriosityHPC-Basics" page, you will find links to information on the board and device. Click **Download** to download the example project. Unzip the contents.

Tip: Download to the `Users>UserName>MPLABXProjects` directory.

3.4 Opening the Example Project in MPLAB X IDE

In MPLAB X IDE, click the "Open Project" icon  . Scroll through the list, select the "CuriosityHPC-Basics" project and click **Open Project**.

View the Projects and Files Windows

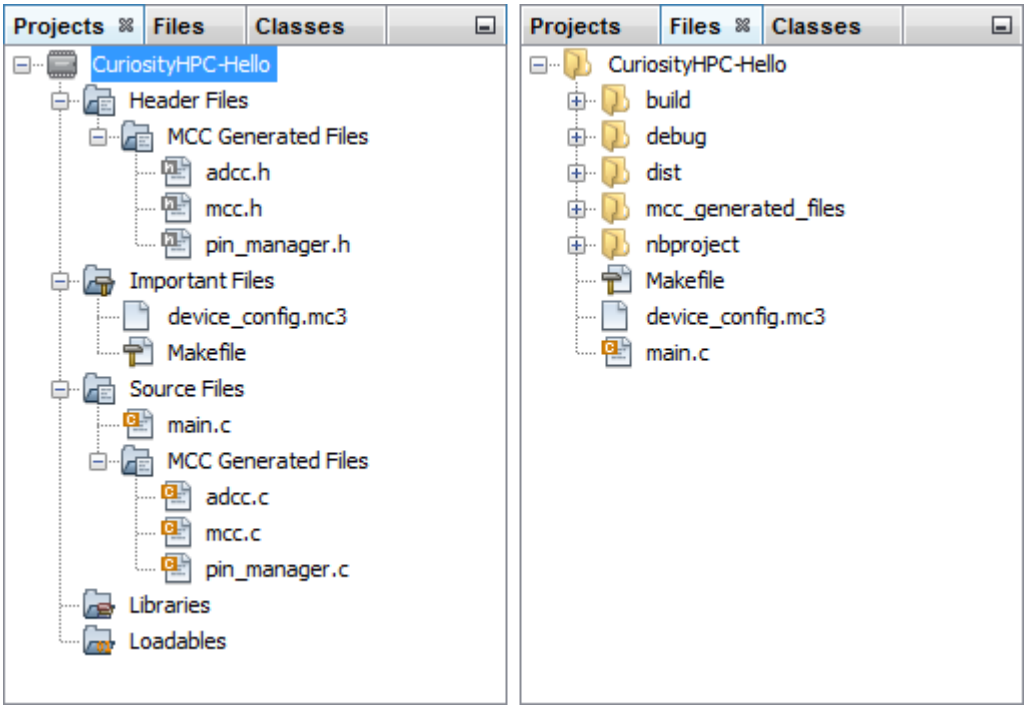
The Projects window displays the project tree with files grouped by category. For more information on these categories, see 8.1 Projects Window View.

Click on the Files window to see a file manager view of project files. See 8.2 Files Window View.

If you double click on a file name in either window, the related file will open in an Editor window. To close the tab, click on the "x" next to the file name.

Right click on the project name in the Projects window to view the pop-up (context) menu. You can do the same on the project's subfolders and window background.

Figure 3-1. Projects and Files Windows

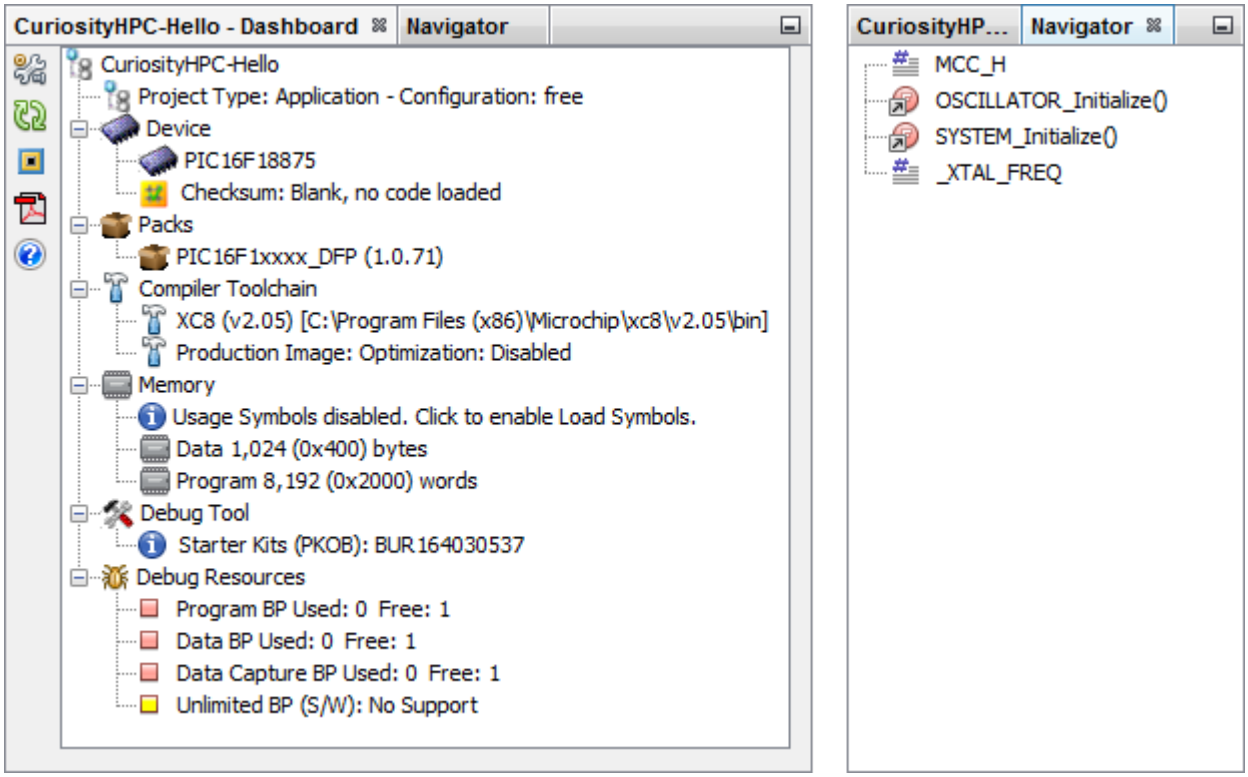


View the Dashboard and Navigator Windows

The Dashboard window shows details about the project. For more information, see 12.6 Dashboard Window.

The Navigator window provides a compact view of the file that is currently selected and simplifies navigation between different parts of the file.

Figure 3-2. Dashboard Window on Project Open



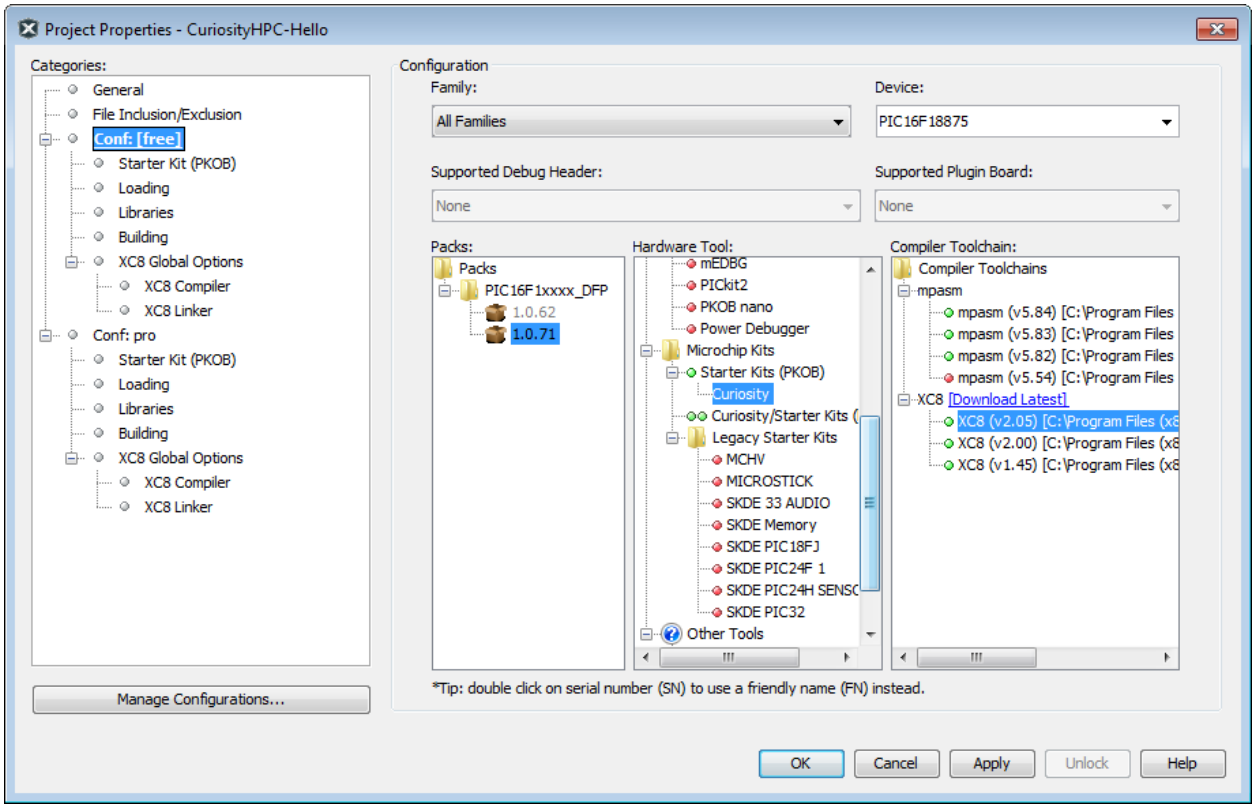
3.5 Setting Project Properties

In the Projects window, right click on the project name to drop down a menu. Select "Properties."

Review the Project Properties window. Click **OK** to save any changes.

Categories	The project is set to use the "free" version of the compiler. If you have a licensed version, you may select "pro."
Device	Verify this matches the device on the board.
Packs	The pack version shown in the figure is for MPLAB X IDE v5.20.
Hardware	Curiosity should be selected as a Starter Kit with PKOB (the built-in debugger).
Compiler Toolchain	The tutorial project was created using MPLAB XC8 v1.37. You can change to the latest compiler version (as in the figure) or you can download the original project version from: Downloads Archive

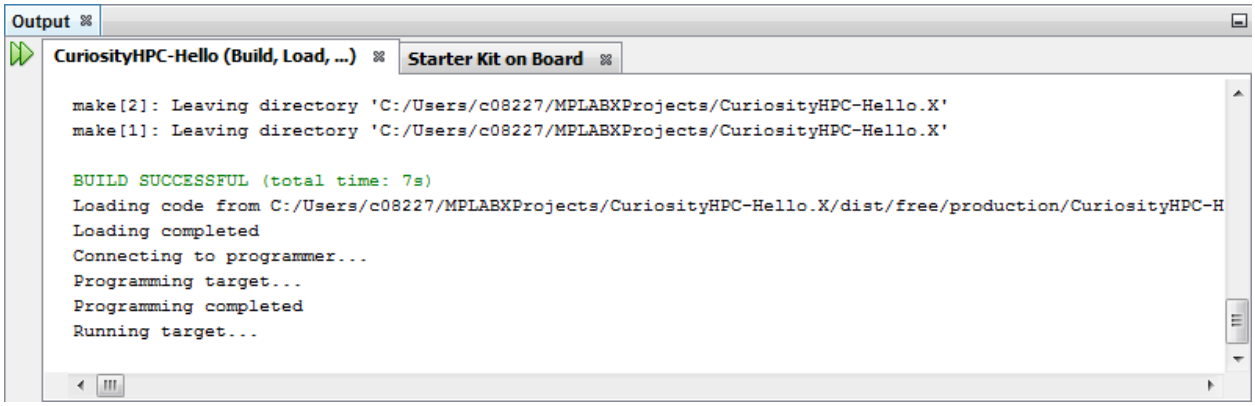
Figure 3-3. Project Properties Window



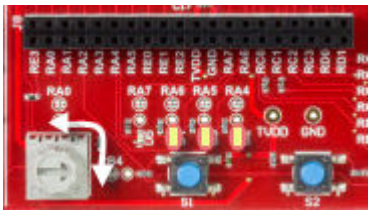
3.6 Running the Code

Click on the Run Project icon  to run your program. Run progress will be visible in the Output window (see figure). Alternately, use the "Hold in Reset" button  to toggle between device Reset and running.

Figure 3-4. Output Window - Code Running



Turn the potentiometer on the board to see the LEDs light up representing pot values.



3.7 Debugging the Code

For this tutorial, the code used has been tested and runs successfully. However, your own code may need to be debugged as you develop your application.

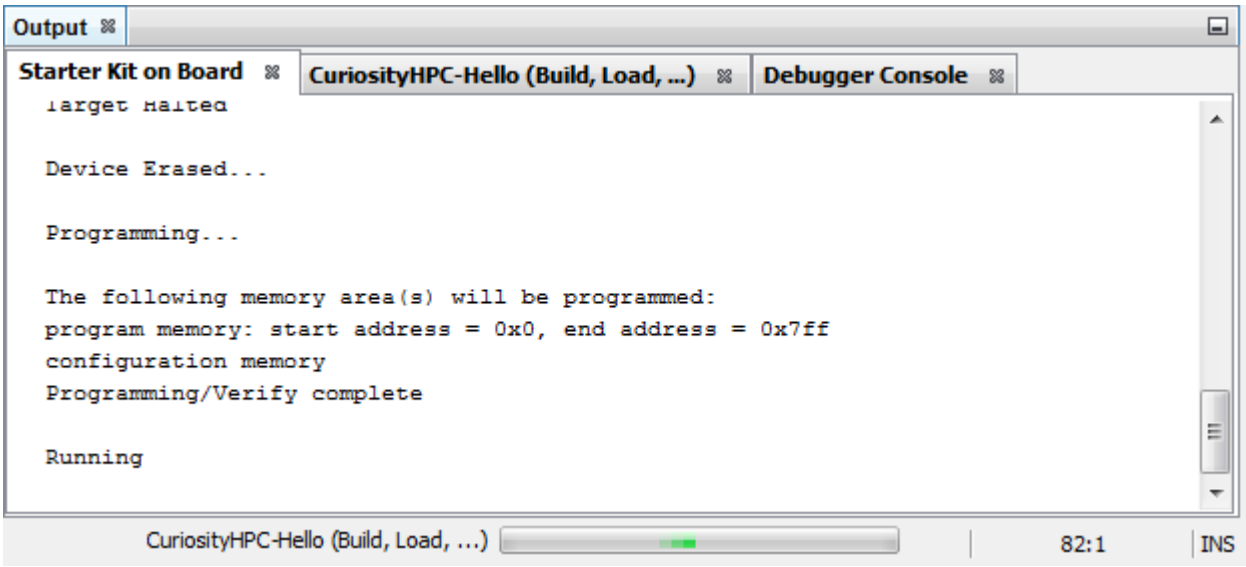
Click on the "Debug Project" icon  to begin a debug session. Debug progress will be visible in the Output window (see figure.)

To halt your application code		Pause
To run your code again		Continue
To end execution of your code		Finish Debugger Session

For more on debugging C/C++ code projects, see also NetBeans Help:

Debugging C/C++ Projects Tutorial

Figure 3-5. Ouptut Window - Code Running in Debug Mode



3.8 Setting Breakpoints

When debugging code, it can be useful to suspend execution at a specific location in code so that variable values can be examined. To do this, use breakpoints.

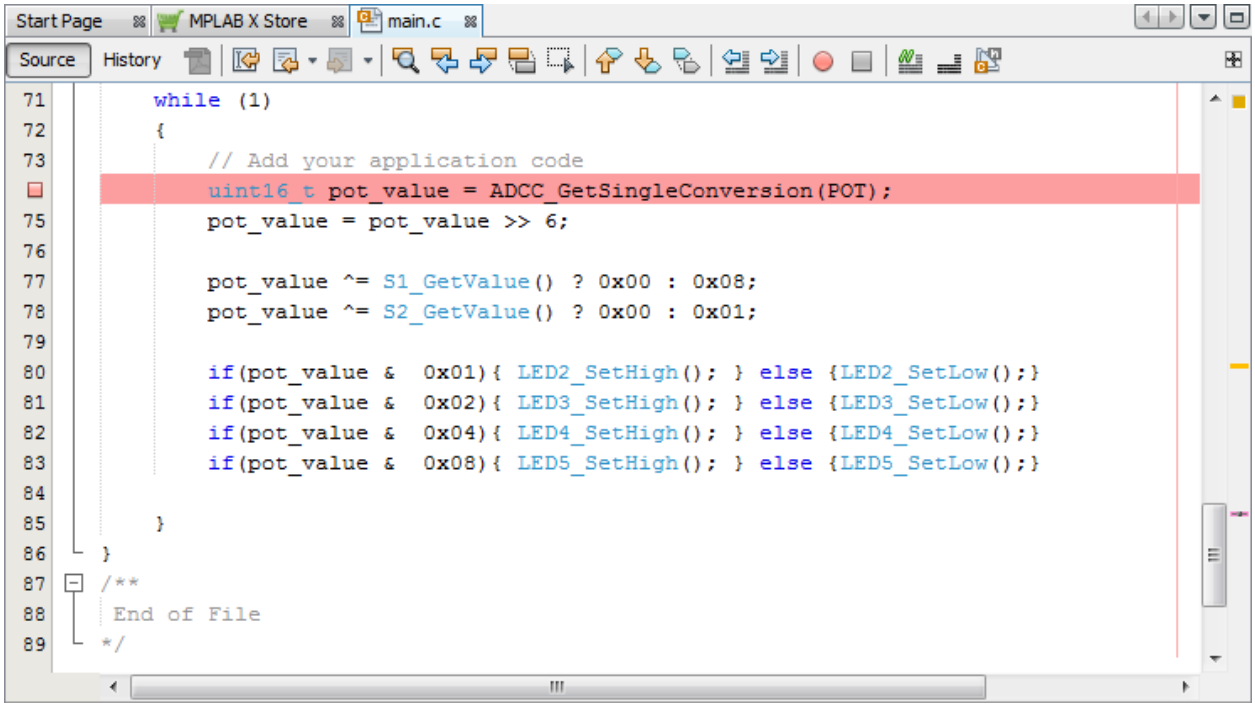
For more information on breakpoints, see also NetBeans Help:

3.8.1 Set Breakpoint

If your code is running, end execution by clicking the "Finish Debugger Session" icon  .

In the Projects window, under "Source Files," double click on the file `main.c` to open it in an Editor window. Set a breakpoint on the line in the figure by clicking in the left margin of the line.

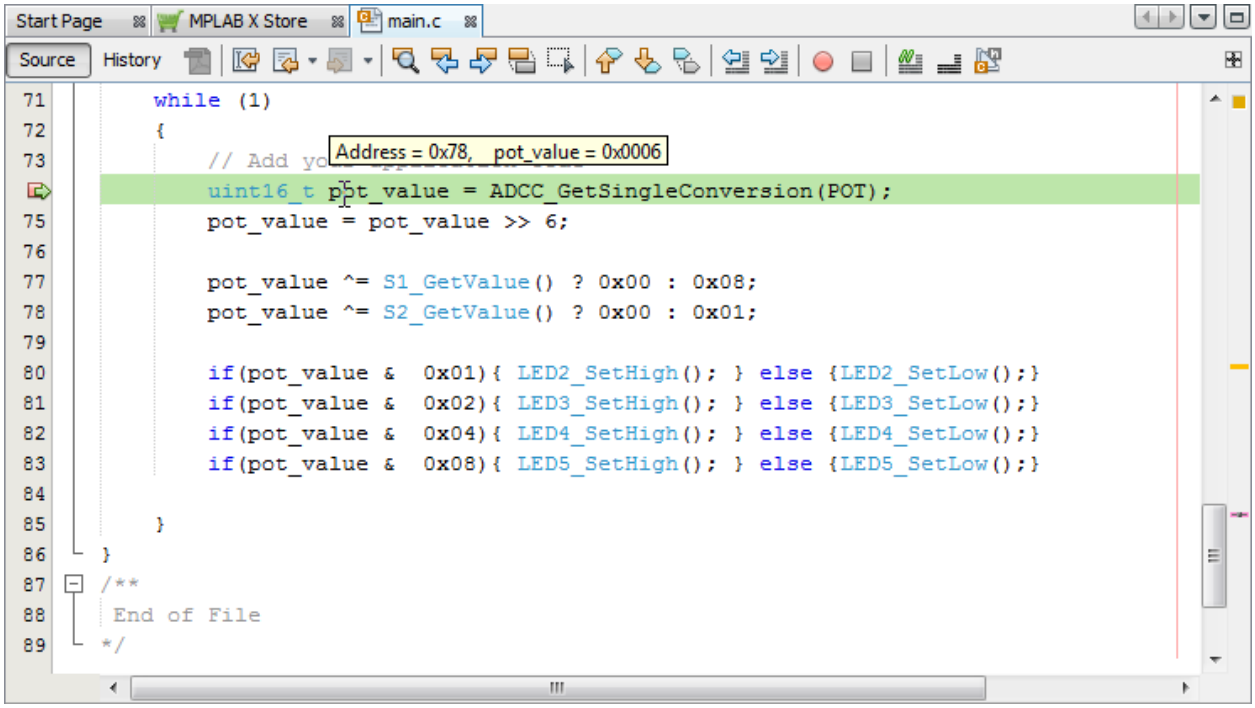
Figure 3-6. Breakpoint Set in Code



3.8.2 Halt at Breakpoint

Click the "Debug Project" icon  to execute the program again. The program will halt at the breakpoint. Hover over the `pot_value` variable to see its current value.

Figure 3-7. Program Execution Halted at Breakpoint



3.8.3 Troubleshooting

If you see a bulb warning icon  next to lines of code in the editor, you may need to set the path for include files. See:

Set The Include Directory Path

Note: For MPLAB XC8 v2.xx, both AVR and PIC MCUs are supported, so the compiler install directory structure has changed.

3.9 Stepping Through Code

Use Step Into to move through the code from the breakpoint halt. Stepping allows you to examine changes in variable values or determine if the program flow is correct.

Step Over – Executes one source line of a program. If the line is a function call, it executes the entire function and stops.

Step Into – Executes one source line of a program. If the line is a function call, it executes the program up to the function's first statement and stops.

Step Out – Executes one source line of a program. If the line is a function call, it executes the functions and returns control to the caller.

Run to Cursor – Runs the current project to the cursor's location in the file and stops program execution.

Figure 3-8. Code During Step

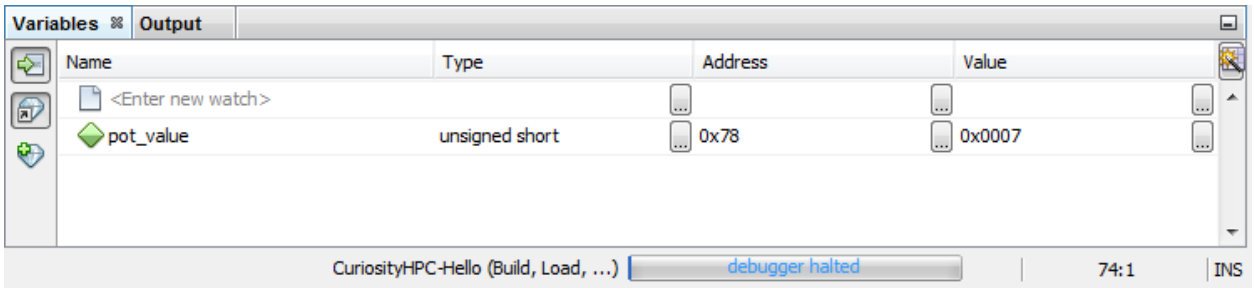
```
71      while (1)
72      {
73          // Add your application code
            uint16_t pot_value = ADCC_GetSingleConversion(POT);
75          pot_value = pot_value >> 6;
76
            pot_value ^= S1_GetValue() ? 0x00 : 0x08;
78          pot_value ^= S2_GetValue() ? 0x00 : 0x01;
79
80          if(pot_value &  0x01){ LED2_SetHigh(); } else {LED2_SetLow();}
81          if(pot_value &  0x02){ LED3_SetHigh(); } else {LED3_SetLow();}
82          if(pot_value &  0x04){ LED4_SetHigh(); } else {LED4_SetLow();}
83          if(pot_value &  0x08){ LED5_SetHigh(); } else {LED5_SetLow();}
84
85      }
86  }
87  /**
88   End of File
89  */
```

3.10 Viewing Variable Values

You can view the changing values of variables in the Variables window. To open, select *Window>Debugging>Variables*. For more information on this window, see 4.17 Watch Local Variable Values Change.

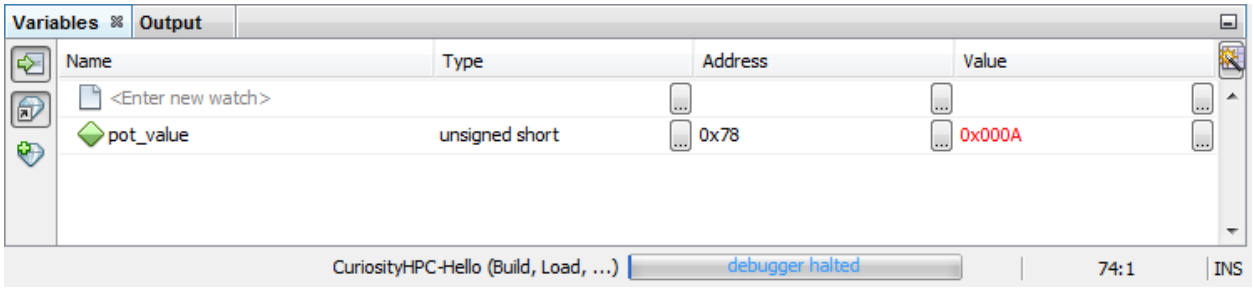
Click the "Debug Project" icon  . The code will execute and stop at the breakpoint. `pot_value` information will be shown in the Variables window.

Figure 3-9. Variables Window - First Breakpoint Halt



Turn the pot on the board so the value will change. Click the "Continue" icon  . The code will execute and stop at the breakpoint again. View the Variables window to see that the value has changed (red text).

Figure 3-10. Variables Window - Second Breakpoint Halt



3.11 Watching Symbol Values Change

Watch the values of global symbols or SFRs change in the Watches window. Determining if these values are as expected during program execution will help you to debug your code.

3.11.1 Edit the Code

Click the "Finish Debugger Session" icon  .

Currently `pot_value` is a local symbol. To make it a global symbol, declare it before `main()`.

```
#include "mcc_generated_files/mcc.h"
uint16_t pot_value;
/*                         Main application
 */
void main(void)
```

Then remove `uint16_t` from the line that contains the first usage of this symbol.

```
    while (1)
    {
        // Add your application code
        pot_value = ADCC_GetSingleConversion(POT);
```

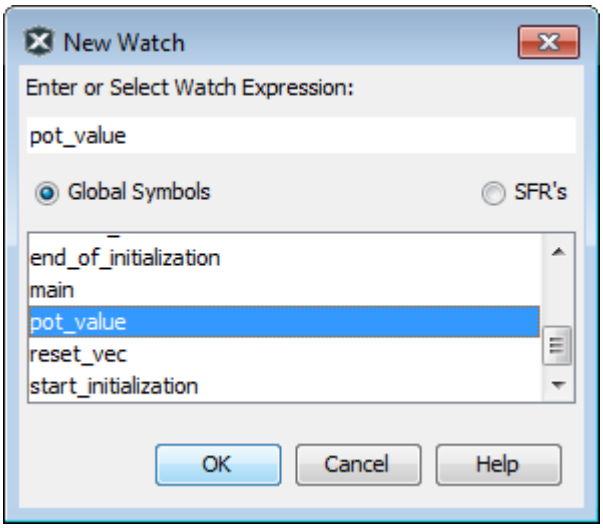
Place a breakpoint at this line and again click the "Debug Project" icon  . The program should halt on this line.

3.11.2 Create a New Watch

To create a new watch:

1. Select *Debug>New Watch*. The New Watch dialog will open.
2. Enter a Watch expression, in this case `pot_value`, and then click **OK**. The Watches window will now appear on the desktop with the symbol.

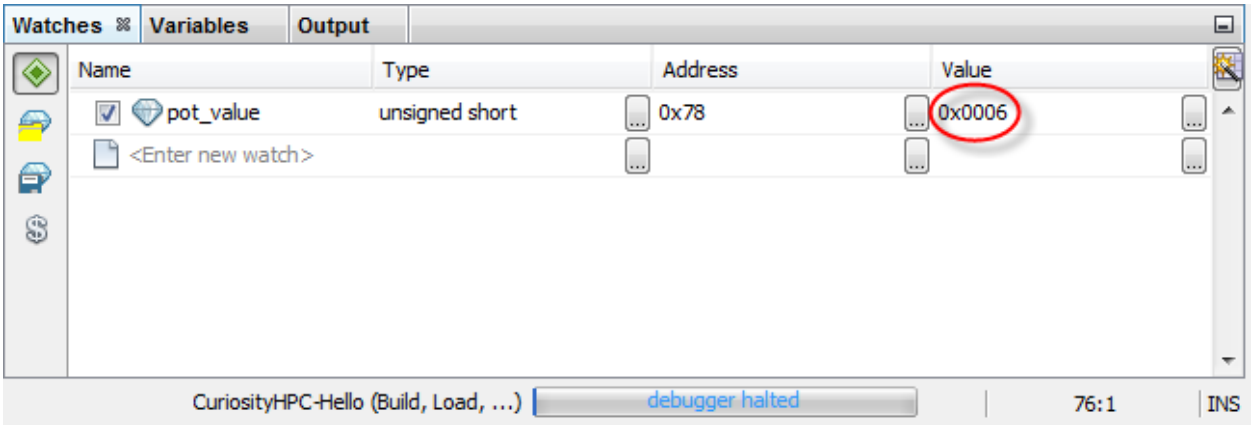
Figure 3-11. New Watch Symbol



3.11.3 View Value Changes in Watches Window

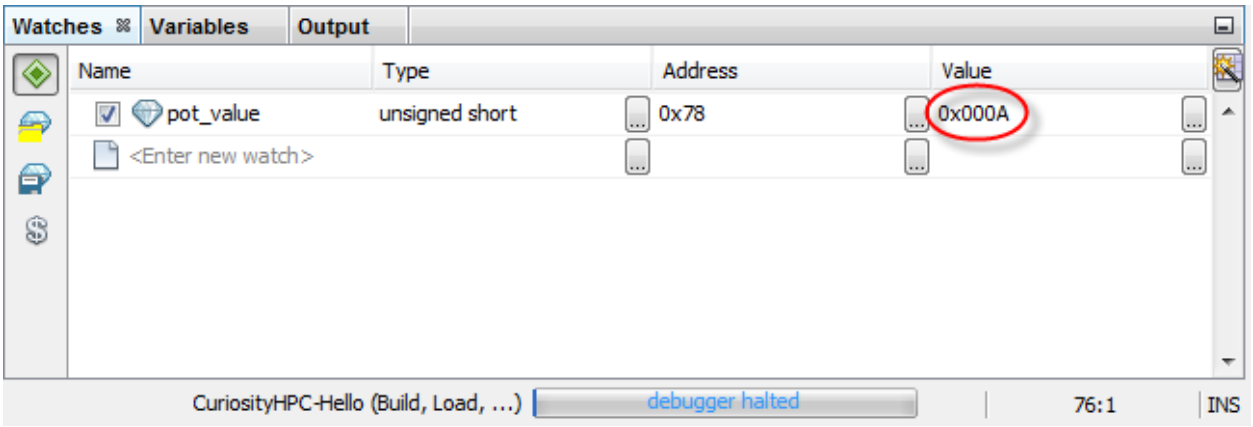
Click the "Debug Project" icon  . The code will execute and stop at the breakpoint. `pot_value` information will be shown in the Watches window.

Figure 3-12. Watches Window - First Breakpoint Halt



Turn the pot on the board so the value will change. Click the "Continue" icon  . The code will execute and stop at the breakpoint again. View the Watches window to see that the value has changed.

Figure 3-13. Watches Window - Second Breakpoint Halt



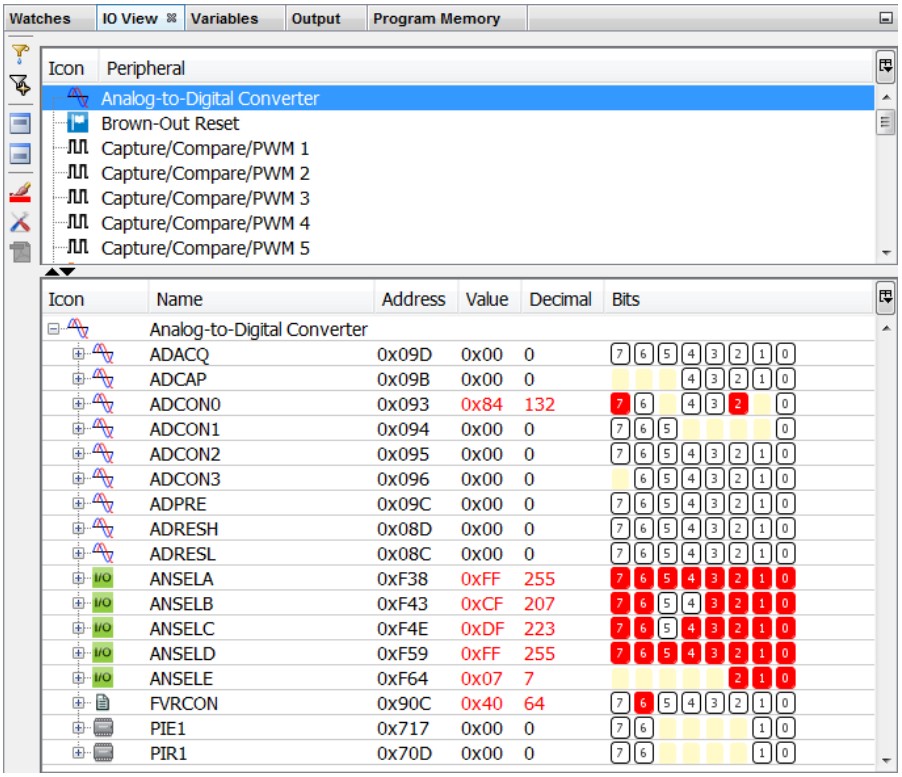
For more on this window, see 12.19 Watches Window.

3.12 Viewing I/O Registers

The I/O View window provides a graphical view of the I/O memory map of the device associated with the active project. This debug tool will display the actual register content when debugging, allowing verification of peripheral configurations. It can also be used to modify the content of a register without having to recompile.

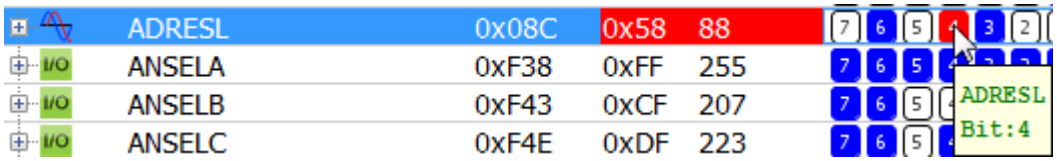
Open the I/O View window by selecting *Window>Debugging>IO View*. In the Peripheral (upper) part of the window, click on "Analog-to-Digital Converter." The registers associated with this peripheral will be shown in the Register (lower) part.

Figure 3-14. IO View Window



Since the ADC is used to convert potentiometer signals into digital signals for LED display, some values will change if you turn the pot. Turn the pot and then click the "Continue" icon  .

You may also change register values directly by clicking on them.



For more information on this window, see the following sections:

- 5.20 View Registers for the Project (I/O View)
- 12.8 I/O View Window

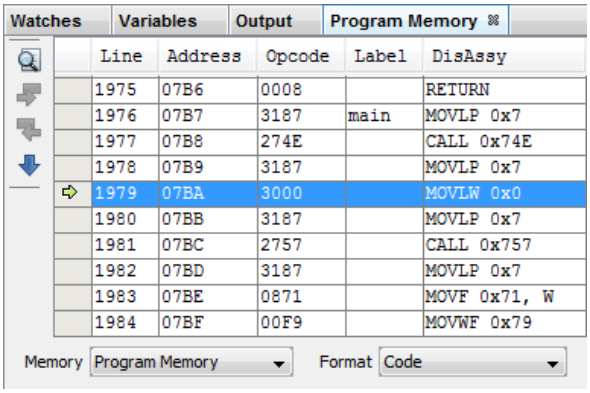
3.13 Viewing Device Memory (including Configuration Bits)

MPLAB X IDE has flexible, abstracted memory windows that provide a customizable view of differing types of device memory. Select *Window>Target Memory Views* to see a list of device-specific memory windows.

For example, to view the Flash memory:

1. Select *Window>Target Memory Views>Program Memory*.
2. The Program Memory window will open showing the last halt location.

Figure 3-15. Memory Window Content at Breakpoint Halt



To set Memory window options, right click in the Memory window to pop up a menu with various options such as display options, fill memory, table import/export, and output to file. The content of the menu depends on the window. See 12. MPLAB X IDE Windows and Dialogs.

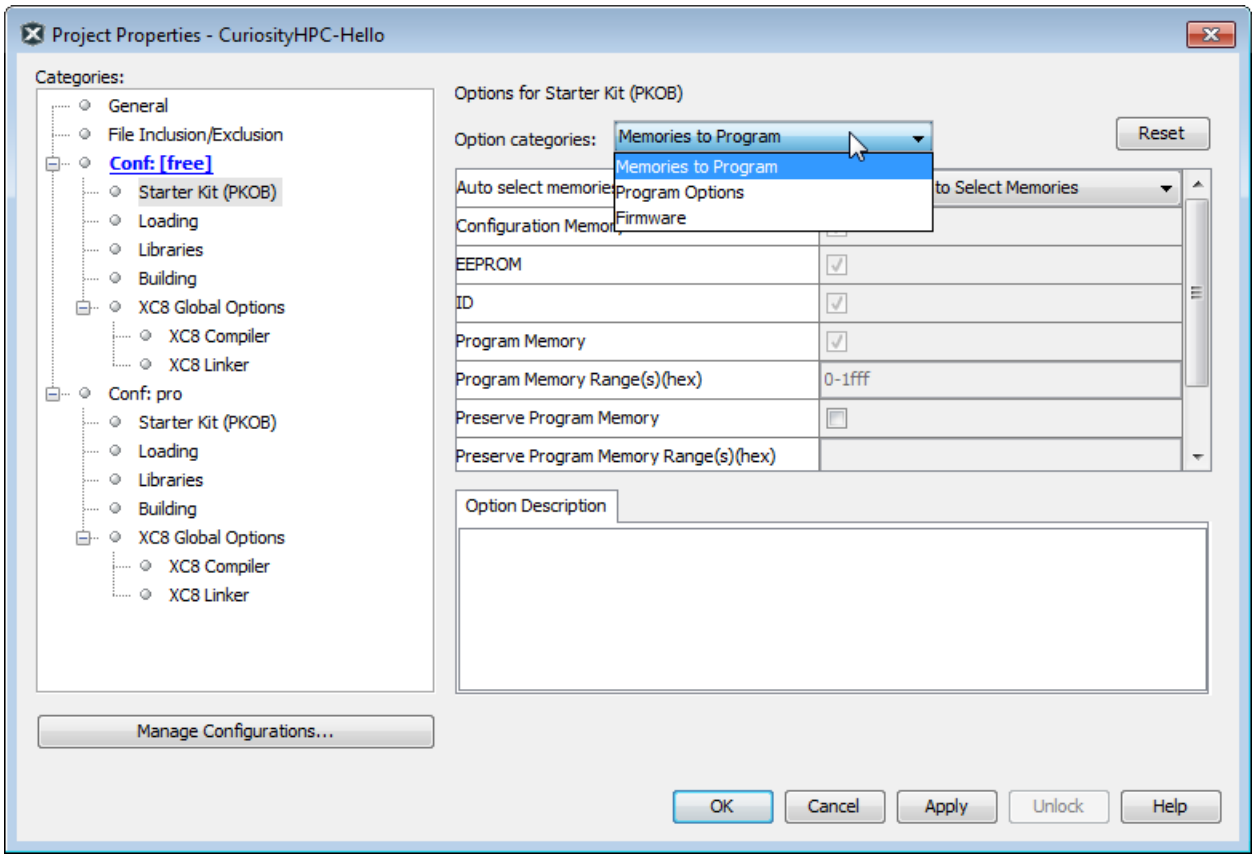
To refresh Flash Memory window:

1. Halt your program (Finish Debugger Session).
2. Click on the"Read Device Memory" icon  .

3.14 Programming a Device

After your code is debugged, you can program it onto a target device.

First, check the programming options in the Project Properties window. For this tutorial, no options need to be changed.



Finally, program the device by clicking the "Make and Program" icon  .

Note: Not all programming functions are in the MPLAB X IDE. For additional programming support, see the MPLAB IPE included with the MPLAB X IDE installation.

4. Basic Tasks

The following steps show how to work with projects in MPLAB® X IDE.

Table 4-1. Working with Basic Tasks

1 Preliminaries	1. Before You Begin, install MPLAB X IDE, set up any hardware tools (install USB drivers and properly connect to your target) and install language tools for your selected device. 2. Launch MPLAB X IDE to begin.
2 Create and Build a Project	1. Create a New Project by using the New Project wizard. Then View Changes in Desktop Panes. 2. Open Project Properties to View or Make Changes to Project Properties, Set Up or Change Debugger/Programmer Tool Options and Set Up or Change Language Tool Options. 3. Set Language Tool Locations and Set Other Tool Options in the Tools Options dialog. 4. Create a New File to add to your project or Add Existing Files to a Project. Enter or edit your application code to the File window. 5. Discover other features for Editor Usage. 6. Add and Set Up Library and Object Files. 7. Set File and Folder Properties to keep or exclude individual files or entire folders from the build. 8. Set Build Properties for your project. Here you may select the project configuration type, pre- and post-build steps, using the checksum as the user ID, and loading an alternative hex file on build. 9. Build a Project.
3 Execute Code	1. Run Code to execute your code. 2. Debug Code to execute your code in a debug session.
4 Debug Code	1. Control Program Execution with Breakpoints. Set breakpoints in-line or via the Breakpoint window. 2. Step Through Code as the program executes. 3. Watch Symbol Values Change in the Watches and Variables windows. 4. View or Change Device Memory. Memory types are dependent on the device selected. Additionally, see Set Configuration Values in the Configuration Bits Window.
5 Program a Device	1. Program a Device using simple toolbar buttons.

4.1 Create a New Project

A *Project* is a group of source files and associated information about how to compile and link the source files and run the resulting program. The IDE stores project information in a project folder that includes a makefile and metadata files. Your source directories do not need to be physically located in the project folder, though it is recommended for project portability.

In the IDE, you always work inside a project even if your program is contained in a single source file. Each project must have a makefile so the IDE can build the project. A project's makefile can be generated by the IDE or you can use a makefile that was previously created outside the IDE.

Projects with IDE-generated makefiles are called **Managed Projects** and come in several varieties. When you create a project in MPLAB X IDE using the New Project Wizard, you will be able to choose from available types of project.

Projects that use makefiles, created outside the IDE, are called **User Makefile Projects**. For more information on creating this type of makefile, see Creating Makefiles Outside of MPLAB® X IDE.

You work with projects through the IDE's **12.15 Projects Window**.

You can change a Standalone (Application) project to a Library project by changing the Configuration Type. For details, see 4.10.1 Change Project Configuration Type.

4.1.1 MPLAB X IDE v5 New Project Format

The new project format in v5.00 and greater supports packs which contain versioned device information. Packs can be found under:

```
<MPLAB X IDE install directory>\v5.xx\packs
```

Managing Projects

Projects created in or updated to v5.00 or greater are not backward compatible to v4.xx.

When you open a project created prior to v5.00, a message displays, announcing that it will upgrade the project to the current project version.

- If you choose **Yes**, the project will be upgraded to the current version and the project can no longer be opened with the earlier version of the IDE.
- If you choose **No**, you cannot save any modifications to the project in v5.00 or greater. You must make any changes in v4.xx.

If you upgraded a project to v5.xx but need to revert a project back to v4.xx, install the plugin in MPLAB X IDE found under *Tools>Plugins>Available Plugins>Save As v4.xx Project*. Once that plugin is installed, you can save your project in the older version by selecting *Tools>Embedded>Save as MPLAB X v4.xx Project*.

About Device Packs

Previously (v4.20 and earlier versions), supported device (*PIC) files that were built into each MPLAB X IDE version. You could not update the devices supported in this version; you had to wait until the next MPLAB X IDE version for device support.

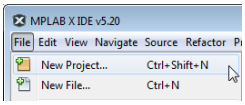
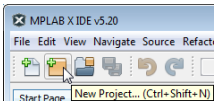
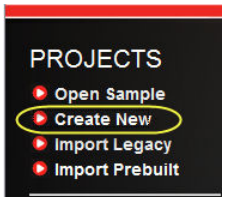
Now (v5.00 and latest versions), the device files are grouped into versioned device family packs. Although each MPLAB X IDE version comes with device packs, you may upgrade the device pack version to include new devices, new device feature support, or device bug fix support.

For more information on device packs, see:

5.1 Work with Device Packs

4.1.2 Launch New Project Wizard

New Projects can be launched in several ways.

	*File>New Project* (or Ctrl+Shift+N)
	New Project icon (on the File toolbar)
	**Start Page**, **Learn & Discover** or **My MPLAB X IDE** tab, "Projects," > "Create New"

4.1.3 Step 1: Choose Project

Choose a project category. In most cases you will choose a project type from "Microchip Embedded":

The following options are available to choose for a project category:

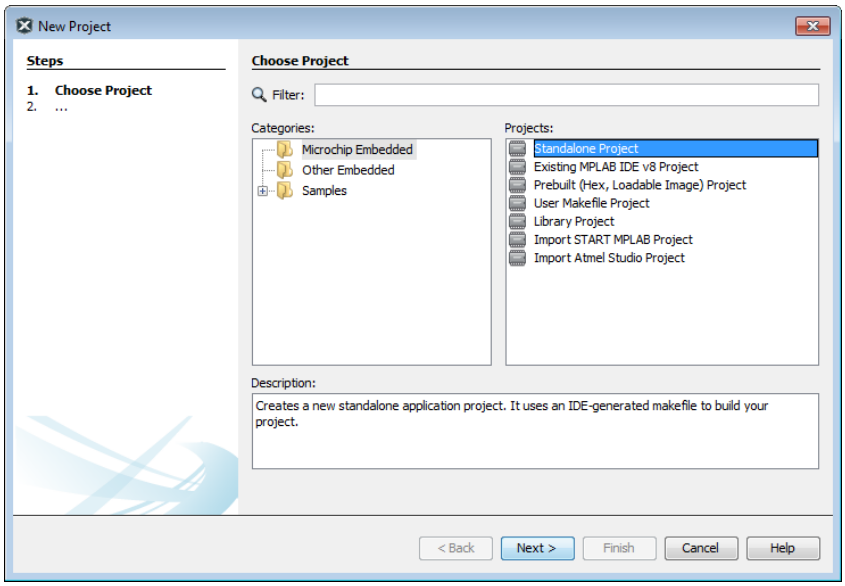
- Standalone Project – Create a new C and/or assembly code project. This type of project is shown in this section.
- Existing MPLAB IDE v8 Project – Convert your existing MPLAB IDE v8 project into an MPLAB X IDE project. For details see 5.3 Import an Existing MPLAB IDE v8 Project.
- Prebuilt (Hex, Loadable Image) Project – Load an existing project image into MPLAB X IDE. For details, see 5.4 Prebuilt Projects.
- User Makefile Project – Create a project that makes use of an external makefile. For details, see 6.6 Create User Makefile Projects.
- Library Project – Create a new C and/or assembly code project that will build into a library, instead of an executable hex file. For details, see 5.7 Library Projects.
- Import START MPLAB Project and Import Atmel Studio Project - Import these Atmel projects. For details, see 5.8 Import Atmel Studio 7 or Atmel START Project.

Other options are also available:

- 5.9 Other Embedded Projects – projects from other vendors.
- 5.10 Sample Projects – includes ready-to-use projects for different device families and project templates for different device families.

After you have made your selections, click **Next>** to move to the next dialog.

Figure 4-1. Project Wizard – Choose Project



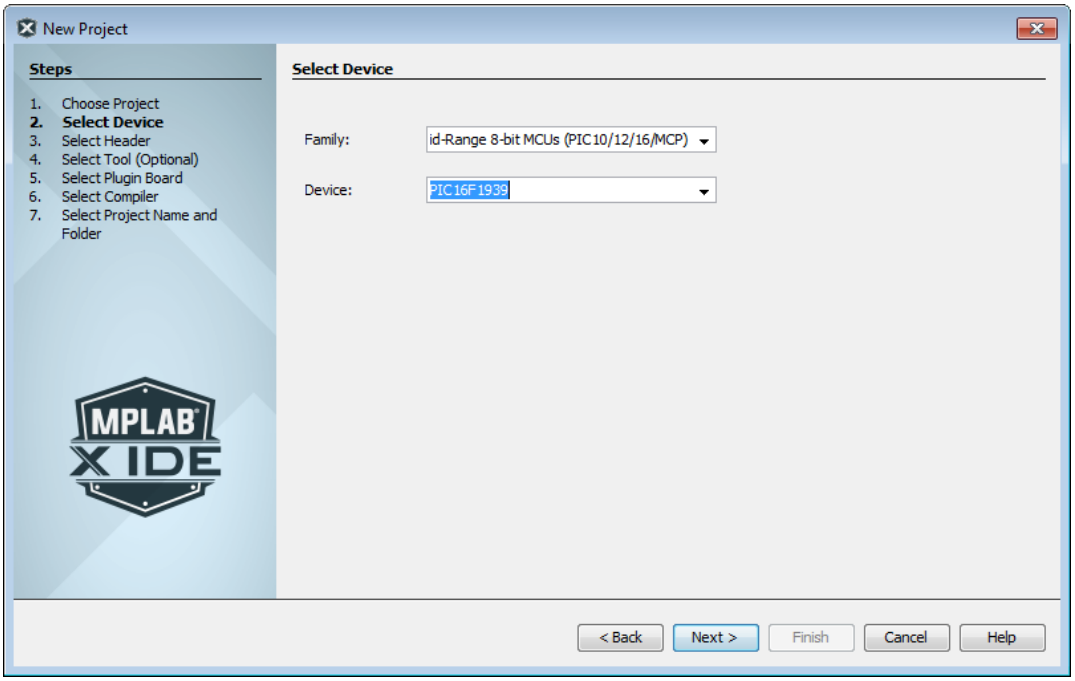
4.1.4 Step 2: Select Device

Select the device you will be using in your application from the "Device" drop-down list. To narrow your selection list, chose a "Family" first.

For LF devices, be aware that you will need to set up your physical device to be powered by a lower voltage than "F" type devices, typically 3.3 V instead of 5.0 V. If a higher voltage will damage your device, MPLAB X IDE will display a Warning dialog to remind you.

Click **Next>**.

Figure 4-2. Project Wizard – Select Device



4.1.5 Step 3: Select Header

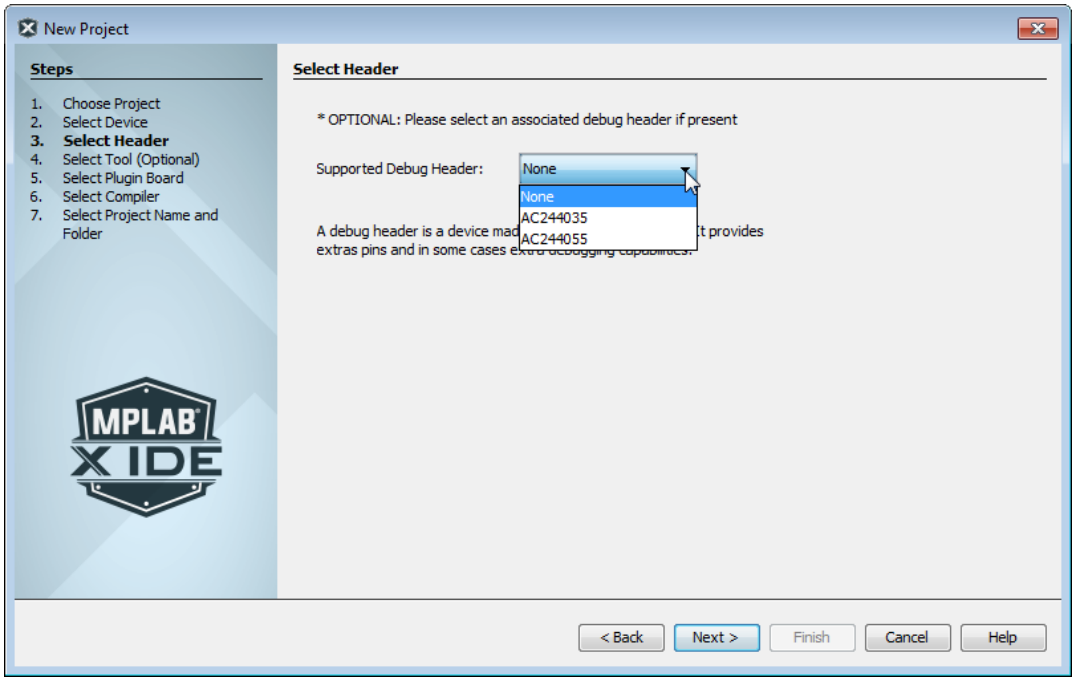
Select a header for your project device. If one is not available, MPLAB X IDE will not show this step. If one is available, consult the Developer Help topics below for more information:

- Processor Extension Paks and Debug Headers
- Emulation Extension Paks and Emulation Headers

Click **Next>** when done.

Note: You can select a header later (if one is available) using the Project Properties window.

Figure 4-3. Project Wizard – Select Header



4.1.6 Step 4: Select Tool

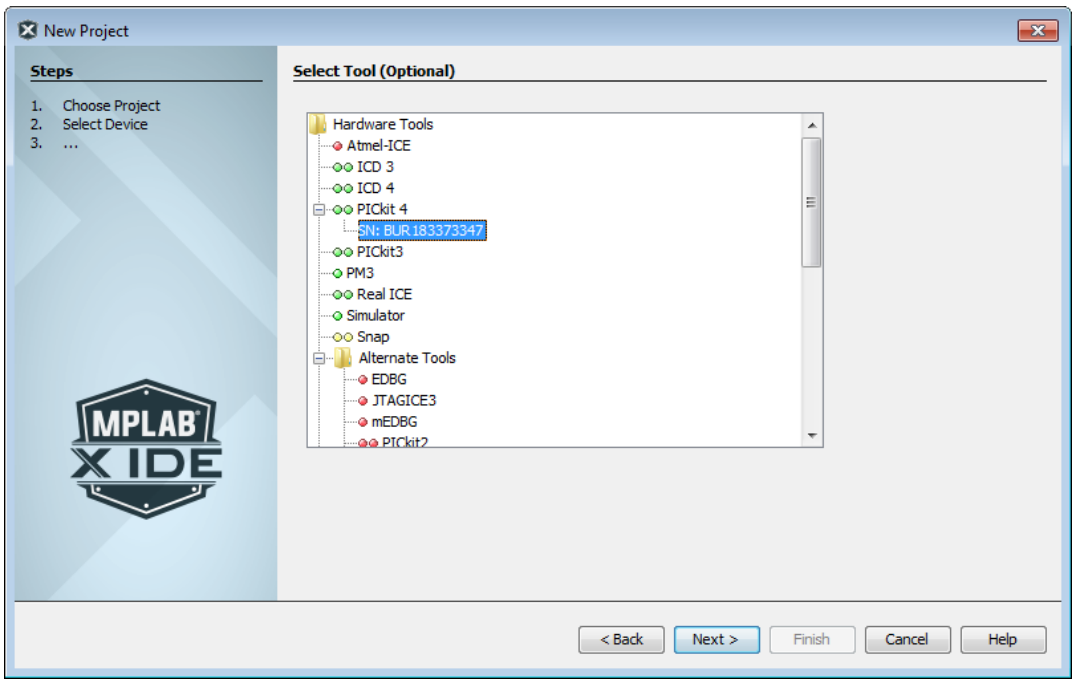
Select debug the tool.

For information on the colored circles next to tool names, see 4.1.6.1 Project Tool Support.

For hardware tools, you will notice that a serial number (SN) is specified below any tool that is connected to your computer. This allows you to select from several connected hardware tools.

Select your tool and then click **Next>**.

Figure 4-4. Project Wizard – Select Tool

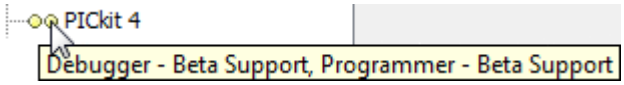


4.1.6.1 Project Tool Support

Project tool support for the project device is signified by the colored circles (lights) in front of the tool name.

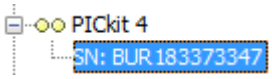
Light	Color	Support
	Green	Full (Implemented and fully tested)
	Yellow	Beta (Implemented but not fully tested)
	Red	None

If you cannot see the colors, mouse over a light to pop up text about support.



For hardware tools that can be used as either a debugger or a programmer, there are two lights next to the tool name, where the first (left-most) light signifies Debugger Support and the second light signifies Programmer Support.

Also for hardware tools, you will notice that a serial number (SN) is specified below any tool that is connected to your computer. This allows you to select from several connected hardware tools.



If you are using a third-party hardware tool, ensure you have properly installed that tool if you do not see it listed here. For more information, see 6.8 Work with Third-Party Hardware Tools.

4.1.7 Step 5: Select Plugin Board

Select the MPLAB REAL ICE in-circuit emulator plugin board. If the emulator is not chosen as your debug tool, MPLAB X IDE will not show this step.

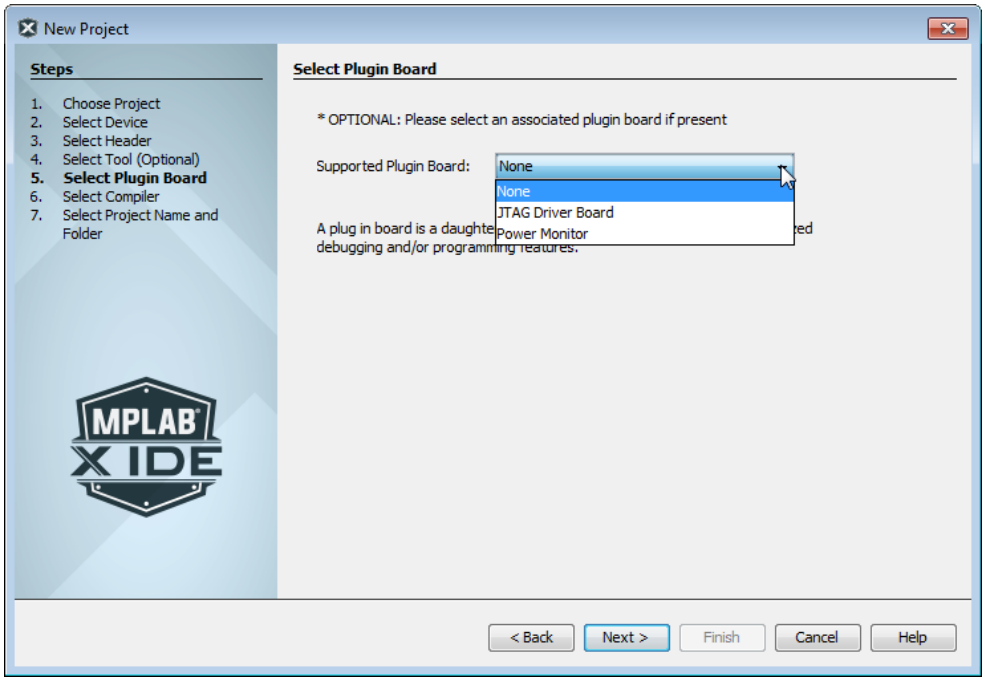
A plugin board is a circuit board that is inserted into the emulator's driver board slot to provide or add debug functionality.

Table 4-2. Emulator Plugin Boards

Supported Plugin Board	Board Description
None	Standard Communications driver board
None	High-Speed Communications driver board
JTAG Driver Board	JTAG Adapter board
Power Monitor Board	Power Monitor board (also inserts into logic probe connector)

Select your tool and then click **Next>**.

Figure 4-5. Project Wizard – Select Plugin



4.1.8 Step 6: Select Compiler

Select the language tool, either a C compiler or an assembler. Selectable language tools are show as:

ToolAbbreviation (ToolVersion) [ToolExecutablePath]

For information on the colored circles (lights) next to tool names, see 4.1.6.1 Project Tool Support.

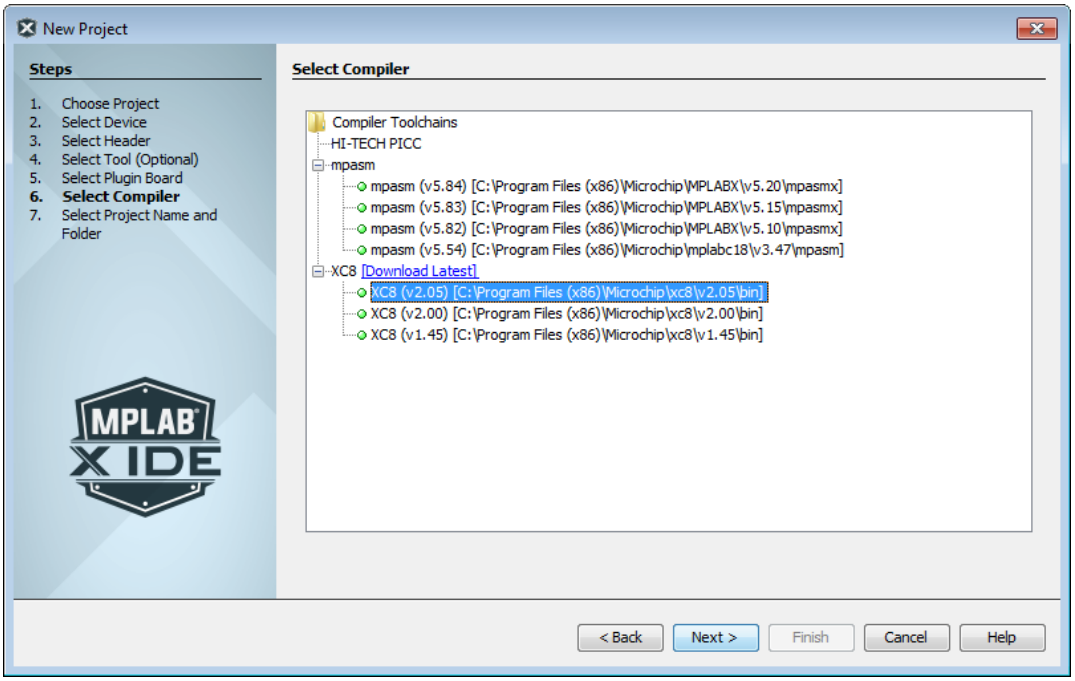
If you do not see your language tool listed or you do not see any support, ensure you have installed the tool. See 2.5 Install the Language Tools.

Note: For AVR GNU and Arm GNU toolchains, ensure the compilers are installed where MPLAB X IDE can search for them, e.g., for Windows OS, under `C:\Program Files (x86)\Microchip`. Otherwise you will have to enter the path. See 4.7 Set Language Tool Locations.

If your tool is listed under Build Tools but you do not see support, then your project device may not be supported by your tool or tool version. Consider selecting or installing another language tool that supports the device. Alternately, you can also use "Download Latest" to download and install the latest version of an MPLAB XC compiler, which may support your device.

Select your tool and then click **Next>**.

Figure 4-6. Project Wizard – Select Compiler (or Other Language Tool)



4.1.8.1 Supported Language Tools

The following tables show current and legacy MPLAB X IDE support for language tools. Current Language Tools provide new device support. Legacy tools do not.

Table 4-3. Microchip Language Tools – Current

Toolchain	Full Name	Device Support
XC8	MPLAB XC8 C Compiler for PIC MCUs	8-bit PIC® MCUs
	MPLAB XC8 C Compiler for AVR MCUs	8-bit AVR® MCUs
XC16	MPLAB XC16 C Compiler	16-bit PIC MCUs, dsPIC® DSCs
XC32	MPLAB XC32 C/C++ Compiler for PIC32M MCUs	32-bit PIC32M MCUs
	MPLAB XC32 C/C++ Compiler for PIC32C/SAM MCUs	32-bit PIC32C and SAM MCUs
Arm® GNU	Arm GNU C Compiler	32-bit Arm MCUs
AVR GNU	AVR GNU C Compiler	8- and 32-bit AVR MCUs
MPASM	MPASM Assembler, MPLINK Object Linker and Utilities	8-bit PIC MCUs

Table 4-4. Microchip Language Tools – Legacy

Toolchain	Full Name
8-Bit PIC MCU Language Tools	
C18*	MPLAB C Compiler for PIC18 MCUs
HI-TECH PICC	HI-TECH C Compiler for PIC10/12/16 MCUs
HI-TECH PICC18	HI-TECH C Compiler for PIC18 MCUs
16-Bit PIC MCU, dsPIC DSC Language Tools	
ASM30**	MPLAB Assembler, Object Linker and Utilities for PIC24 MCUs and dsPIC DSCs

.....continued	
Toolchain	**Full Name**
C30	MPLAB C Compiler for PIC24 MCUs and dsPIC DSCs
C24	MPLAB C Compiler for PIC24 MCUs (subset of C30)
dsPIC	MPLAB C Compiler for dsPIC DSCs (subset of C30)
HI-TECH DSPICC	HI-TECH C Compiler for PIC24 MCUs and dsPIC DSCs
32-Bit PIC MCU Language Tools	
C32	MPLAB C Compiler for PIC32 MCUs
HI-TECH PICC32	HI-TECH C Compiler for PIC32 MCUs
* Most compilers come with an assembler, linker, and utilities. But MPLAB C18 is supported by the MPASM toolchain. ** No longer included with MPLAB X IDE as of v1.30. Please use the assembler that comes with one of the 16-bit compilers.	

For more on each language tool, consult the language tool documentation.

For third-party language toolchains (CCS, etc.), see the "Readme for Third Party Tools.htm" file on the Start Page, "Release Notes and Support Documentation."

4.1.8.2 Download Latest MPLAB XC Compilers

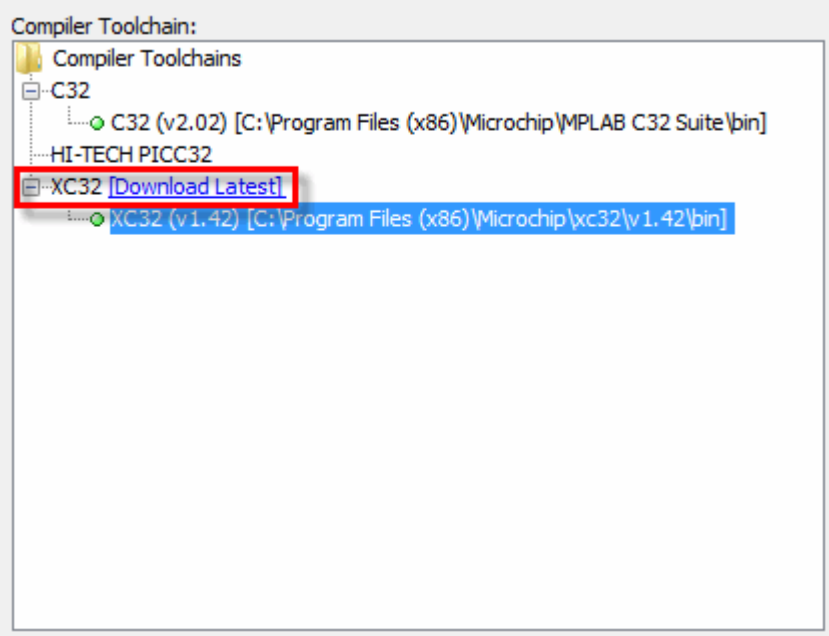
Click the "Download Latest" link to download the latest version of an MPLAB XC C compiler. Also for some operating systems, it will install the compiler and update the compiler selection tree.

The "Download Latest" link may be found in the New Project wizard under Step 6. "Select Compiler," Compiler Toolchains>XCn, where n = 8, 16 or 32.

Note: If there is no internet connection, this link will not be displayed.

Once a project has been created, the link will be available in the Project Properties window.

Figure 4-7. Download Latest



After the "Download Latest" link has been clicked:

1. A message dialog displays, telling you what will happen. You will be able to stop the download and installation at this time.
2. If you already have the latest compiler installed, a message dialog asks if you would like to continue.
3. You will be asked to select the destination of the compiler installer.
4. When downloading, there will be a progress window indicating how much of the download is left.
5. Windows and Mac OS systems: Immediately after the installer is downloaded, it begins installing the compiler. After the compiler is installed, MPLAB X IDE attempts to add the compiler to the selection tree. If it can't be added, you will be notified that you need to manually add the compiler to MPLAB X IDE (4.7 Set Language Tool Locations).
6. Linux OS systems: You must manually install the downloaded compiler. Once added, MPLAB X IDE should recognize it. If not, you will need to manually add the compiler to MPLAB X IDE (4.7 Set Language Tool Locations).

4.1.9 Step 7: Select Project Name and Folder

Select the project name, location and other project features.

Enter the project name. The name will be appended with `.X` by convention. See the "Project Folder" text box.

Browse to a folder location. You can create a new project folder if you need one. By default, projects will be placed in:

- Windows 7 and newer – `C:\Users\UserName\MPLABXProjects`
- Linux – `/home/UserName/MPLABXProjects`
- Mac – `/Users/UserName/MPLABXProjects`

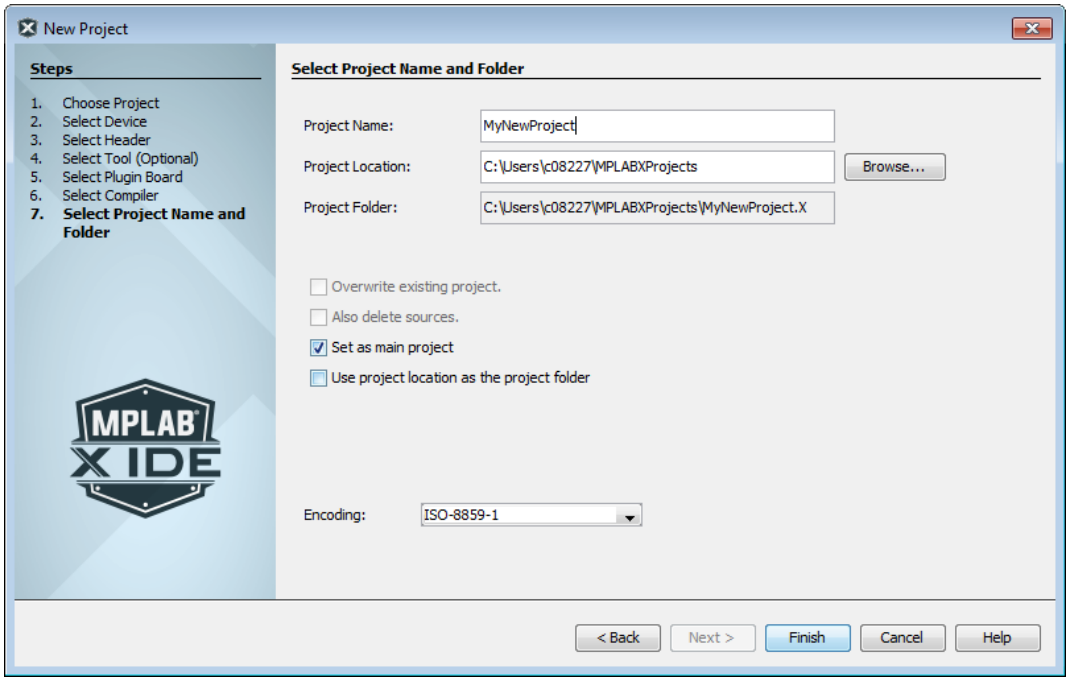
Table 4-5. Other Features

Set as main project	Specify this project as your main project
Use project location as project folder	Create the project in the same folder as the project location. This is useful if you are importing an MPLAB IDE v8 project and want to maintain the same folder for your MPLAB X IDE project (as when you want the build to be the same). MPLAB X IDE uses a folder to store project information whereas MPLAB IDE v8 uses a `.mcp` file. Therefore you can only have one MPLAB X IDE project share a folder with an MPLAB IDE v8 project (`.mcp` file).
Encoding	Select the encoding for the project. The default is ISO-8859-1 (Latin 1) character set. This selection will specify the code syntax coloring, which can be edited under *Tools>Options* (*MPLAB X IDE>Preferences* for macOS), **Fonts and Colors** button, **Syntax** tab.

When you are done, click **Finish** to complete new project creation.

Note: You can change a Standalone (Application) project to a Library project by changing the Configuration Type. For details, see 4.10.1 Change Project Configuration Type.

Figure 4-8. Project Wizard – Select Project Name and Folder



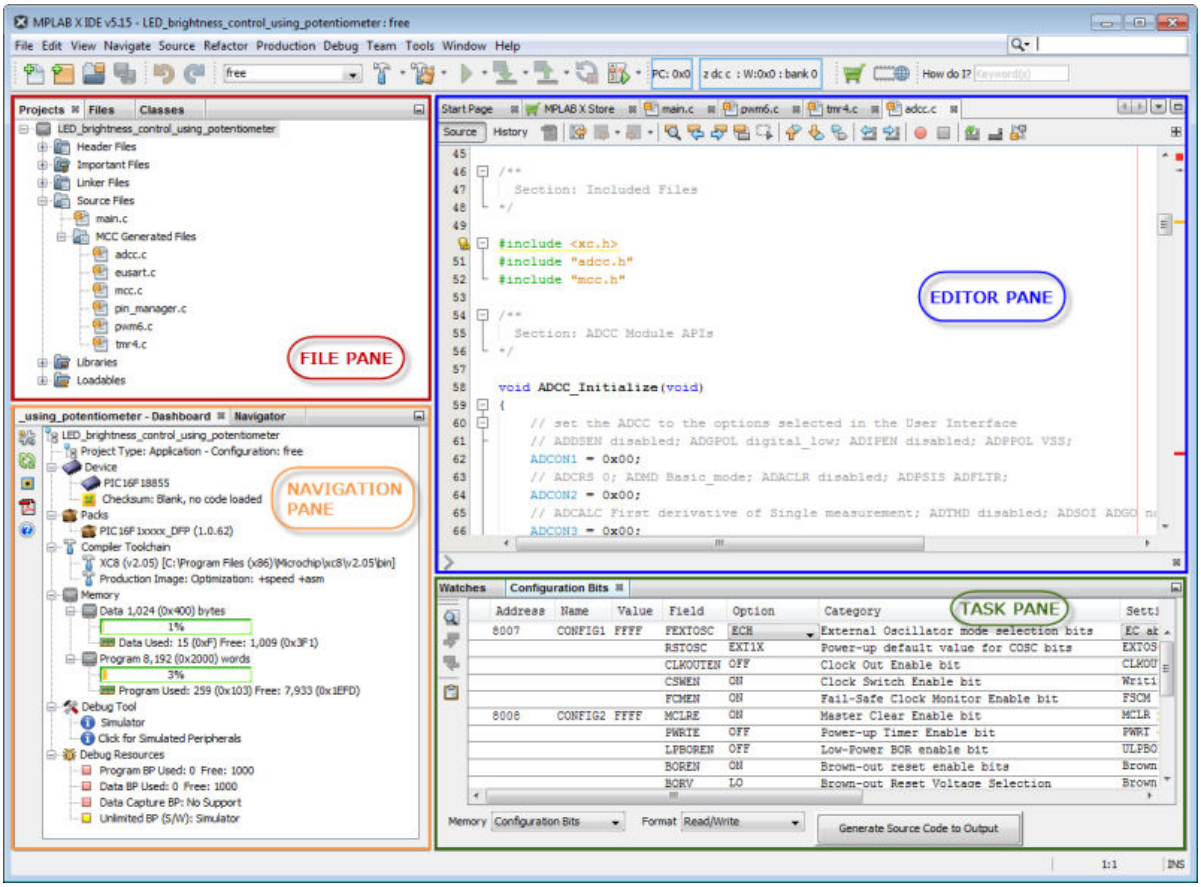
4.2 View Changes to Desktop

The MPLAB X IDE desktop is divided into four **panes**. When you have created your project, several **windows** will open in these panes.

Desktop Panes

- **File pane** – displays file-related information.
  - The Projects window displays the project tree with files grouped by category.
  - The Files window displays the project files according to the folder organization on your computer.
  - The Classes window displays any classes and their functions, variables and constants in the code. Double click on an item to see its declaration.
- **Navigator pane** – displays information on items in the File pane.
  - The Dashboard window displays information about the selected project.
  - The Navigator windows displays information on the symbols and variables in the project file selected.
- **Editor pane** – for viewing and editing project files. The Start Page and MPLAB X Store are also visible here.
- **Task pane** – displays task output from building, debugging, or running an application.

Figure 4-9. MPLAB X IDE Desktop - Panes



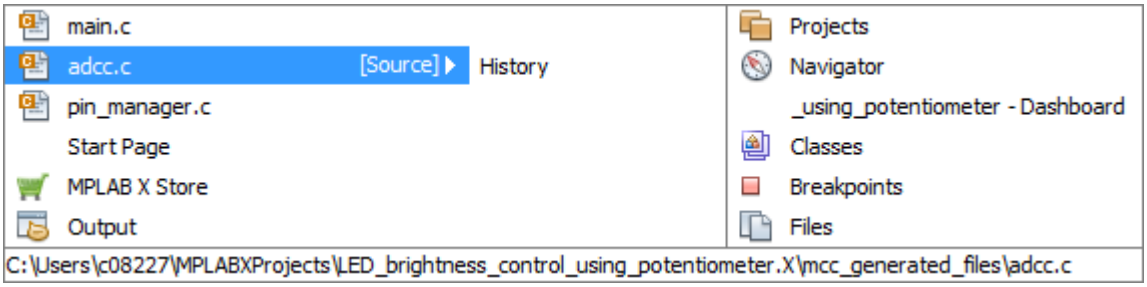
Desktop Windows

Although panes are useful for describing sections of the IDE interface, windows may be moved from one pane to another. So panes do not always represent the window content. Therefore, only windows are discussed in most documentation.

To move around between windows on the desktop:

- Click on a window to make it in focus.
- While in one window, you can change focus to any other open window by pressing in holding <Ctrl> and then pressing <Tab>. Select the window you want from a pop-up list.

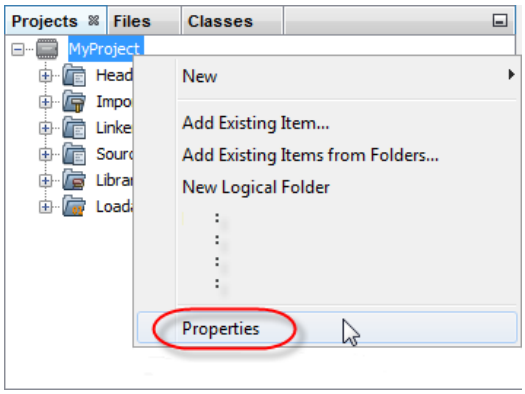
Figure 4-10. Switch Window Focus



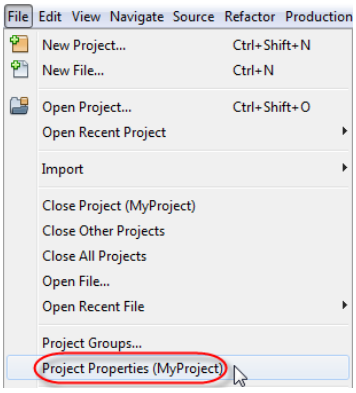
4.3 Open Project Properties

Once a project has been created, you can view or change the project properties in the Project Properties window. Access this window by performing one of the following actions.

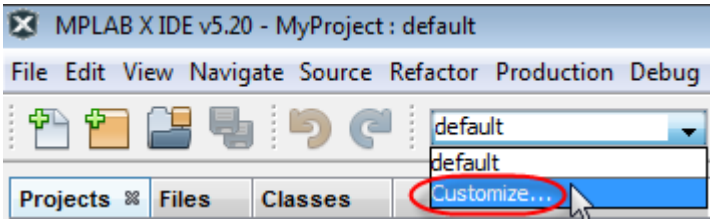
1. Right click on the project name in the Projects window and select "Properties."



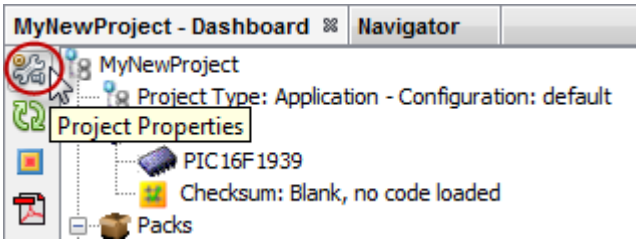
2. Click on the project name in the Projects window and then select *File>Project Properties*.



3. Click on the project name, in the Projects window, then click on the "Set Project Configuration" drop down menu to select "Customize."



4. Click on the "Project Properties" icon on the Dashboard window.



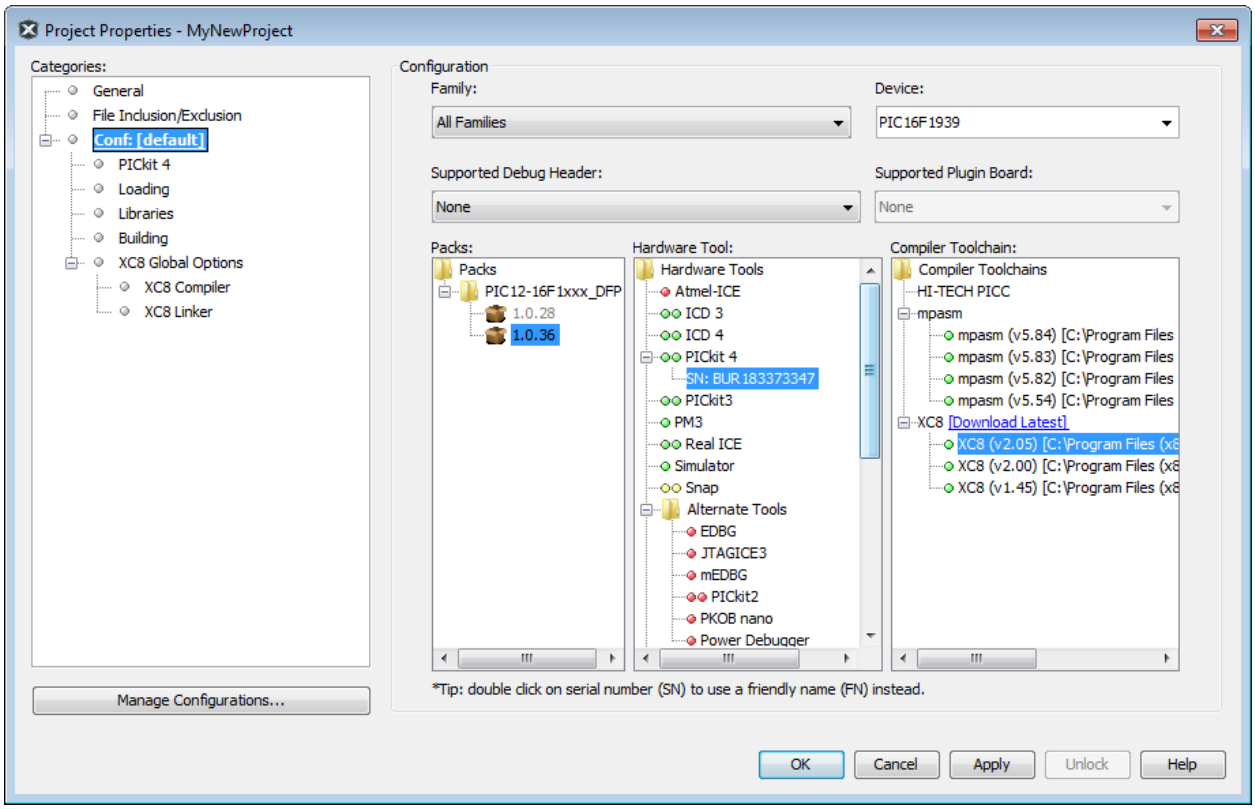
4.4 View or Make Changes to Project Properties

Click the "Conf: [default]" category to reveal the general project configuration, such as the project device, related debug/programmer tool, and language tool.

Categories	The project configuration is set as "default" when created. You can create different configurations by clicking the **Manage Configurations** button. See 6.4 Work with Multiple Configurations.

Device	This should be the device simulated or the device on the board. Care should be taken when changing the device, especially to one from a different family, as the pack, hardware tool and compiler will need to be updated.
Packs	This is the device family pack (DFP) version for the selected device. The latest version of MPLAB X IDE will contain the latest DFP version. However, you may change to a different version, if available. See 5.1 Work with Device Packs.
Hardware	A supported hardware debug tool (or Simulator) should be selected. See 4.1.6.1 Project Tool Support.
Compiler Toolchain	A supported language tool (usually compiler) should be selected. See 4.1.6.1 Project Tool Support. To get the latest version of an MPLAB XC C compiler, click the "Download Latest" link. See 4.1.8.2 Download Latest MPLAB XC Compilers.

Figure 4-11. Project Properties Window - Default Configuration

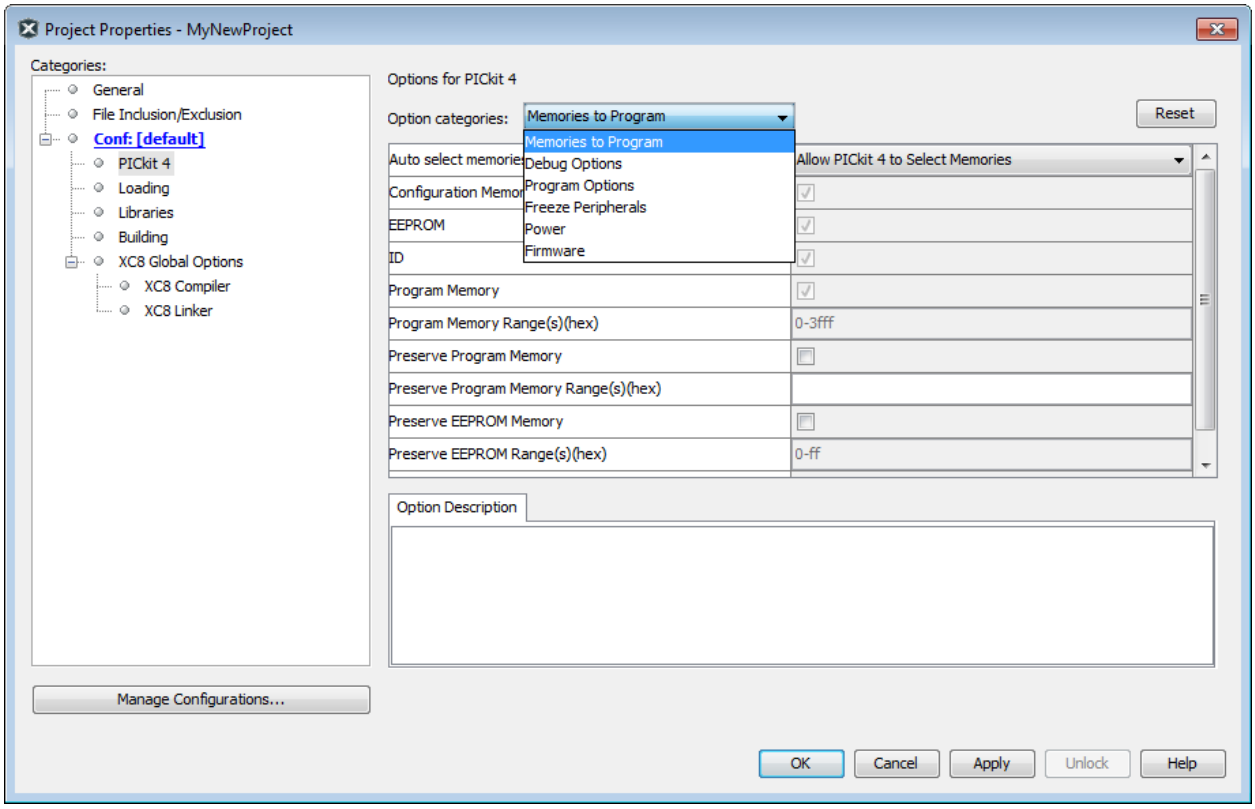


4.5 Set Up or Change Debugger/Programmer Tool Options

Set options for your tools in the Project Properties window.

Click on your hardware tool or simulator (beneath Conf:[default]) to see related setup options. For more on what these options mean, see your tool documentation.

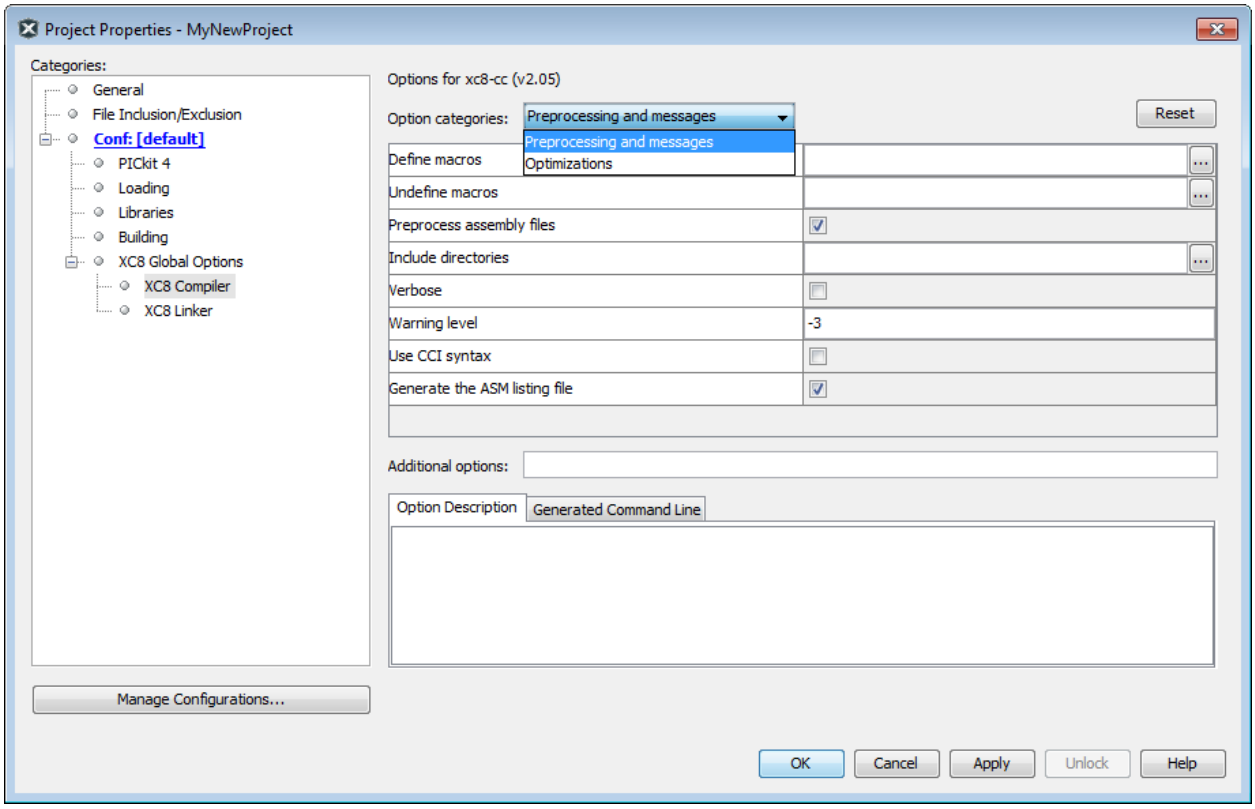
Figure 4-12. Example Tool Setup Page



4.6 Set Up or Change Language Tool Options

Click on your language tool to see related setup options in the Project Properties window. For more on what these options mean, see your language tool documentation.

Figure 4-13. Language Tool Setup Page



IDE Options vs. Command Line Options

Options entered into the IDE may differ in format from those entered on the command line. Click on an option name to see more information on that option in the "Option Description" tab. Also, after entering or selecting option data, click the "Generated Command Line" tab to view its contents and ensure the command line code generated by the option is what you expected.

Global Options

Set global options that apply to all language tools in the language toolsuite. Also use the "User Comments" tab to enter information about why certain options were selected.

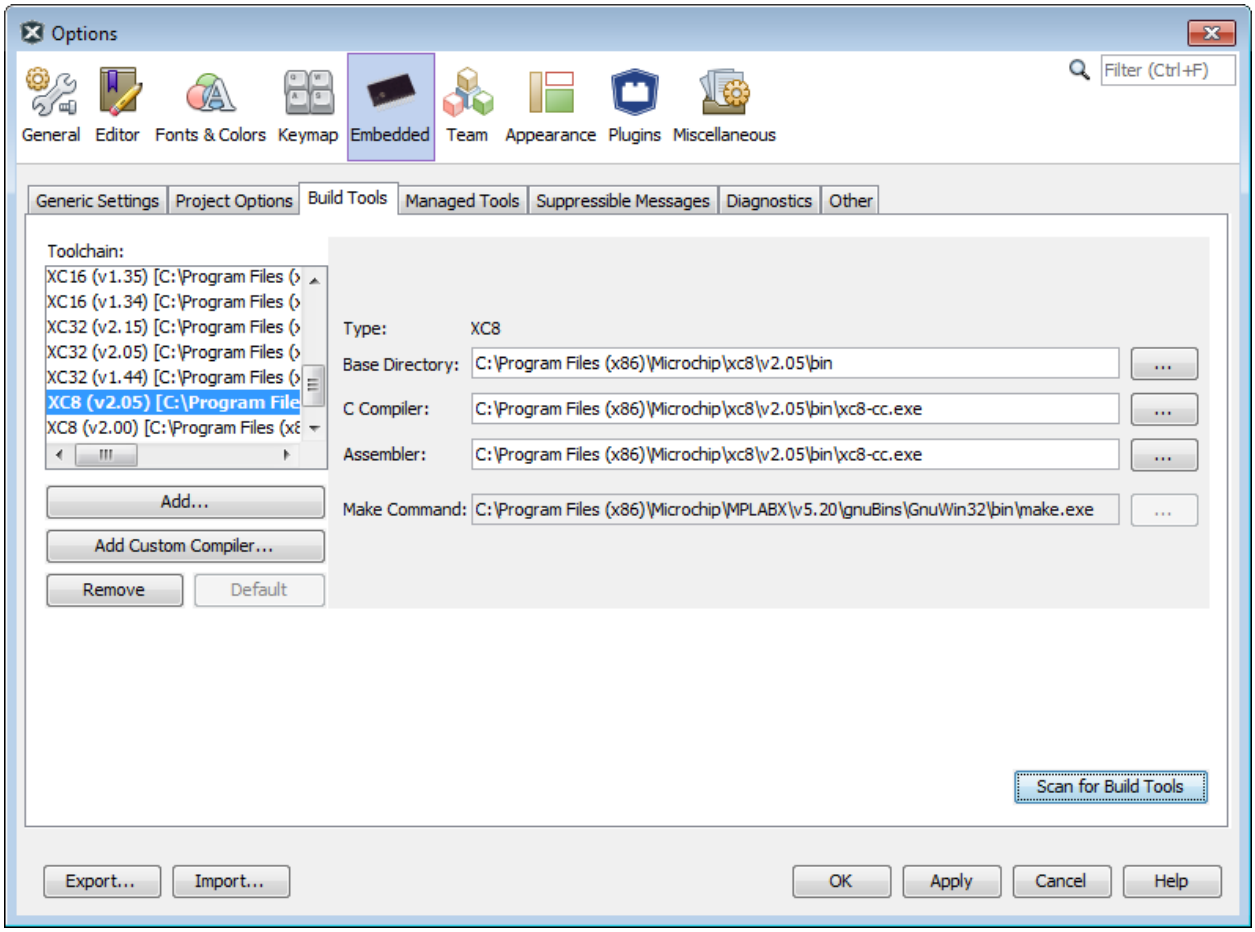
4.7 Set Language Tool Locations

To see which language tools are available to MPLAB X IDE, and to view or change their paths, use the following instructions:

- For macOS – Access the build tools from:
  MPLAB X IDE>Preferences>Embedded>Build Tools from the main menu bar.
- For Other OS – Access the build tools from:
  Tools>Options>Embedded>Build Tools.

The window should automatically populate with all installed toolchains. See also 12.16.3 Build Options Tab.

Figure 4-14. Language Tool (Compiler) Locations



4.7.1 Add or Change a Toolchain

If you do not see your tool listed under "Toolchain," try the following features:

- Scan for Build Tools – scan the environment path and list the language tools installed on the computer.
- Add – manually add the tool to the list by entering the path to the directory containing the tool executable(s), i.e., base directory. Typically, this is the bin subdirectory in the tool installation directory.

If you have more than one version of a compiler available, select one from the list.

To change the path of a tool, enter the new path or browse for it.

4.7.2 Remove a Toolchain

To remove an existing toolchain:

1. Click to select the toolchain from the list.
2. Click the Remove button.

This will NOT uninstall the compiler from your computer; it will only remove MPLAB X IDE awareness of this compiler. To add the toolchain back, see Section 4.7.1 "Add or Change a Toolchain."

If you want to uninstall a compiler from your computer, remove the reference to the compiler in MPLAB X IDE first, as specified above. Then uninstall the compiler. If you uninstall the compiler first and then start MPLAB X IDE, the toolchain paths will appear in red. To remove the reference to this uninstalled compiler, follow the steps above.

4.7.3 About Toolchain Paths

MPLAB X IDE searches for toolchains under default installation paths and the PATH environment variable. An example default path for MPLAB XC16 under a Windows 64-bit operating system is:

```
C:\Program Files (x86)\Microchip\xc16\vx.xx\bin
```

When you install a compiler, you have the choice to determine the location. You can either:

- install at the default location

OR

- install at a different location and add the location to the PATH

If you are installing at a location different from the default, you should add that location to the PATH.

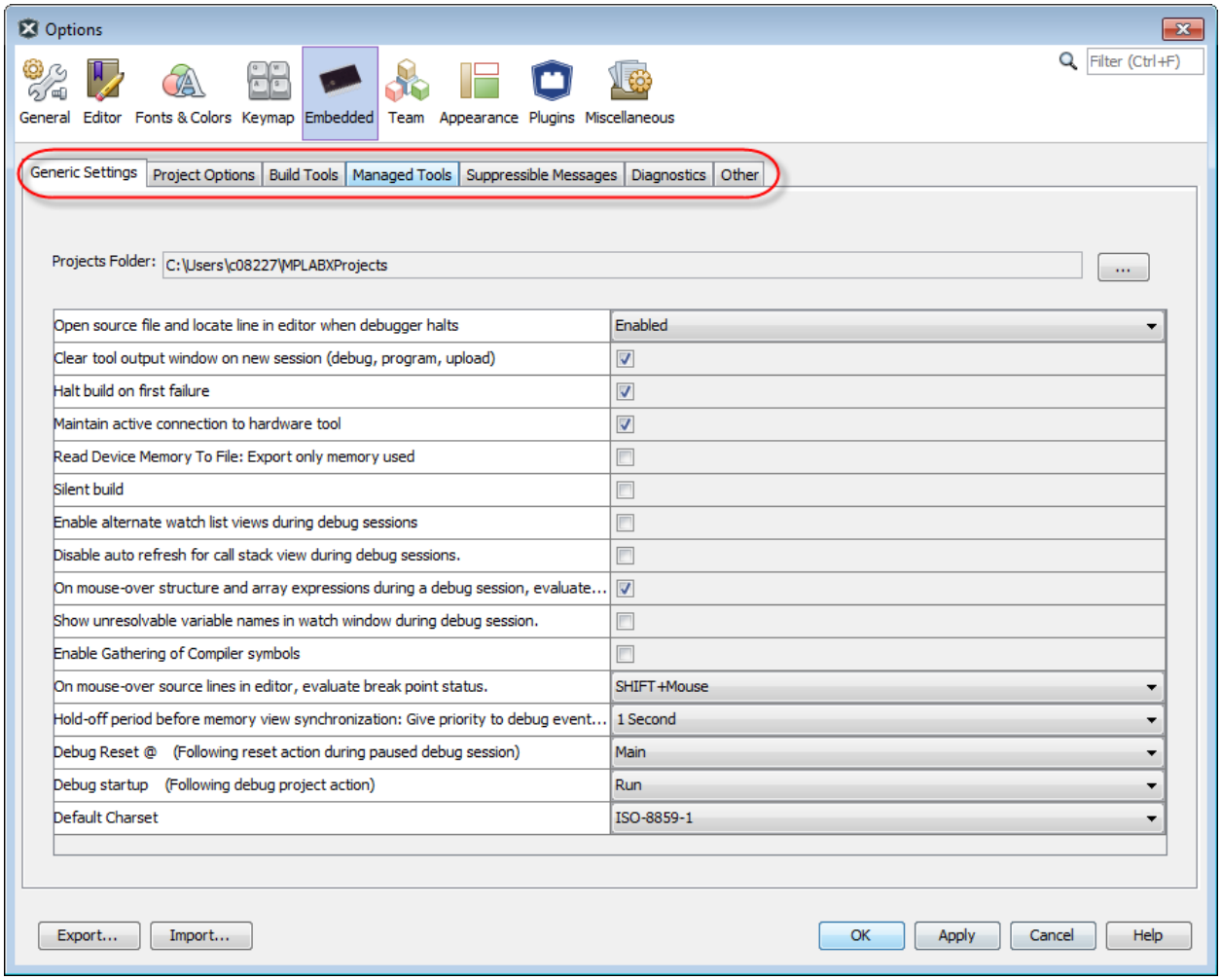
If you do not choose to have the installer add a new location to the PATH, you can point MPLAB X IDE to the compiler location in Tools>Options, Embedded button, Build Tools tab.

Finally, you can manually add the new location to the PATH variable.

If the toolchain is known to MPLAB X IDE, but the path cannot be found, the path will appear in red text.

4.8 Set Other Tool Options

In addition to the build paths, you may set up other Embedded options. Select from the tabs in the Options window, Embedded category. See 12.16 Tools Options Window, Embedded.



4.9 Add Files to a Project

Once you have created a new project, you will need to add files to it to create your application. You can create new, empty files in which to write your code or you can import or reference existing files. The editor can be used to create or update any file contents, making use of language-tool specific color-coding and other features.

File Link Order

The order in which you add files to the project will determine their link order. If you have imported a project from MPLAB IDE v8, the order will be determined by the `.mcp` file. A new file added to the project will be linked last. If you remove a file and then re-add it, it moves to the end of the link list.

Library and Object File Linking

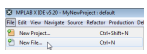
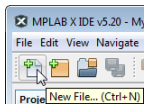
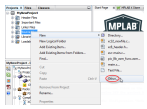
You can reference library and object files to be used by the linker.

Exclude Files from the Build

You can set file or folder level options to exclude files from the build.

4.9.1 Launch New File Wizard

To create a new, empty file to add code to your project, you can use the New File wizard. The wizard can be launched in several ways.

	*File>New File* (or Ctrl+N)
	New File icon (on the File toolbar)
	Right click on a project folder (e.g., "Source Files") in the Projects/Files window and select *New>Other*.

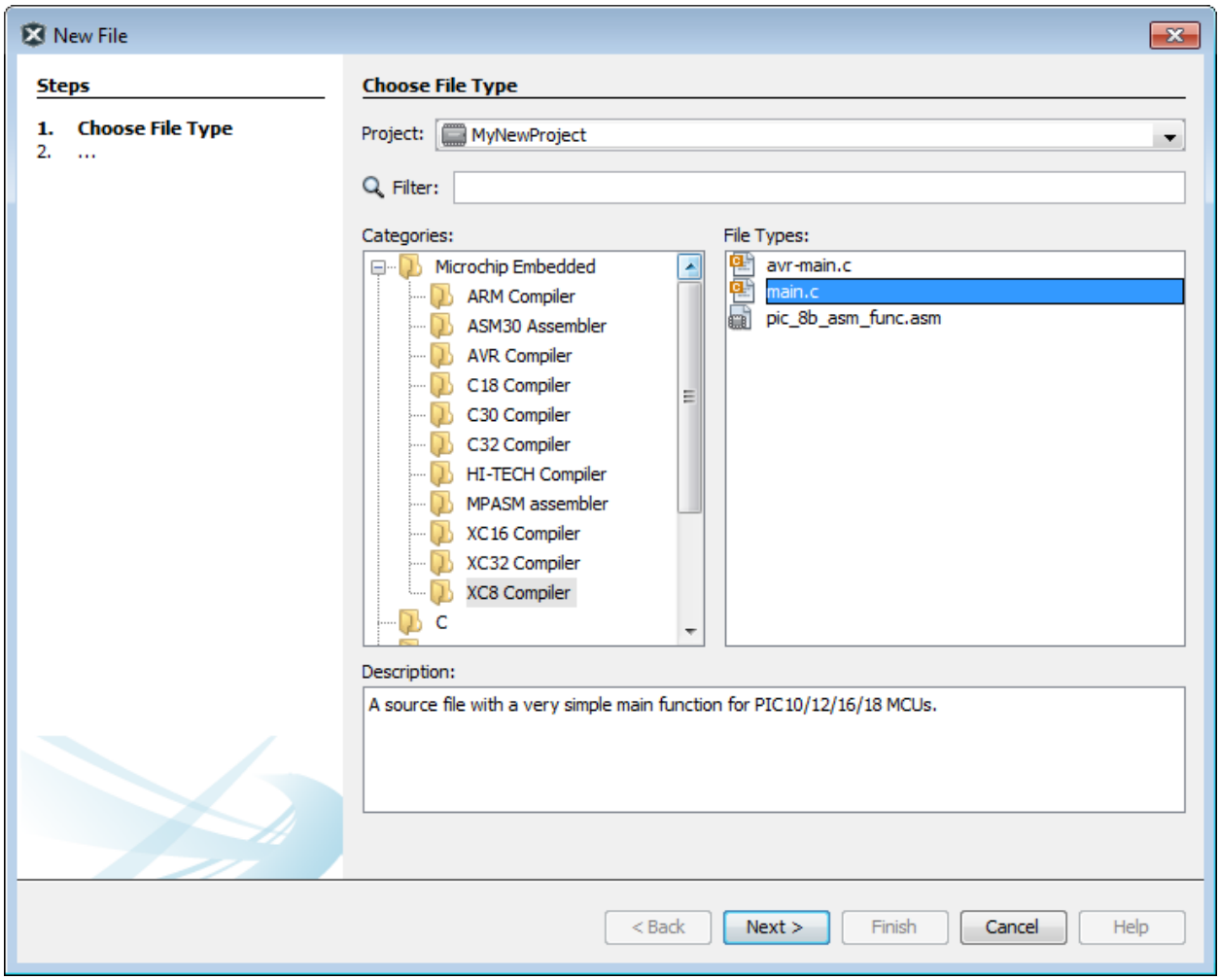
4.9.2 Create a New File

Create a new file by following the steps in the New File wizard.

Step 1. Choose File Type

Choose the file category by expanding "Microchip Embedded" to find an appropriate selection. Then select a file type. Click **Next>**.

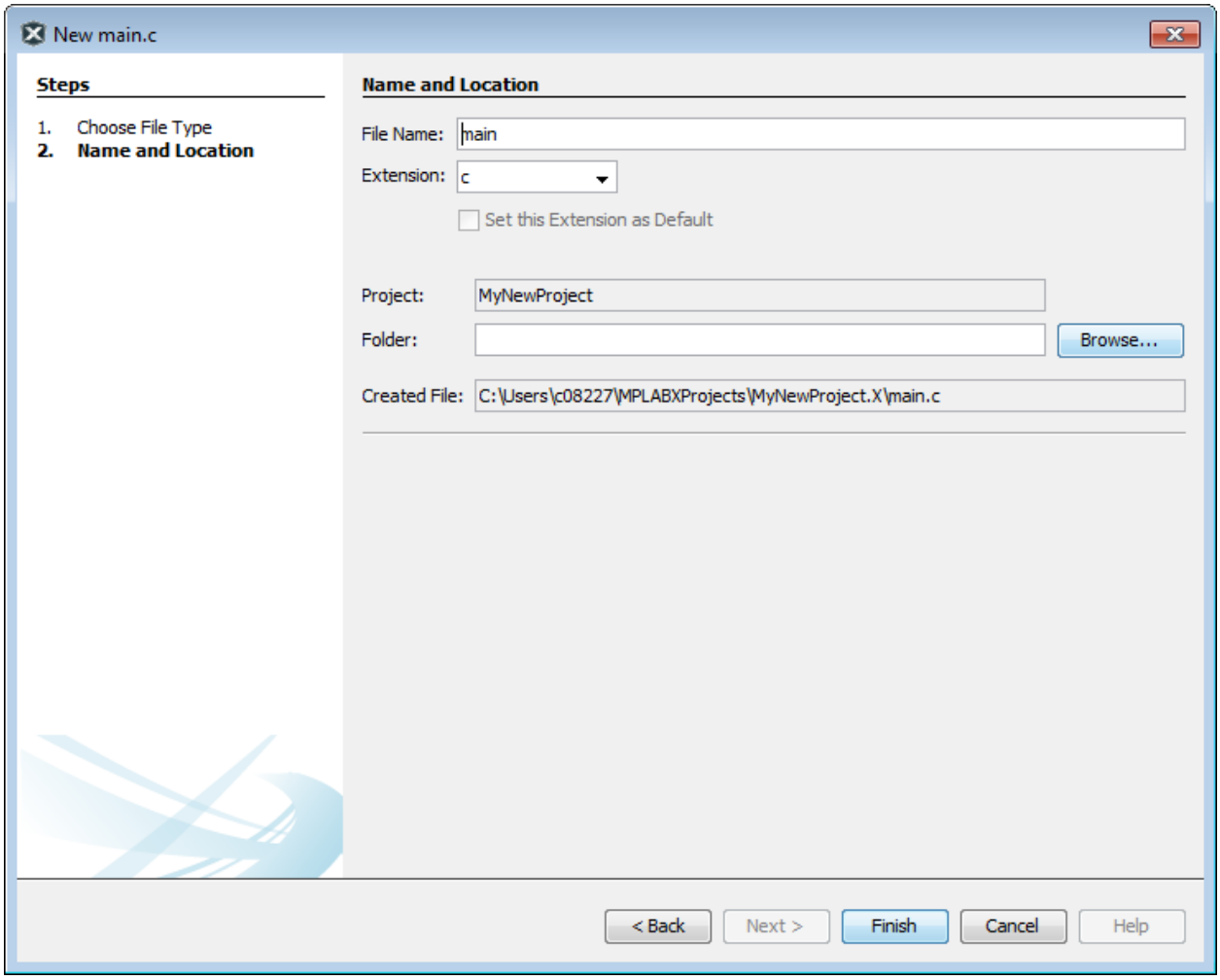
Figure 4-15. File Wizard – Choose Category and Type



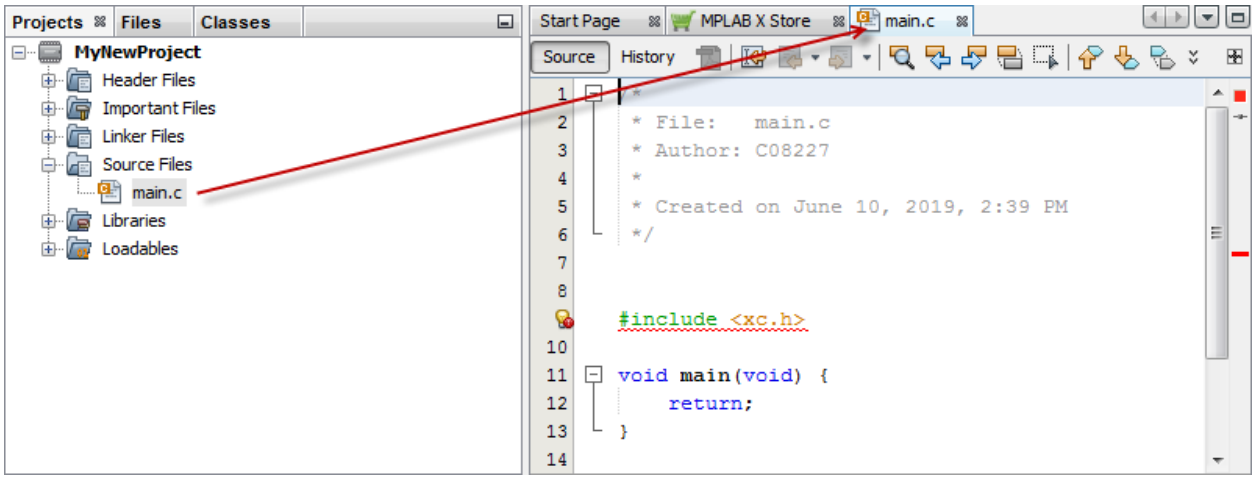
Step 2. Name and Location

Name the file and place it in the project folder. Click **Finish**.

Figure 4-16. File Wizard – Choose Associated Project



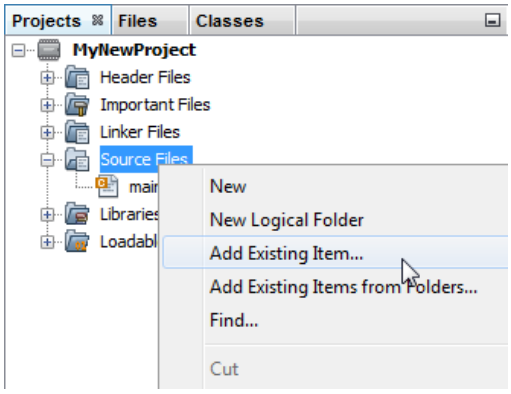
An Editor tab will appear with the file's name. Enter your file content under this tab. The text in the file will have syntax color based on its file type.



4.9.3 Add Existing Files to a Project

Existing files can be added to a project by doing one of the following:

- Right click on the project in the Project/File window and select "Add Existing Item." Once added, you can drag and drop the file into a folder if you wish.
- Right click on a folder (e.g., "Source Files") in the Project/File window and select "Add Existing Item."



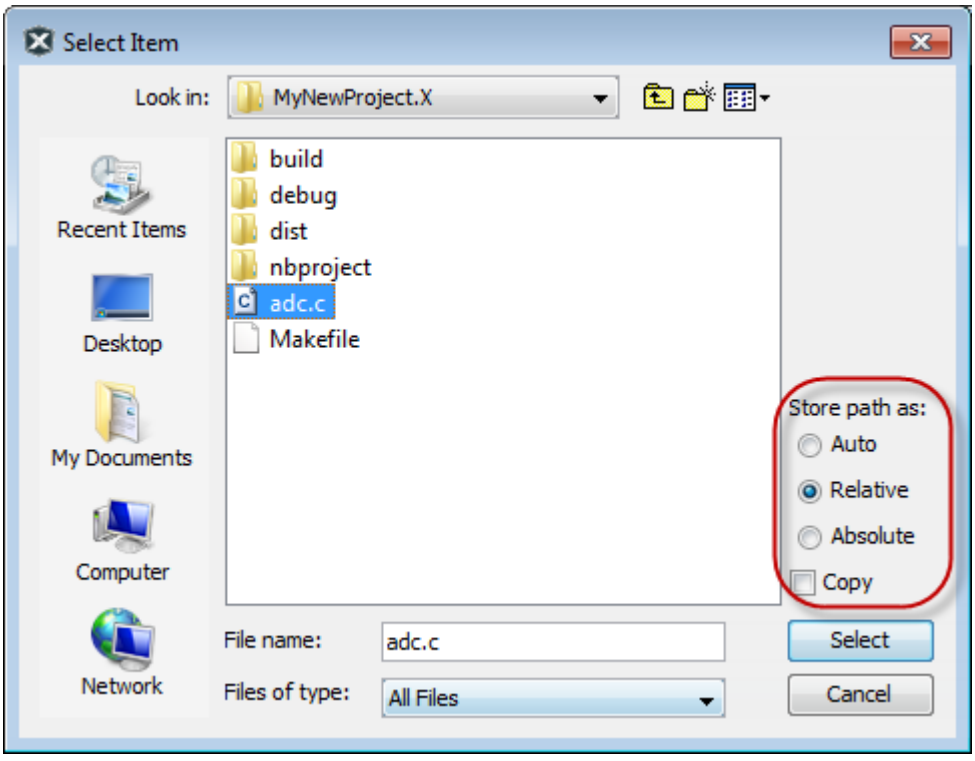
4.9.3.1 Add Files in the Project Folder

When adding a file that is located under the main project folder, you can choose whether to add it as:

- Auto – let MPLAB X IDE decide how best to locate the file
- Relative – specify the file location relative to the project (the most portable project)
- Absolute – specify the file location by an absolute path
- Copy – copy the specified file to the project folder

Once you select a path mode, MPLAB X IDE will remember this setting. To change it, see "File Path Mode" under in 12.16.2 Project Options Tab.

The file will appear in the Projects window. An Editor tab with the file's name will appear also.



4.9.3.2 Add Files Outside the Project Folder

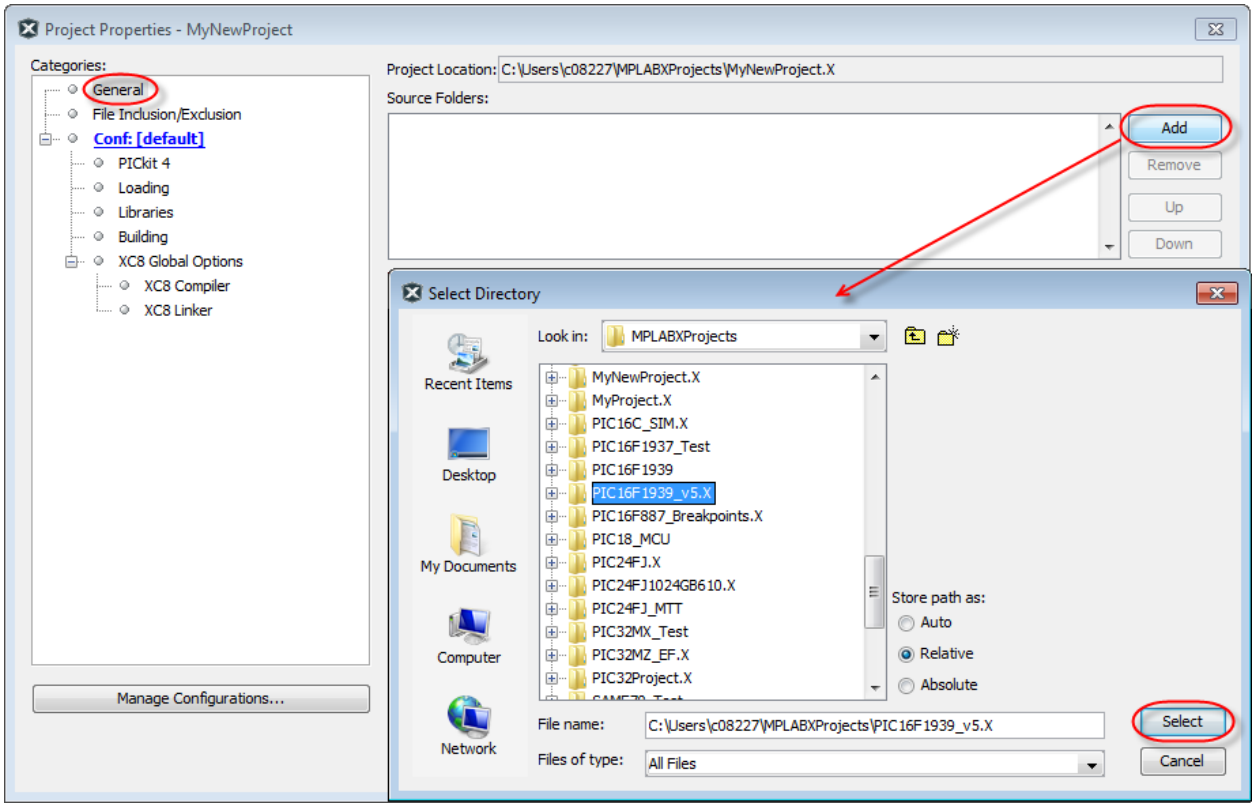
When adding a source file that is not in the project folder, add the file (store the path) as "Relative." This creates an external folder (`_ext`) so the project can find the file when it is building.

To make use of navigation features such as the `//TODO` action item and file context menus, you must tell the project where the files are located. To do this:

1. In the Projects window, right click on the project name and select "Properties."
2. Under "Categories," click "General."
3. Next to "Source Folders," click Add.
4. Browse to the path of the external file(s) you have added to the project. Click the **Select** button.
5. Click **Apply** or **OK**. Then rebuild the project.

When you import an MPLAB IDE v8 project, the source files are a not in the project folder. Use *File>Import>MPLAB IDE v8 Project* to automatically import the files.

Figure 4-17. Specify External File Path



4.9.4 Edit Code in Files

To create new code or change existing code, use the NetBeans editor. For more information about the editor, see the NetBeans Help topic:

https://netbeans.org/features/java/editor.html

Also see MPLAB X IDE-related information about this editor:

7. Editor

4.9.5 Add and Set Up Library and Object Files

You can reference library files to be used by the linker in the following locations:

- the Libraries folder in the Projects window
- the Project Properties window

Additional library file set up can be done in the language tool librarian.

4.9.5.1 Libraries Folder

In the Projects window, right click on the Libraries folder to see these options:

- Add Library Project
  Make a project that produces a library as its output required for your project, i.e., establish a dependency. For more on this, see the NetBeans Help topic:

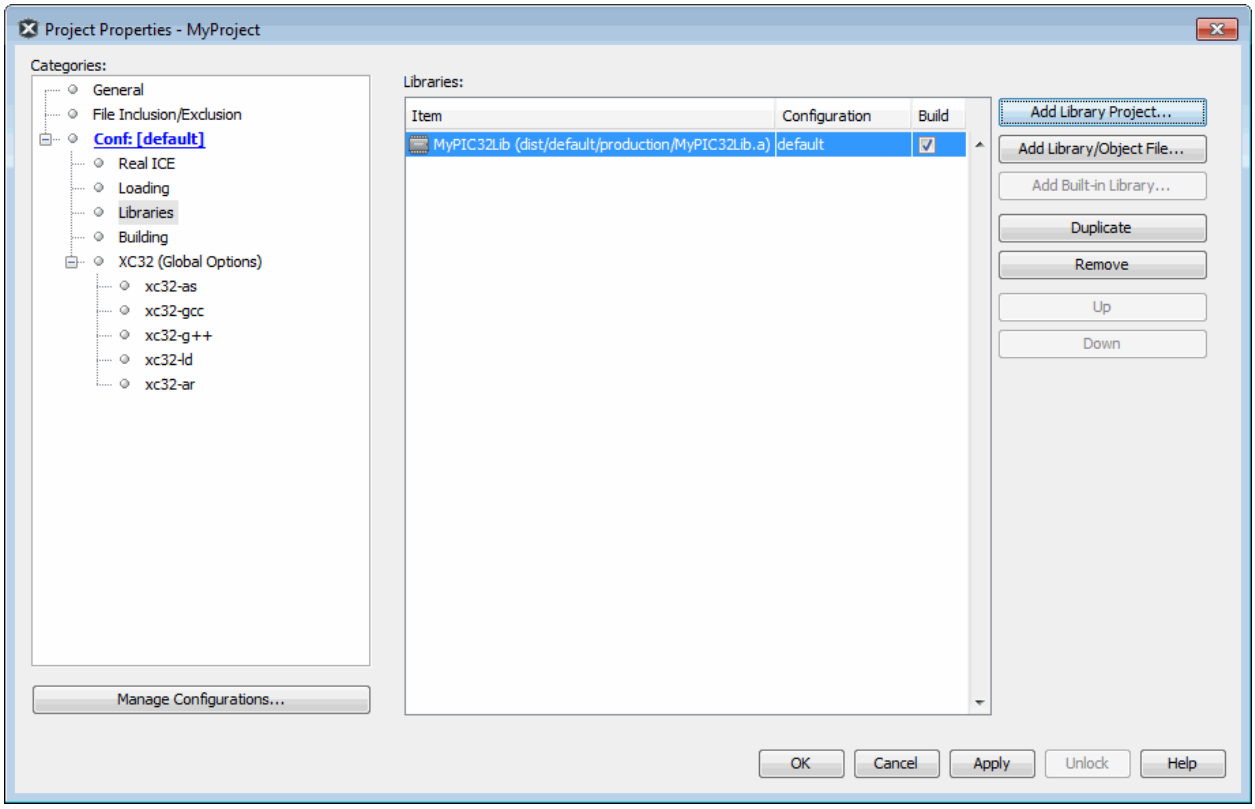
  https://netbeans.org/kb/73/java/project-setup.html#projects-dependencies.
- Add Library/Object File
  Add an existing library file or pre-built (object) file to the project.
- Properties
  Open the project's Properties window to select additional options.

4.9.5.2 Project Properties Window: Libraries Category

Open the Project Properties window and click on the "Libraries" category. Any files added to the Libraries folder in the Projects window will be visible here. You may add additional files and manage these files using the buttons on the right.

The order of files in the library list determines link order, which is important when the libraries included in the project reference each other. Libraries referencing objects from a secondary library should be placed higher in the link order list. An incorrect link order may result in an "undefined reference to" error during linking. For details on link order, see language tool documentation.

Figure 4-18. Manage Library/Object Files and Set Up Link Order

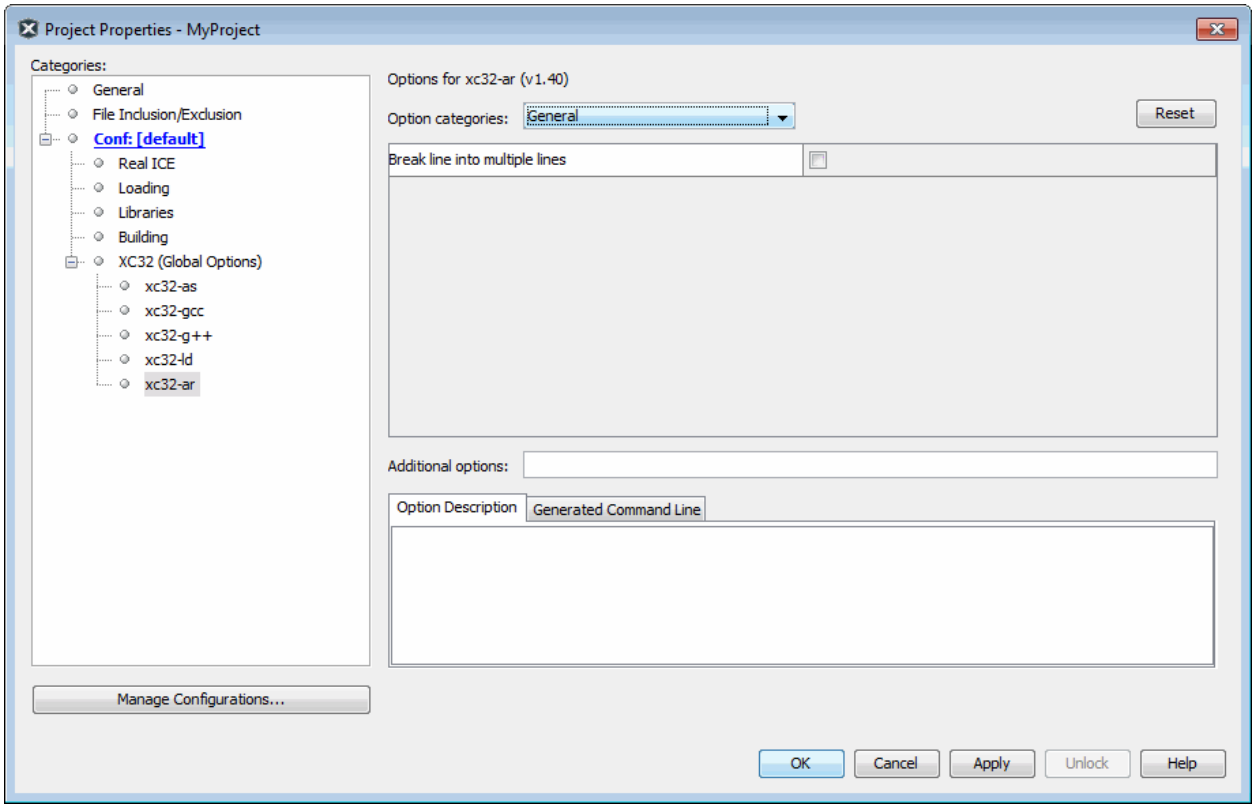


4.9.5.3 Project Properties Window: Librarian Category

Open the Project Properties window and click on the librarian/archiver under your language tool category. Consult your language tool documentation to determine the executable name of your librarian.

In the left pane, click on the name of the linker. Then, in the right pane, select the option "Libraries." Choose library options from this list.

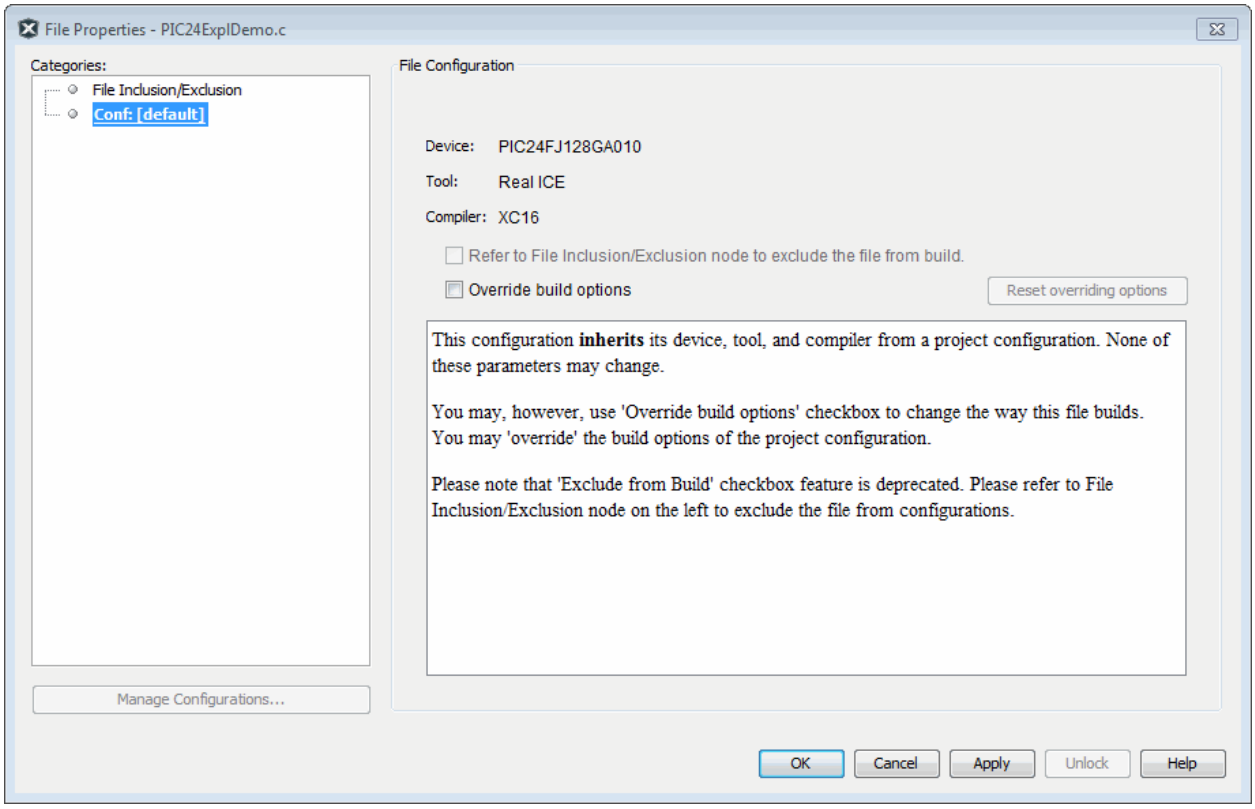
Figure 4-19. Set Up Libraries Options in the Librarian



4.9.6 Set File and Folder Properties

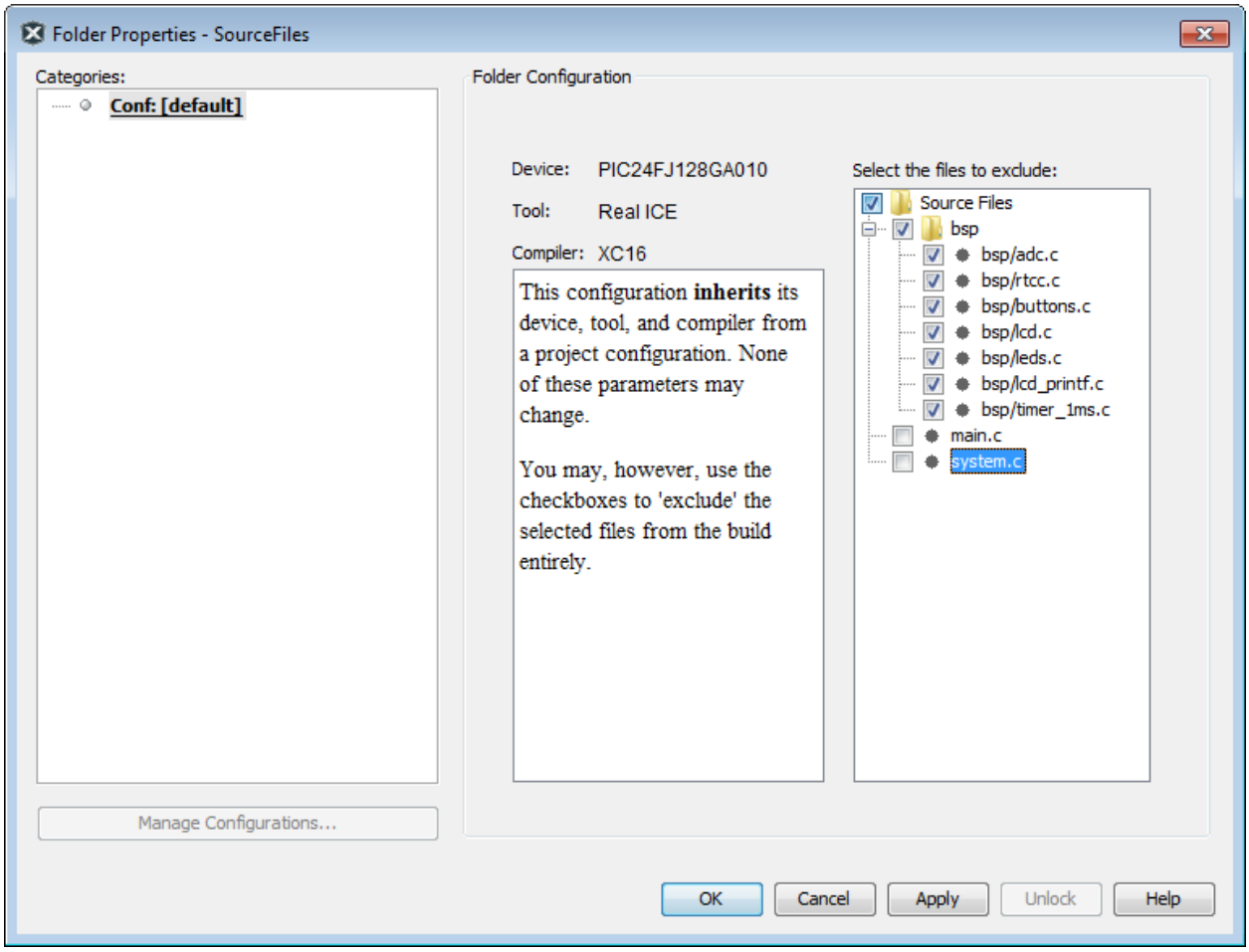
A project file can be built differently from other files in the project using the File Properties dialog. Access this dialog by right clicking on the file name in the Projects window and selecting "Properties." Select to exclude the file from the project build or override the project build options with the ones selected here. To override build options, check the checkbox and then click "Apply" to see selection options appear under "Categories."

Figure 4-20. File Properties



An entire (virtual) folder can be excluded from the build process by right clicking on the folder name in the Projects window and selecting "Properties." Select to exclude folders or individual files from the build. Selecting a folder that is above other folders will select the entire contents of that folder. You can deselect files or other folders as you wish.

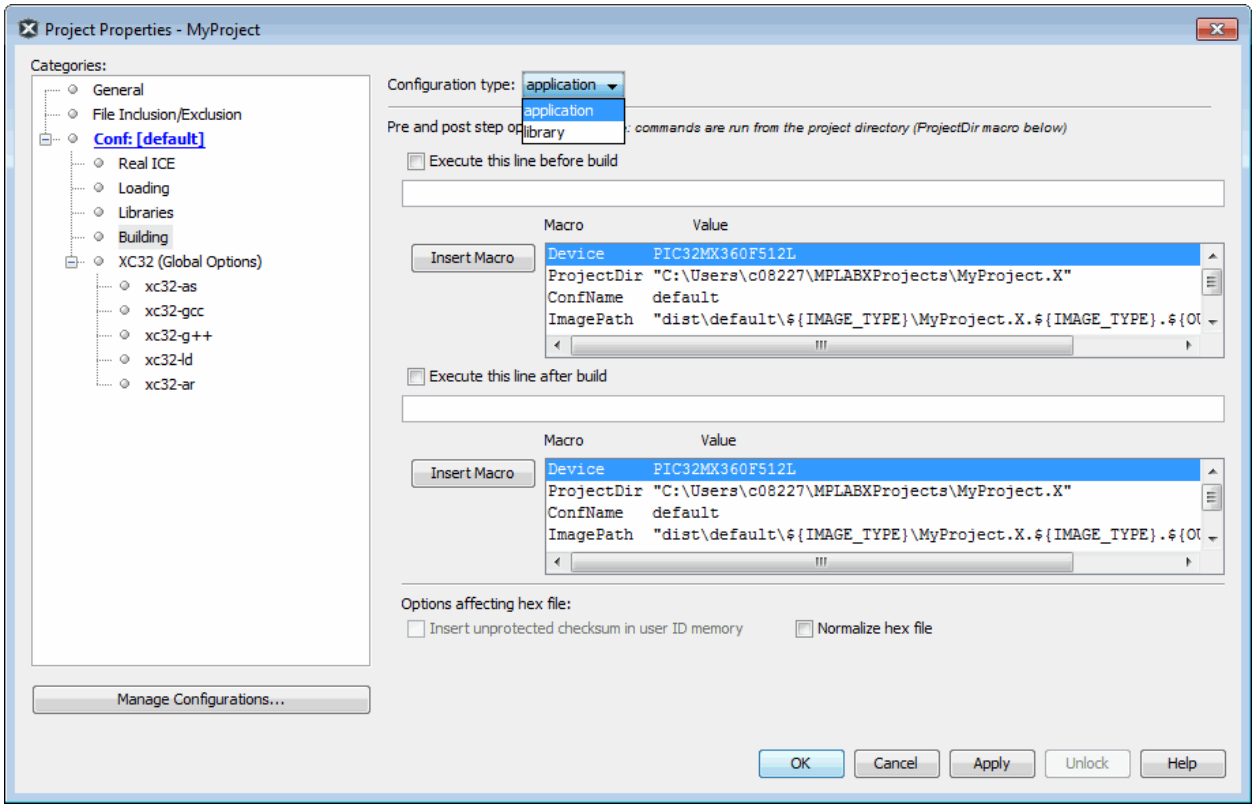
Figure 4-21. Folder Properties



4.10 Set Build Properties

Before building the project, you may want to set up additional build properties in the Project Properties window, Building category.

Figure 4-22. Make Options Build Properties



4.10.1 Change Project Configuration Type

As of MPLAB X IDE v2.15, when you create a Standalone or Library project in the New Project wizard, you are creating a project with either a configuration type "application" or "library" respectively. This means you can switch the configuration type of your existing project if you wish, with the caveats specified in the following sections.

Prebuilt and User Makefile projects are unaffected by this feature (i.e., they may not be changed to a different type of project once created).

Switch from a Standalone to a Library Project

When switching project configuration type from "application" to "library," it is your responsibility to remove `main()`. Also, any loadables in the "application" project will not be built/linked once the configuration type is "library." For more on loadables, see 5.5 Loadable Projects, Files and Symbols.

If the toolchain (i.e., XC8) does not support libraries, then the combo box will be disabled and you cannot switch the configuration to "library."

Switch from a Library to a Standalone Project

When switching project configuration type from "library" to "application," it is your responsibility to add `main()`.

4.10.2 Execute this line before/after the build

Type in a command to be executed at the very beginning or at the very end of the build process. These commands are inserted into the `nbproject/Makefile-$CONF.mk` file and allow you to customize the build process. If you need to refer to some of the project-related items (like the image name) in the script or program you are calling, use the supplied macros.

You can type the macros yourself or click **Insert Macro** to copy the macro name into the current position in the edit box. Commands are run in the make process with the current directory being set to the MPLAB X IDE project directory. The project directory is defined as the project that contains the `nbproject` folder

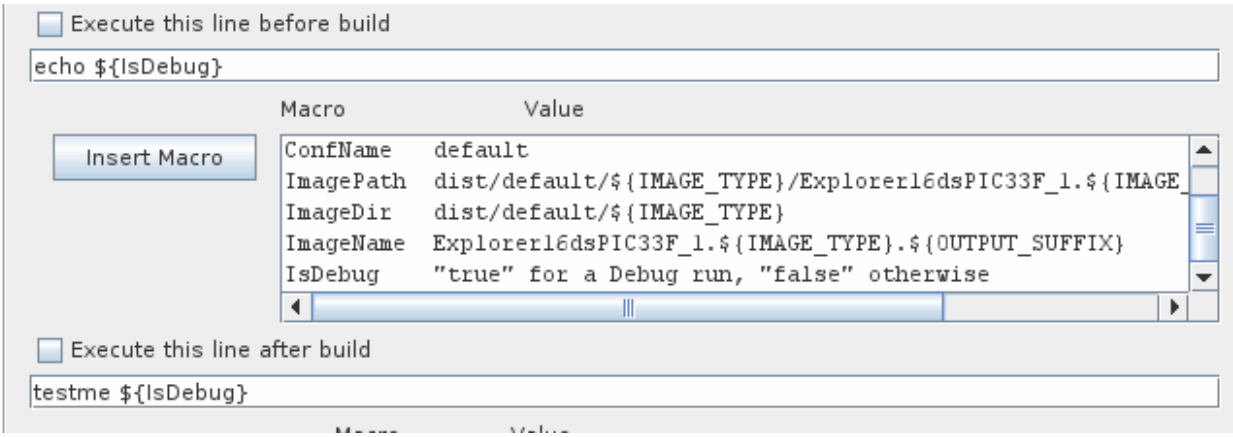
Table 4-6. Macros

Name*	Function
Device	device for the currently-selected project configuration
ProjectDir	location of the project files on the PC
ConfName	name of the currently-selected project configuration
ImagePath	path to the build image
ImageDir	directory containing the build image
ImageName	name of the build image
IsDebug	"true" for Debug mode; "false" otherwise
* Additional macros may be available, depending on which compiler is used.	

Example: Execute a Process Only During Debug

On the window shown below, click the "IsDebug" macro to select it, then click **Insert Macro** to insert it into a script.

Figure 4-23. Build Properties – Pre/Post Build



Check the value of `$1` (Linux or Mac OS) or `%1` (Windows OS) in the script being run.

Bash Code Example

```
#!/bin/bash
echo is $1
if [ "$1" == "true" ]; then
echo We are in debug
else
echo We are in production
fi
```

Batch Code Example

```
@echo off
if "%1" == "true" goto debug
:production
@echo We are in production
goto end
:debug
@echo We are in debug
:end
```

4.10.3 Insert unprotected checksum in user ID memory

If it is supported by the device (not grayed out), click the check box to use the checksum number that was generated by the build as the user ID.

For the PIC32 family of devices, the checksum is dependent on the USER ID. Therefore, this feature is not supported under PIC32 devices.

4.10.4 Normalize hex file

A hex file is normalized when line addresses are in incremental order and data is buffered to be one size (16 bytes), where possible.

For example, the following lines (records) are from the file `myfile.hex`, output from the MPLAB XC16 C compiler/ linker:

```
:0801f800361e0000361e000057
:020000040000fa
:10020800361e0000361e0000361e0000361e000096
```

When this file is normalized, the file data looks like this:

```
:10022800361E0000361E0000361E0000361E000076
:10023800361E0000361E0000361E0000361E000066
:10024800361E0000361E0000361E0000361E000056
```

which is of the format:

```
:bbaaaarrdd...ddcc
```

where:

:	Start record
bb	Byte count
aaaa	Address
rr	Record type
dd	Data
cc	Checksum

In MPLAB X IDE, the application Hexmate is used to normalize Intel hex files. When the "Normalize hex file" option is checked, the following is called after a build:

```
hexmate myfile.hex -omyfile.hex
```

Essentially Hexmate unpacks the entire hex file and arranges the data at the addresses specified by the hex file. It then repackages the resulting memory image into a new hex file. In the resulting file, all the data is in ascending order and contiguous. If there is a gap in the addresses, then there will also be a gap in the output file (there is no filling of unused addresses).

For more information on using Hexmate, see the "*MPLAB XC8 C Compiler User's Guide*" (DS50002053) in the `docs` folder of the installed MPLAB XC8 C compiler. Although Hexmate is described in the MPLAB XC8 documentation, it can be used with any compiler.

A normalized hex file is useful for programming code (such as a bootloader) over a serial connection, as the variation of record bytes is minimized.

4.11 Build a Project

For MPLAB X IDE, it is not necessary to build the project first and then run or debug. Building is part of the run and debug processes. For initial development or major changes, however, you may want to make sure that the project builds before attempting to run or debug.

To build a project:

- In the Projects window, right click on the project name and select "Build." You may also select "Clean and Build" to remove intermediary files before building.
- Click on the "Build Project" or "Clean and Build Project" toolbar icon.

Build progress will be visible in the Output window.

Available build functions are:

Table 4-7. Build Options on Toolbar Buttons

Button Icon	Function	Description
	Build Project	Make all the files in the project.
	Build for Debugging	Make all the files in the project and add a debug executive to the built image.
	Build with PRO Comparison	If you use an MPLAB XC C compiler in free mode, you can build in PRO mode as well and see a comparison output of differences, if any exist. See 5.25 Compare MPLAB XC Compiler Free vs. PRO Licenses.
	Clean and Build	Remove previous build files and make all the files in the project.
	Clean and Build for Debugging	Remove previous build files and make all the files in the project. Add a debug executive to the built image.
	Clean and Build with PRO Comparison	Remove previous build files and then Build with PRO Comparison.

To view errors in the Output window:

1. Right click in the Output window and select "Filter."
2. In the Filter dialog, check "Match Case" and enter ": error" to show only errors in the Output window that stopped the build.
3. Toggle the filter on and off using <Ctrl>+<G>.

See you language tool documentation for a discussion of errors.

To view checksum information:

Open the Dashboard window (see 5.19 View the Dashboard Display) to see the checksum after a build.

4.12 Run Code

After the code builds successfully, you can run the application.

How Run Works

When you click the "Run Project" toolbar icon  , the following occurs:

- A build is done, if the make process determines that it is necessary.
- For in-circuit debuggers/emulators and programmers, the target device is automatically programmed with the image (without the debug executive) and the device is released to run, i.e., no breakpoints or other debug features are enabled.

Note: To hold a device in Reset after programming, click the "Hold in Reset" icon  instead of "Run Project".

- For simulators, the application executes, without debugging capability.

Run progress is visible in the Output window.

Running Your Application Code

1. In the Projects window, select your project or make it the main project (right click on the project and select "Set as Main Project").
2. Run your program by either:
   - Clicking on the "Run Program" icon  .
   - Clicking on the "Make and Program Device" icon  .

Run progress is visible in the Output window.

4.12.1 Run Considerations

When running your program, consider that MPLAB X IDE operation connects to the hardware tool at runtime (Run or Debug). Consequently, settings made in MPLAB X IDE are passed to the tool only at runtime. Settings changed during a halt are updated only when runtime is initiated again.

To always be connected to the hardware tool (like MPLAB IDE v8), see *Tools>Options* (*MPLAB X IDE>Preferences* for macOS), **Embedded** button, **Generic Settings** tab, "Maintain active connection to hardware tool" checkbox.

See also 4.20 Program a Device.

4.13 Debug Code

If your code does not run successfully, you should debug it.

How Debug Works

When you click the "Debug Project" toolbar icon  , the following occurs:

- A build is done, if the make process determines that it is necessary.
- For in-circuit debuggers/emulators, the target device or header is automatically programmed with the image (including the debug executive) and a debug session is started.
- For simulators, a debug session is started.

Debug progress is visible in the Output window.

Debugging Your Application Code

To debug your application code:

1. In the Projects window, select your project or make it the main project (right click on the project and select "Set as Main Project").
2. Click on the "Debug Project" icon  or "Step Into" icon to begin debugging.

To halt your application code:

Click on the "Pause" icon  to halt your program execution.

To run your code again:

Click on the "Continue" icon  to start your program execution again.

To end execution of your code:

Click on the "Finish Debugger Session" icon  to end your program execution.

To launch the debugger:

If your code is built for debugging and you simply want to launch the debug tool, you can do so by selecting the down arrow next to the "Debug Project" icon and select "Launch Debugger."

Table 4-8. Debug Icons

Icon	Function	Related Menu Item
	Debug Project	Debug>Debug Project
	Step Into	Debug>Step Into
	Pause	Debug>Pause
	Continue	Debug>Continue
	Finish Debug Session	Debug>Finish Debugger Session

4.13.1 Debug Macros Generated

MPLAB X IDE generates debug macros for use with Microchip language tools. Macros passed to Microchip compilers and assemblers are shown in the table below.

Table 4-9. Microchip Tools Debug Macros

Macro Name	Assoc. Tool	Function
__DEBUG	All	specifies that this is a debug build
__MPLAB_REAL_ICE__ __MPLAB_ICD3__ __MPLAB_PK3__ __MPLAB_PICKIT2__	XC8	specifies the hardware debug tool in use The format is __MPLAB_xxx__, where xxx represents the hardware tool specifier.
__MPLAB_DEBUGGER_REAL_ICE __MPLAB_DEBUGGER_ICD3 __MPLAB_DEBUGGER_PK3 __MPLAB_DEBUGGER_PICKIT2	XC16, XC32, MPASM	specifies the hardware debug tool in use The format is __MPLAB_DEBUGGER_xxx, where xxx represents the hardware tool specifier.
__MPLAB_DEBUGGER_PIC32MXSK	XC32	specifies the starter kit in use

You can use these macros in your own code. For example:

```
#ifdef __DEBUG
fprintf(stderr,"This is a debugging message\n");
#endif
```

4.13.2 Debug Considerations

When debugging your code, consider the following issues:

- You need to be in a debug session (debug mode) to activate many debugging features – for example, to view variable values in the watch or memory windows.
- MPLAB X IDE operation is always connected to the hardware tool. To connect only at runtime, see *Tools>Options* (*MPLAB X IDE>Preferences* for macOS), **Embedded** button, **Generic Settings** tab, and uncheck "Maintain active connection to hardware tool."
  When unchecked, settings made in MPLAB X IDE will be passed to the tool only at runtime. Settings changed during a halt are updated only when runtime is initiated again.
- For some applications you may need to break down the debug steps for independent execution. To do this, use the steps under *Debug>Discrete Debugger Operation*.
- Disable compiler optimizations when debugging or use a 'debug' optimization if available for your compiler. In the Project Properties window, select the language tool executable under "Categories." Then select "Optimizations" from "Options categories" and choose the lowest level, i.e., "0" if available.

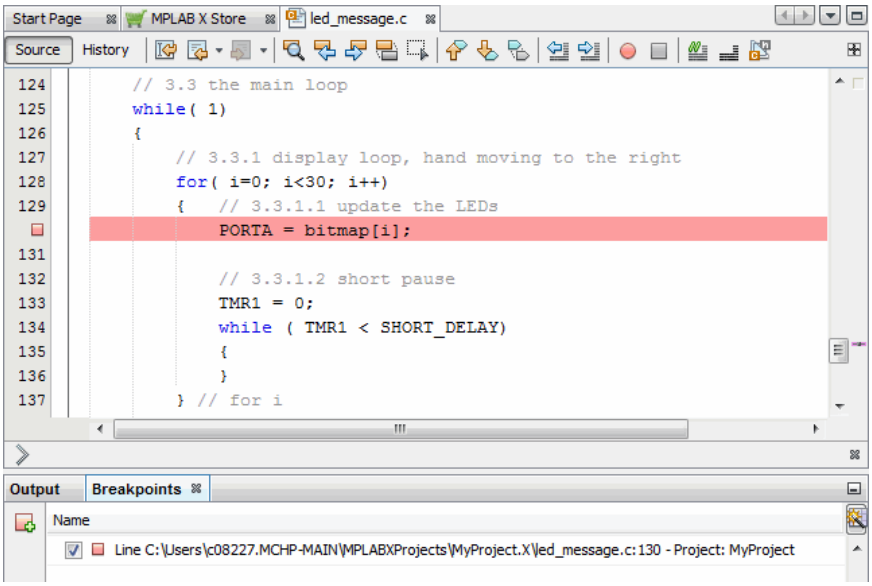
4.14 Control Program Execution with Breakpoints

When debugging code, it can be useful to suspend execution at a specific location in code so that variable values can be examined. To do this, use breakpoints.

For more on breakpoints, see the NetBeans Help topic:

https://netbeans.org/kb/docs/cnd/debugging.html#breakpoints

Figure 4-24. Breakpoint and Breakpoints Window



Related Links

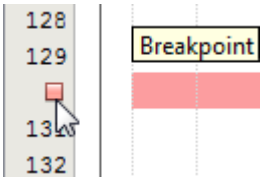
12.2 Breakpoints Window
12.3 New Breakpoint Dialog

4.14.1 Set or Clear a Simple Line Breakpoint

To set or clear a breakpoint on a line, do one of the following:

- Click the left margin of the line in the Source Editor (see figure)
- Click in the line and then press Ctrl+F8
- Select *Debug>Toggle Line Breakpoint*

Figure 4-25. Click to Enable/Disable Breakpoint



4.14.2 Set Breakpoints with the Breakpoint Dialog

To open the New Breakpoint dialog, complete the following steps:

1. Click on the "Create New Breakpoint" icon  in the upper left of the Breakpoints window.
2. Select *Debug>New Breakpoint*. Choose a breakpoint type and set up options.

Table 4-10. Breakpoint Types

Selection	Description
Line	Break on a line of code in an Editor window.
Data	Break on Reads/Writes to an address in data (RAM) memory. This includes SFR and symbol locations.
Address	Break on the execution of an address in program (Flash) memory.
Event	Break on a specific event, such as a reset, sleep/wake, watchdog timer time-out, etc.
Function	Break on entry into a specified function.

For options available for each breakpoint type, see 12.3 New Breakpoint Dialog.

4.14.3 Set or Clear Breakpoints in the Breakpoints Window

To view and toggle the breakpoint in the Breakpoints window, complete the following steps:

1. Toggle the breakpoint by checking/unchecking the checkbox.
2. Select *Window>Debugging>Breakpoints*.

4.14.4 Set a Breakpoint Sequence (Device Dependent)

A breakpoint sequence is a list of breakpoints that execute, but do not halt, until the last breakpoint is executed. Sequenced breakpoints can be useful when there is more than one execution path leading to a certain instruction and you only want to exercise one specific path.

To **create a Breakpoint Sequence**, complete the following steps:

1. Right click on an existing breakpoint or shift click to select a group of existing breakpoints and right click on the group.
2. From the pop-up menu, go to "Complex Breakpoint" and select "Add a New Sequence."
3. Enter a name for your sequence in the dialog box and click OK.
4. The breakpoint(s) will appear under the new sequence.
5. To add additional existing breakpoints to the sequence, right click on the breakpoint and select *Complex Breakpoint>Add to Name*, where *Name* is the name of the sequence.
6. To add new breakpoints to the sequence, right click on the sequence and select "New Breakpoint."

To **select the Sequence Order**, complete the following steps:

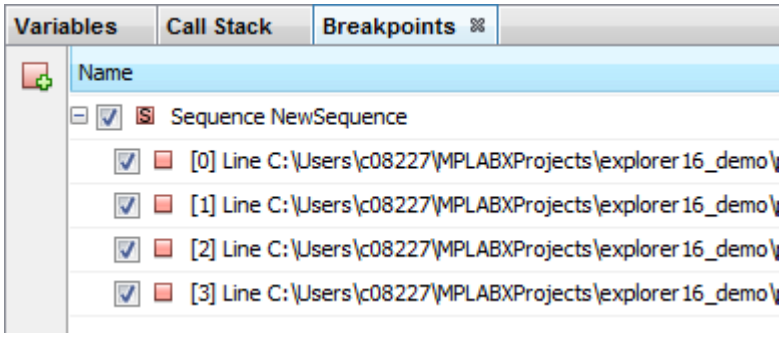
1. Expand on a sequence to see all items.
2. Right click on an item and select *Complex Breakpoints>Move Up or Complex Breakpoints>Move Down*. Sequence execution of breakpoints is bottom-up; the last breakpoint in the sequence occurs first.

To **Remove a Sequence** or Breakpoint, either:

- Right click on the item and select "Disable" to temporarily remove the item.

- Right click on the item and select "Delete" to permanently remove the item.

Figure 4-26. Breakpoint Sequence Example



4.14.5 Set a Breakpoint Tuple (Device Dependent)

For MPLAB X IDE, a tuple represents an ANDed list of breakpoints. ANDed breakpoints can be useful when a variable is modified in more than one location and you need to break only when that variable is modified in one particular location.

Only two breakpoints can be ANDed and these must consist of one program memory breakpoint and one data memory breakpoint. Breakpoint 1 AND breakpoint 2 must occur at the same time before a program halt.

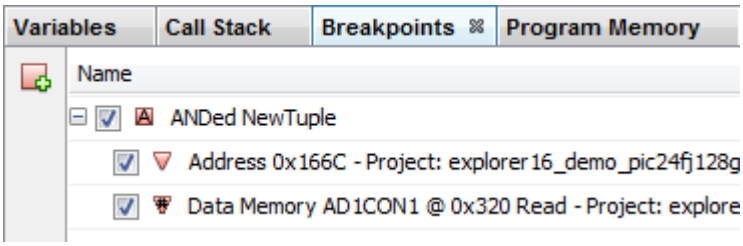
To **Create a Breakpoint Tuple**:

1. Click on the "Create New Breakpoint" icon  in the upper left of the Breakpoints window to open the New Breakpoint dialog.
2. Create an address breakpoint. Click **OK** to add it to the Breakpoints window.
3. Repeat steps 1 and 2 to create a data breakpoint.
4. Right click on one breakpoint and select *Complex Breakpoint>Add to New Tuple*.
5. Enter a name for your tuple in the dialog box and click **OK**.
6. The breakpoint will appear under the new tuple.
7. Right click on the other breakpoint and select *Complex Breakpoint>Move to Name*, where *Name* is the name of the tuple.

To **Remove a Tuple** or Breakpoint, do one of the following:

- Right click on the item and select "Disable" to temporarily remove the item.
- Right click on the item and select "Delete" to permanently remove the item.

Figure 4-27. Breakpoint Tuple Example



4.14.6 See Breakpoint Address(es)

To see the address of a breakpoint before and during a debug session, complete the following steps:

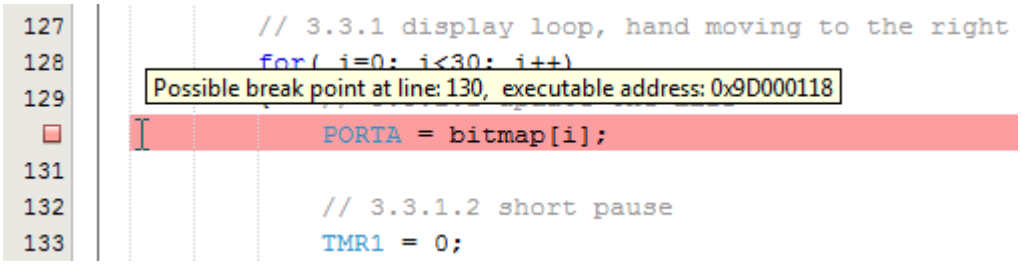
1. Under *Tools>Options*, **Embedded** button, **Generic Settings** tab, select an action for "On mouseover source lines in editor, evaluate break point status."
2. Set a line breakpoint.
3. Click the Debug Project icon to debug your code.

4. Place the cursor between a breakpoint icon and the beginning of source code, and perform the action of step 1 to see the mouseover.

Single Breakpoint Address

In many cases, you will see a single address. Open a program/execution memory window (*Window>Target Memory Views*) to see the memory address and instructions at that location.

Figure 4-28. See Breakpoint Address

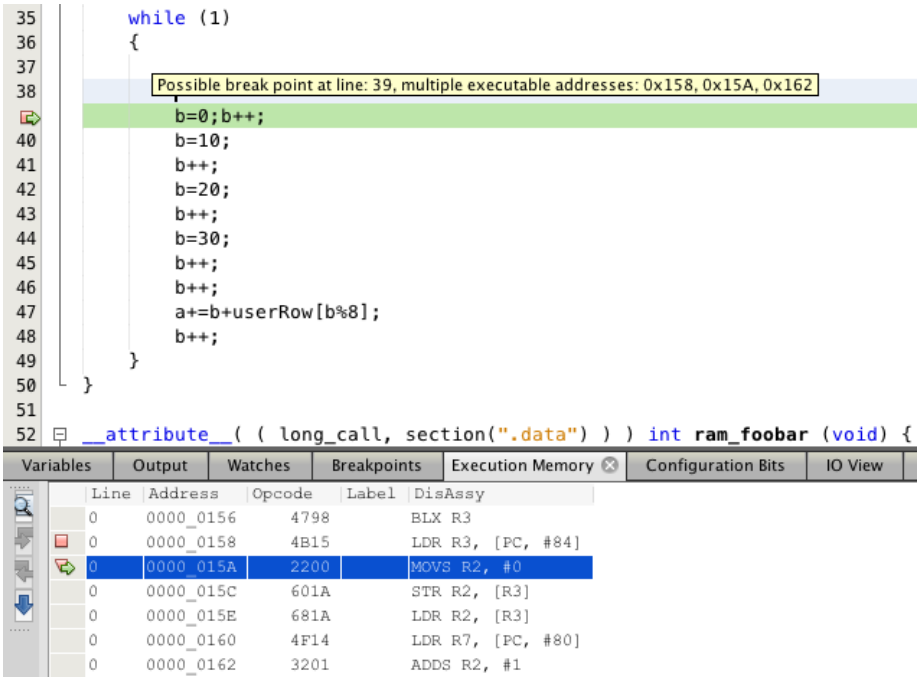


Multiple Breakpoint Addresses

In some cases, you will see multiple possible breakpoint locations. A possible cause for this could be higher compiler optimizations (-Os). Previous to MPLAB X IDE v5.20, only the first address was shown. Now all addresses are.

Open a program/execution memory window (*Window>Target Memory Views*) to see each memory address and instructions at that location. If your breakpoint is not working, you can select other locations in the memory window. When the breakpoint hits, it will still show at the same line number in code.

Figure 4-29. See Multiple Breakpoint Addresses



4.14.7 Hardware Breakpoint Usage

If you must be halted at a line of code to:

- switch from a software breakpoint to a hardware breakpoint
- set a hardware breakpoint

Be aware that debugging will cause a halt on the same line of code. Debug again to continue program execution or perform a single step to execute the code on the breakpoint line.

4.14.8 Dual Partition Devices and Breakpoints

Due to the nature of debugging, dual partition device breakpoints must be set in the active partition once the debugger has halted there. So starting in partition 1, you cannot set breakpoints in partition 2. You need to run up to the switch, switch to partition 2, and then set breakpoints that will reside in partition 2.

4.14.9 Breakpoint Applications

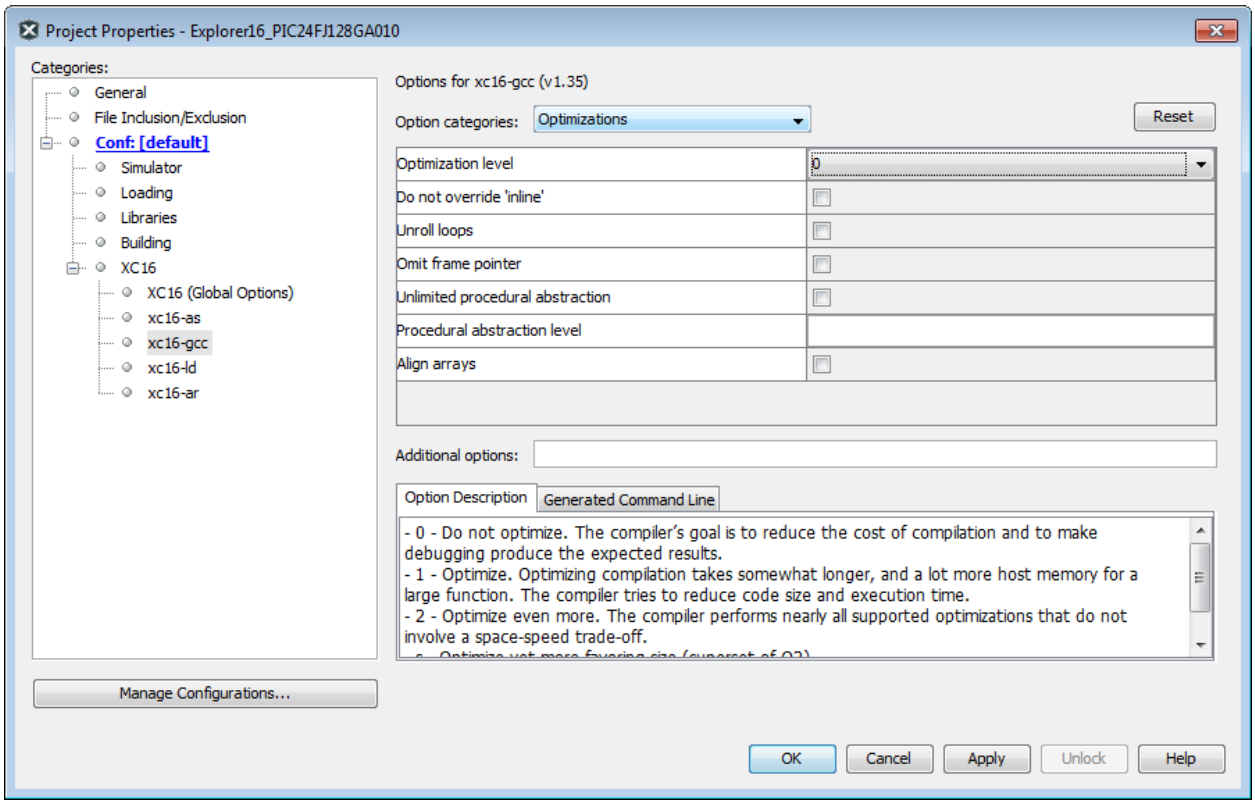
To determine the timing between breakpoints use the stopwatch (see 5.15 Use the Stopwatch).

To determine breakpoint resources open the Dashboard window (see 5.19 View the Dashboard Display) to see the number of available and used breakpoints and whether software breakpoints are supported.

4.14.10 Compiler Optimizations and Breakpoints

Compiler optimizations can change how a program is executed. This can affect memory address mapping of breakpoints, a produce multiple address possibilities. When debugging code, it is recommended that you use the lowest level optimizations to avoid breakpoint issues.

Figure 4-30. Lowest Optimization Level Example



4.14.11 Breakpoint Resource Usage

Hardware debuggers (e.g., in-circuit debug tools) support a limited number of breakpoints. The number of breakpoints available is dependent on the device selected. To see the number of breakpoints available and keep track of the number you have used, see the Dashboard window (5.19 View the Dashboard Display).

The following MPLAB X IDE features use breakpoints to accomplish their functions.

	Step Over
	Step Out

	Run to Cursor
	Reset to Main

If you attempt to use one of these features when no breakpoints are available, a dialog will be displayed telling you that all resources are used.

4.14.12 Breakpoint Awareness

Using tool tips and icons, breakpoint information is presented in and out of a debug session:

- When out of a debug session and no build has occurred - if you try to set a breakpoint, a mouse tool tip displays with a blue torn icon stating no debug information is available.
- When out of a debug session and a build has occurred - if a line has been optimized out, a breakpoint can either be possible or not possible.
- When in a debug session - the tool tip shows whether a breakpoint can be set or not.

The option to show a tool tip can be disabled if you do not want it (*Tools>Options*, **Embedded**, "Generic Settings").

4.14.13 Broken Breakpoint Icon

There are occasions when the breakpoint icon will appear broken ():

- When out of a debug session and no build has occurred, if you try to set a breakpoint a mouse tool tip will occur with a torn icon stating no debug information is available.
- When in a debug session, source files may not be able to be found due to path, file or folder naming restrictions (see Path, File and Folder Name Restrictions).

4.15 Step Through Code

Use one of the stepping functions on the Debug menu and Debug toolbar to move through code either from the beginning of code or after a breakpoint halt. Examine changes in variable values (see next section) or determine if the program flow is correct.

There are several ways to step through code:

	**Step Over** – Executes one source line of a program. If the line is a function call, it executes the entire function and stops.
	**Step Into** – Executes one source line of a program. If the line is a function call, it executes the program up to the function's first statement and stops.
	**Step Out** – Executes one source line of a program. If the line is a function call, it executes the functions and returns control to the caller.
	**Run to Cursor** – Runs the current project to the cursor's location in the file and stops program execution.

In addition to the Editor window, you can single-step through code in the Disassembly window (5.16 View the Disassembly Window) and program memory in a Memory window.

4.16 Watch Symbol Values Change

Watch the values of symbols that you have selected change in the Watches window. Knowing whether these values are set as expected during program execution will help you to debug your code.

The following symbols may be added to a Watches window:

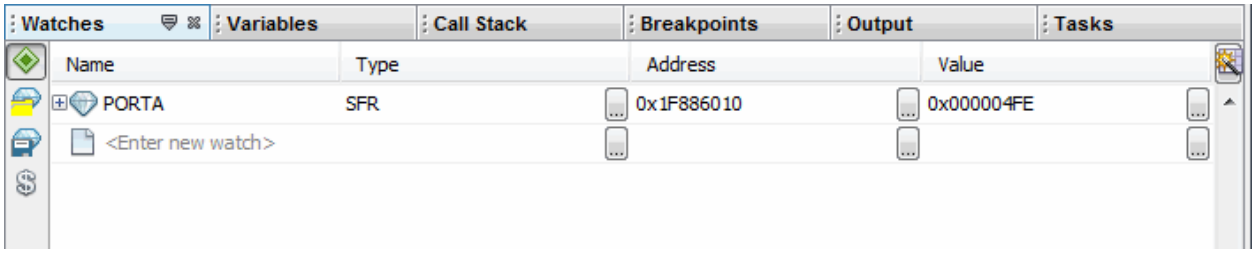
- Global symbols – visible after a build
- SFRs – special function registers (device dependent)
- Absolute addresses

In general, you must Pause from debugging to be able to see updated values. However, some tools allow runtime updates (you can see the values change as your program is running). Check your tool documentation to see if it supports this feature.

For all devices except PIC32 MCUs, symbols used in a runtime watch must be sized to match the device memory. That is, you need 8-bit symbols when using an 8-bit device.

For C-language enumerated types, you may enter either the enum label (text) or integer value in the window. For labels, be aware that they are case sensitive.

Figure 4-31. Watches Window – Program Pause



To view the Watches window do one of the following:

- Select *Window>Debugging>Watches* to open the window.
- If the window is already open, click the Watches tab in the Output window.

To create a new watch directly:

You can add a symbol directly to the Watches window by doing one of the following:

- Double click in the name column and type in a global symbol, SFR, or absolute address (0x300).
- Right click on a global symbol or SFR in the Editor window and select "New Watch."
- Select the global symbol or SFR in the Editor window and drag-and-drop it into the Watches window.

To create a new watch using the New Watch dialog:

You can add a symbol or SFR to the Watches window by doing one of the following:

- Right click in the Watches window and select "New Watch" or select *Debug>New Watch*. Click the selection buttons to see either Global Symbols or SFRs. Click on a name from the list and then click **OK**.
- Select the symbol or SFR name in the Editor window and then select "New Watch" from the right click menu. The name will be populated in the window. Click **OK**.

To create a new runtime watch:

Before you add a runtime watch to the Watches window, you need to set up the clock:

1. Right click on the project name and select "Properties."
2. Click on the debug tool name (e.g., PICkit 4), and select the "Clock" option category.
3. Set the runtime instruction speed.

To add a global symbol or SFR as a runtime watch, follow the instructions under To create a new watch using the New Watch dialog:, except select "New Runtime Watch" instead of "New Watch."

For all devices except PIC32 MCUs, symbols used in a runtime watch must be sized to match the device memory. That is, you need 8-bit symbols when using an 8-bit device.

To view symbol changes:

1. Debug and then Pause your program.
2. Click the Watches tab to make the window active.
3. For watch symbols, continue the debug session and Pause to see changing values. For runtime watch symbols, continue Debug and watch the values change as the program executes.

You must be in a debug session to see the values of symbols – global symbols, SFRs, arrays, register bitfields, etc.

Note: You cannot view the value of macros in this way. Use the Macro Expansion Window by right clicking on the macro in the editor during debug.

To change the radix of a watch symbol:

Right click in the line of the symbol and select "Display Value As."

To perform other tasks:

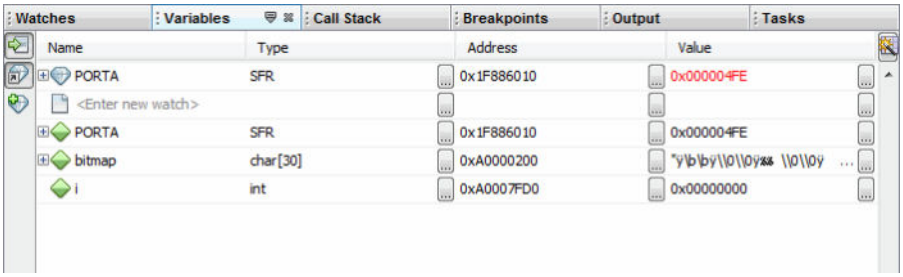
For more on watches, see 12.19 Watches Window.

4.17 Watch Local Variable Values Change

Watch the values of local variables change in the Variables window. Determining if these values are as expected during program execution will help you to debug your code.

In general, you must Pause from debugging to be able to see updated values. However, some tools allow runtime updates. Check your tool documentation to see if it supports this feature.

Figure 4-32. Variables Window – Program Pause



To view the Variables window do one of the following:

- Select *Window>Debugging>Variables* to open the window.
- Click the Variables tab in the Output window if the window is already open.

To view variable changes:

1. Debug and then Pause your program.
2. Click the "Variables" tab to view the window and see the local variable value.

To change the radix of a variable:

Right click in the line of the variable and select "Display Value As."

4.18 View or Change Device Memory

MPLAB X IDE has flexible, abstracted memory windows that provide a more customized view of the different types of device memory during debug. You must Pause from debugging to be able to see updated values in this window.

View Device Memory

1. Click a window in a pane to make the pane active. The memory window will open in this pane.
2. Select a memory view from *Window>Target Memory Views*. The available choices are shown in the following tables. Not all memory types are available for all devices.

Table 4-11. Memory Views – 8- and 16-Bit Devices

Type	Description
Program Memory	all program memory (ROM) on the device
File Registers	all file register (RAM) memory on the device

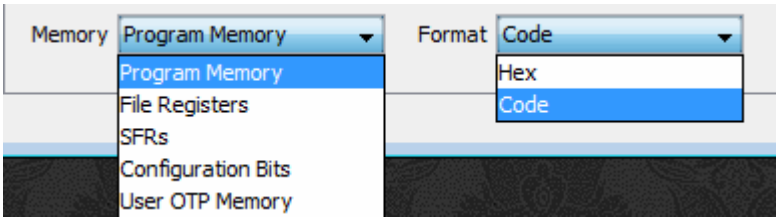
.....continued	
Type	**Description**
SFRs	all Special Function Registers (SFRs)
Peripherals	all SFRs by Peripheral
Configuration Bits	all Configuration registers
EE Data Memory	all EE Data memory on the device
User OTP Memory	User OTP memory
User ID Memory	User ID memory

Table 4-12. Memory Views – 32-Bit Devices

Type	Description
Execution Memory	all Flash memory on the device
Data Memory	all RAM memory on the device
Peripherals	all Special Function Registers (SFRs)
Configuration Bits	all Configuration registers
CPU Memory	all CPU memory
User ID Memory	User ID memory

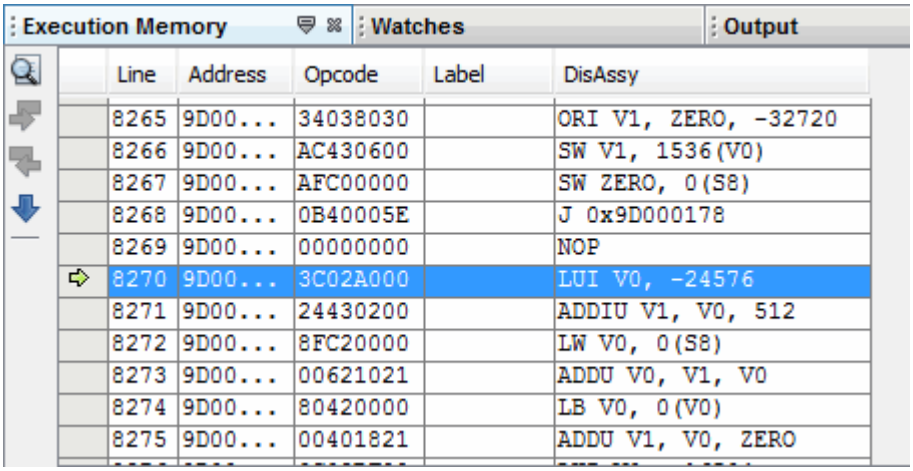
3. When a Memory window is open, you may change your view by selecting a different memory in the Memory drop-down box. Or, you can change the format of the current memory display in the Format drop-down box. The format choices depend on the type of memory chosen and the device.

Figure 4-33. Memory and Memory Format Selection



4. After Debug and then Pause, the window will populate with the memory chosen.
5. Close the window by clicking the "x" on that window's tab.

Figure 4-34. Memory Window Content



Change Device Memory

You must Debug your code to change memory values. You cannot change memory when code is running.

Note: The data will change only during Debug. Your application code is not changed.

Use the following information to change memory values:

- Change a value in the Memory window by clicking in the appropriate column and selecting or entering new data. For some windows, the text will be red to show a change.
- The Fill memory feature is found on the context (right click) menu of most Memory windows.
- For program memory, you must rebuild to see the changes. Use *Debug>Discrete Debugger Operation* to program the target and launch the debugger with the changed data.

Set Memory Window Options with the Context Menu

Right clicking in the Memory window will pop up a context menu with various options such as display options, fill memory, table import/export and output to file. The content of the menu depends on the window. See 12. MPLAB X IDE Windows and Dialogs.

Refresh Selected Memory Windows

For Memory windows showing program, EEPROM, user ID, or configuration bits memory, you can refresh the view by doing the following:

1. If you are debugging, halt your program unless your tool and device support Debug Reads (see below).
2. Click on the icon named "Read Device Memory"  .

Debug Reads

For most devices, you must halt your program (Finish Debugger Session) before you can read device memory. For some devices, you can read while in debug mode (Debug Read). You will know that this is available as the "Read Device Memory" icon will not be grayed out when you are debugging.

Currently, only MPLAB REAL ICE in-circuit emulator and MPLAB ICD 3 support Debug Reads.

Debug Reads are done at target oscillator speeds so if the target is running very slow, a read may take a long time. You can force a fast ICSP read by finishing the debug session and then doing a read since ICSP reads will always be done when not in a debug session.

4.19 Set Configuration Values in the Configuration Bits Window

Select "Configuration Bits" from *Window>Target Memory Views* to open the Configuration Bits window. For details on using memory windows, see 4.18 View or Change Device Memory.

The Configuration Bits window can aid you in developing your Configuration bit settings. However, you must set Configuration bits in code for production.

4.19.1 Working with the Configuration Bits Window

You can temporarily change Configuration bits during a debug session in the Configuration Bits window (see 4.18 View or Change Device Memory.) Once you have the settings you want, you have several options to preserve these settings as discussed in the following sections.

You cannot edit the Configuration bits if you are not in a debug session. Also, if you rebuild, all changes in the Configuration Bits window will be lost.

For a summary of Configuration bits settings for different devices, see 14. Configuration Settings Summary.

Note: For AVR or SAM devices, red text in this window indicates that an action is needed. Hover your mouse over to see pop up text for instructions.

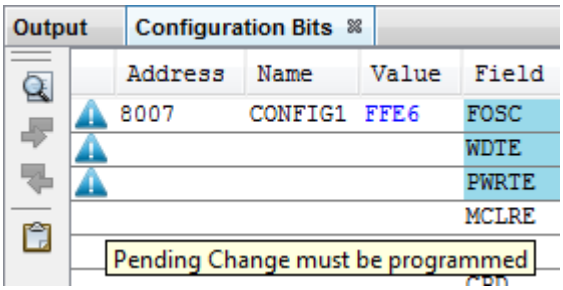
Edit Configuration Settings

Editing values in this window is always active, unlike other memory windows where you must be in debug mode. However, if you rebuild, all changes in the Configuration Bits window will be lost.

Default values of the editable fields will occur on initial entry to Configuration Bits view and also on Reset to Defaults menu action. When configuration bits are expressed in source code, the value of all editable fields will be initialized to the value assigned in source code. This occurs following the build action and after a successful load.

When you change a default value in the configuration window, you are reminded that this is a temporary change that will not be permanent until programmed into the target device.

Figure 4-35. Pending Change



You can change values as follows:

- **Selection Box Values**: Selection box values are available in the "Option" or "Setting" column. Simply click on the column value for the field that you want and select from the list to change.
- **Manual Entry Values:** Manual entry values may be available in the "Option" or "Setting" column. The "Option" text will state that there is a user selectable range, e.g., "User range: 0x0 - 0x3F." The "Setting" text will state that a value needs to be entered, e.g., "Enter Hexadecimal value."

  Once you click on either the "Option" or "Setting" text for the field you want, the corresponding "Value" column will become editable. The value entered will be validated against absolute (raw) values, so there is no need to be concerned about the actual position with regard to where an edited field value will be applied. Attempts to '0' pad entered values to approximate actual mask position will be filtered and corrected to an absolute value to the extent possible.

Generate Source Code to Output

Click on the button Generate Source Code to Output on the bottom of the Configuration Bits window (first figure below). This will generate Configuration Bit Settings in the Output window that you can then copy into your code (second figure below).

Figure 4-36. Configuration Bits Window - Generate Source Code in Output

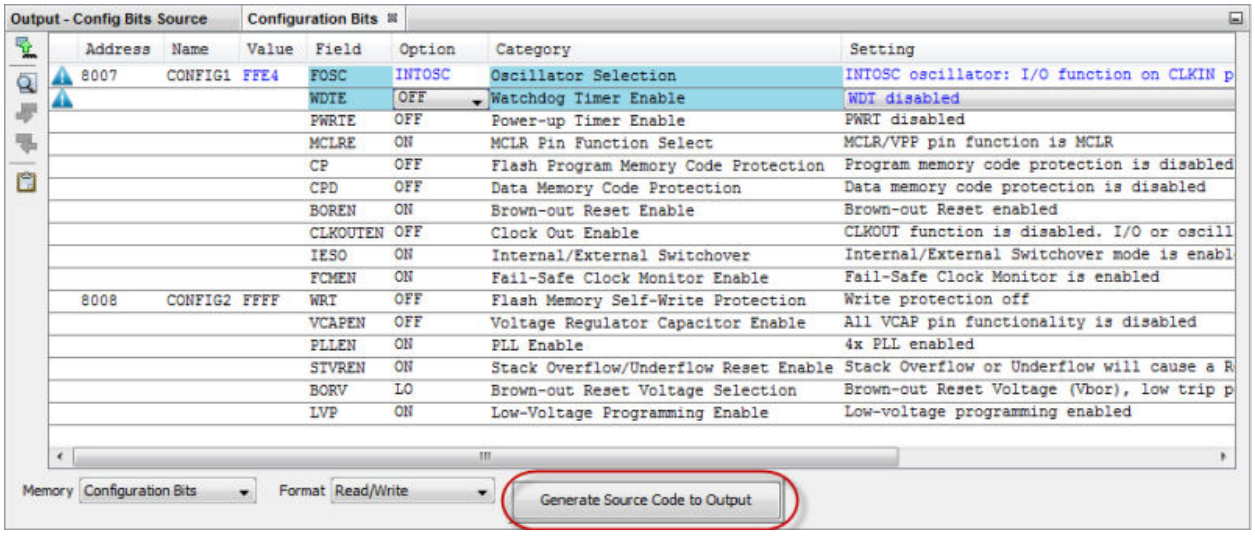
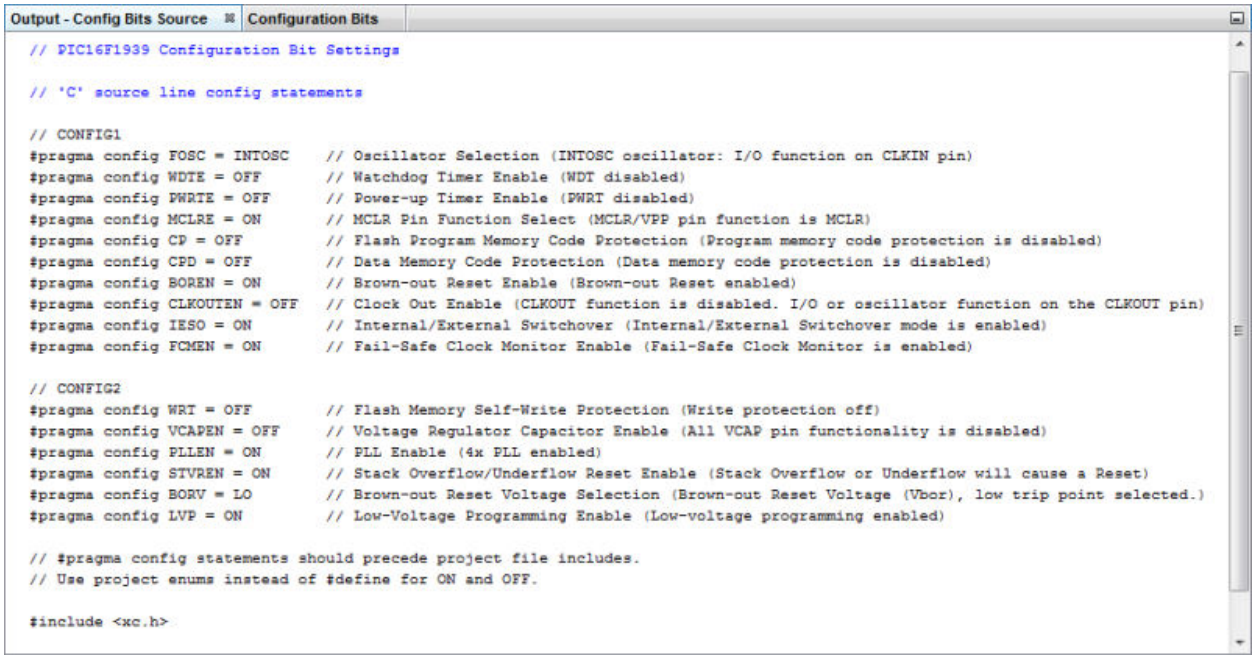


Figure 4-37. Output Window - Generated Source Code



Insert Source Code in Editor

You can select to "Insert Source Code In Editor" from the Configuration Bits window context menu or side bar "clipboard" icon (first figure below). Open or create a main project file in the editor, place the cursor where you want the configuration bits settings to go, then click on the icon (second figure below).

Figure 4-38. Configuration Bits Window - Insert Source Code in Editor

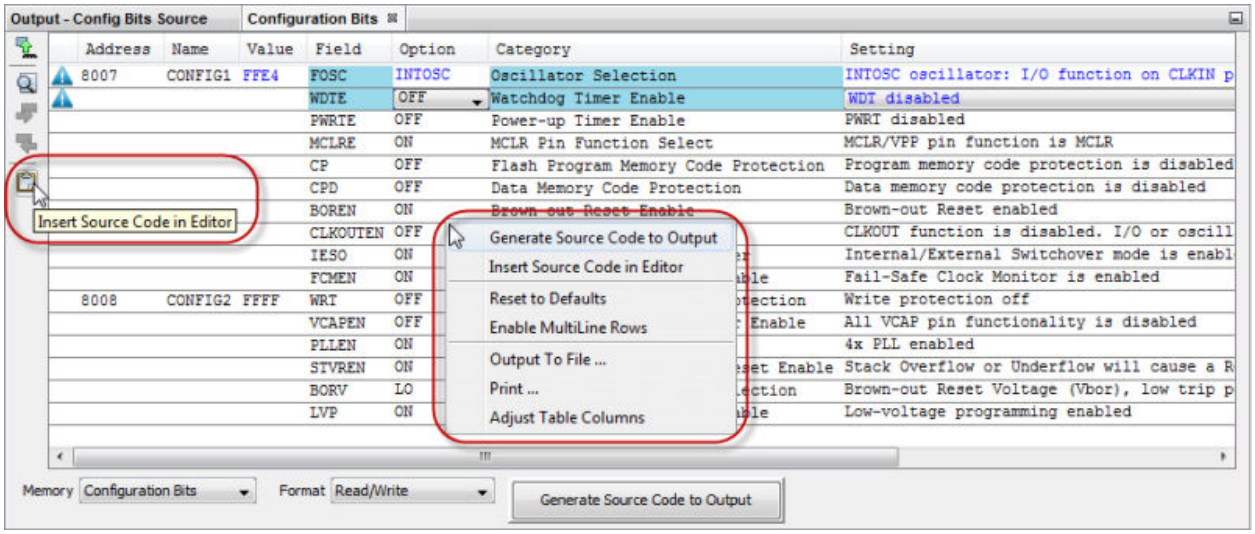
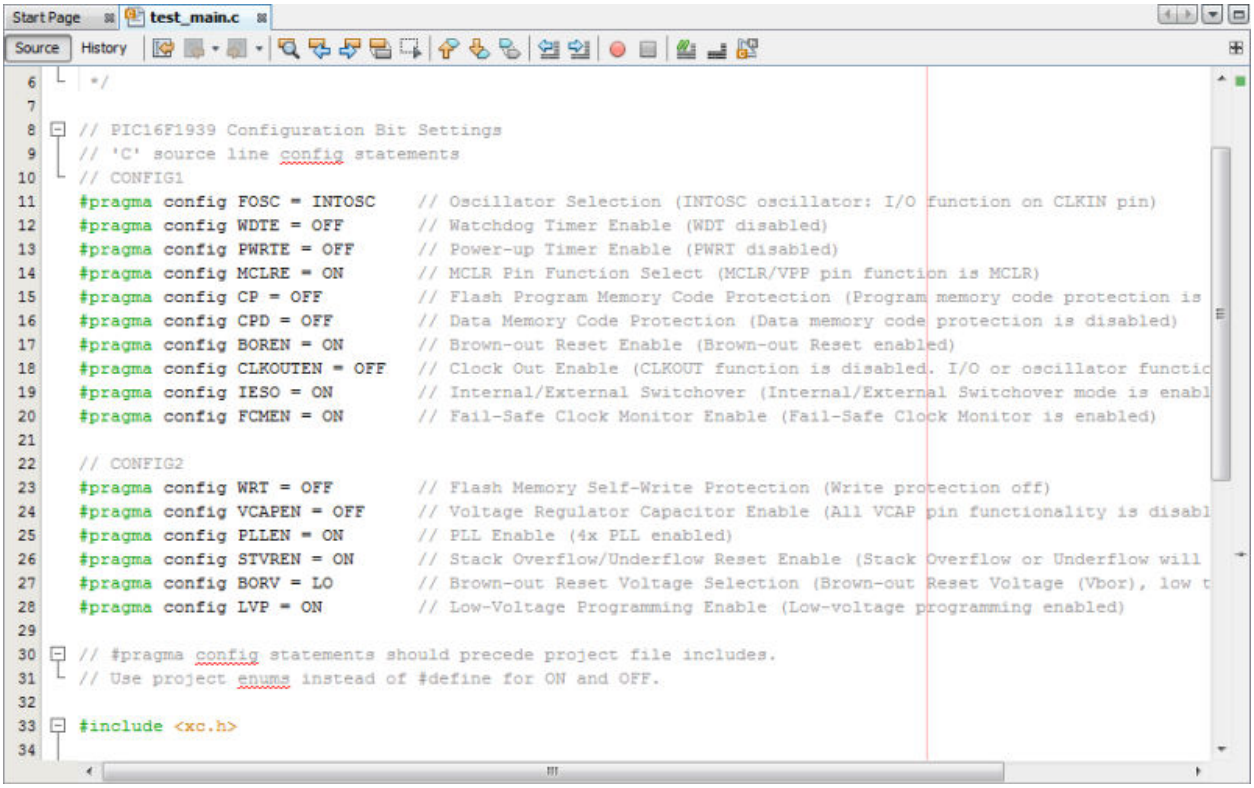


Figure 4-39. Editor Window - Generated Source Code



Program Device for Debugging

Click on the IDE toolbar icon "Program device for debugging" to program the current values of the Configuration Bits window into a device. See 4.20 Program a Device.

4.19.2 Setting Configuration Bits in Assembly Projects

For MPLAB XC assembly projects, you can still use the Configuration Bits window. (This does not apply to MPASM or MPLAB ASM30 projects.) Add a simple C file to your project to contain the generated `#pragma config` macros. Then you will not have to hand-enter assembly `config` words.

There are no differences between the assembly `config` words and `#pragma config`; they both write a single program word and result in the same program size. However, the `#pragma config` syntax is more flexible and allows for improved validation, formatting, and descriptive comments.

4.20 Program a Device

Once your code is debugged, you can program it onto a target device.

Set Project Programming Properties

Set up programming options in the Project Properties window:

1. Right click on the project name in the Projects window and select "Properties."
2. Under "Categories," click on the hardware tool you will use to program your code, e.g., PM3.
3. Review the settings under the "Memories to Program" options category.
   If you wish to use a Preserve Memory option, ensure that your code is not code protected. Code is preserved when the programmer reads the section it needs to save, performs a bulk erase of the device, reprograms the device and then rewrites the area that is preserved with what was saved.
4. Review the settings under the "Program Options" category.
5. Depending on your hardware tool, there may be other programming option categories. Review each one to ensure the settings are correct for your project.

6. When using RTDM with DMCI, Motor Control applications etc., you will need to check the checkbox "Load symbols when programming or building for production (slows process)" under the "Loading" category.

For more information about programming options, please consult your hardware tool documentation.

After the programming options are set up as you desire, you may proceed to program the device.

Perform Programming

To program your target device with debugged code, click the toolbar button Make and Program Device Project.

Other programming-related functions are shown in the Table 4-9. The first function is activated by clicking on the button. For other functions, click on the down arrow next to the button icon.

Table 4-13. Programming Functions on Toolbar Buttons

Button Icon	Function	Details
	Make and Program Device	The project is built (if necessary) and the device is programmed. The program will immediately begin execution on completion of programming.
	Program Device for Debugging	The device is programmed from a debug image. The program will immediately begin execution on completion of programming.
	Program Device for Production	The device is programmed from a production image. The program will immediately begin execution on completion of programming.
	Programmer to Go PICkit 3	Use the Programmer to Go feature of PICkit 3.
	Read Device Memory	Transfer what is in target memory to MPLAB X IDE.
	Read Device Memory to File	Transfer what is in target memory to the specified file.
	Read EE/Flash Data Memory to a File	Transfer what is in target data memory to the specified file.
	Hold In Reset	Toggle the device between Reset and Run.

Program Using the MPLAB Integrated Production Environment

Not all programming functions are in the MPLAB X IDE. For additional programming support, see the MPLAB IPE included with the MPLAB X IDE installation.

	Desktop Icon for MPLAB IPE

5. Additional Tasks

The following steps show how to perform more tasks in MPLAB X IDE.

Table 5-1. Performing Additional Tasks

1 Work with Projects	1. Learn how to Work with Device Packs in your project. 2. Choose how to Open an Existing MPLAB X IDE Project. 3. Import an Existing MPLAB IDE v8 Project into MPLAB X IDE. 4. Open a prebuilt image (Hex, ELF or COF) using the Prebuilt Projects import wizard. 5. Use Loadable Projects, Files and Symbols to combine or replace project hex files. A common application is using loadables for combining bootloaders and application code, as in Loadable Projects and Files: Bootloaders. 6. Create Library Projects to build their output as a library. 7. Open an existing Studio or Start project by using the Import Atmel Studio 7 or Atmel START Project wizard. 8. Create projects from Other Embedded Projects or Sample Projects. 9. Work with Other Types of Files, not just Microchip compiler-supported file types. Also Modify or Create Code Templates to change the default file templates that you use in your project. 10. Switch Hardware or Language Tool used in your project. 11. Modify Project Folders and Encoding of an existing project.
2 Debug Code	1. Use the Stopwatch to determine the timing between breakpoints. 2. View the Disassembly Window to see disassembled code. 3. To navigate function calls, View the Call Stack or View the Call Graph. 4. View the Dashboard Display to see project information such as breakpoint resources, checksums and memory usage. 5. View Registers for the Project (I/O View) during design and debug.
3 Manage Code	1. Improve your Code* by using refactoring and profiling tools. 2. Control Source Code using Local History by using built-in file history. Or Control Source Code using a Revision Control System. 3. Collaborate on Code Development and Error Tracking* by using a team server and an issue tracking system. 4. Compare MPLAB XC Compiler Free vs. PRO Licenses to determine what optimizations work best for your application.
4 Add Functionality	1. Add Plugin Tools to aid code development.
* To see this feature, refer to the **Start Page**, **My MPLAB X IDE** tab, "Extend MPLAB" section, "Selecting Simple or Full-Featured Menus" topic.	

5.1 Work with Device Packs

MPLAB X IDE requires specific information about supported devices in order to simulate or emulate these devices. This information is found in device (`.PIC`) files and includes data on device power requirements, programming methods, architecture, etc.

As of MPLAB X IDE v5.00, the device files are grouped into versioned device family packs (DFPs). Although each MPLAB X IDE version comes with device packs, you may upgrade the device pack version to include new devices, new device feature support, or device bug fix support.

For example, to find the device family pack (DFP) containing the PIC16F19197 device file on a Windows OS computer:

```
C:\Program Files (x86)\Microchip\MPLABX\v5.00\packs\Microchip\PIC16F1xxxx_DFP
\1.0.38\edc\PIC16F19197.PIC
```

where:

- Installation Location: `C:\Program Files (x86)\Microchip\MPLABX`
- Version: `\v5.00`
- Folder Containing Packs: `\packs`
- Device Family Pack: `\Microchip\PIC16F1xxxx_DFP`
- Pack version: `\1.0.38`
- Folder containing Device Files: `\edc`
- Device File: `\PIC16F19197.PIC`

For more information, refer to Developer Help:

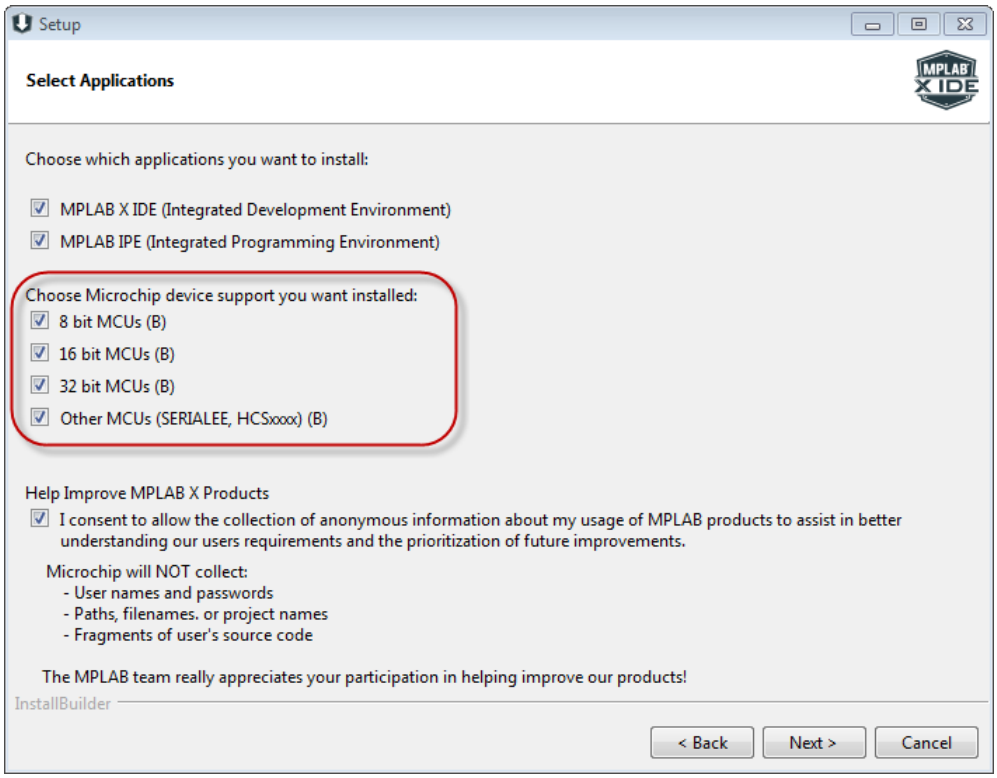
Device Packs in Projects

Device Pack Frequently Asked Questions

5.1.1 Install Packs during MPLAB X IDE Install

When installing MPLAB X IDE (as of MPLAB X IDE v5.20), you may now "Choose Microchip device support you want installed." This will install the packs related to the device architecture(s) chosen. Installing only what you need saves disk space and speeds installation.

This is also called a distributed installer.



Information on installing MPLAB X IDE may be found in the MPLAB X IDE Readme.

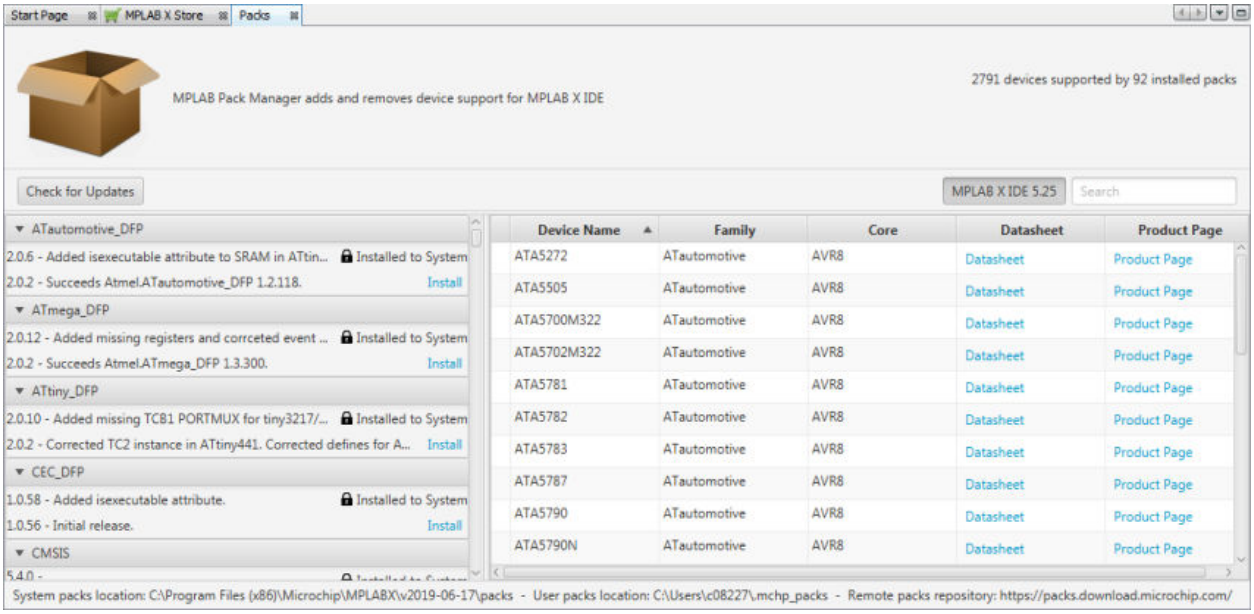
5.1.2 Install Packs after MPLAB X IDE Install

Select *Tools>Packs* to open a list of versioned packs that you can install via the MPLAB Pack Manager.

When first opened, the list is filtered for packs known to be compatible with the current IDE release. Click on the IDE release button to disable this feature and see all pack versions. See below for an example.

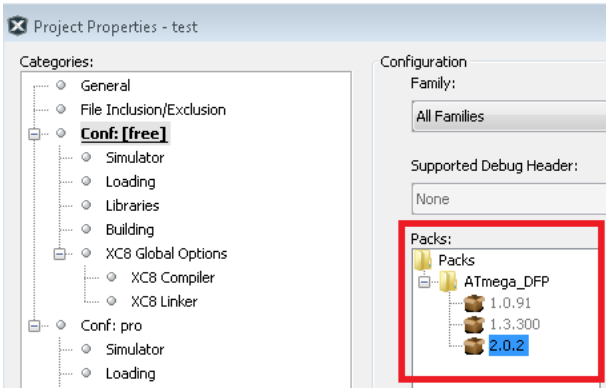
MPLAB X IDE 5.25	Pack list filtered
MPLAB X IDE 5.25	All Pack versions shown

Figure 5-1. MPLAB Pack Manager



5.1.3 Switch Installed Packs

For a project open in MPLAB X IDE, right click on the project name in the Projects window and select "Properties" to open the Project Properties window. View the Packs list to see available packs for the project device.



Packs shown as gray are not included in that version of MPLAB X IDE. To install the pack, see 5.1.2 Install Packs after MPLAB X IDE Install. Turn off the filter to see all versions and then install the version(s) you want.

Note: If you switch devices in the Project Properties to a device that does not have a device pack installed, you **will not** be prompted. You must install manually via the MPLAB Pack Manager.

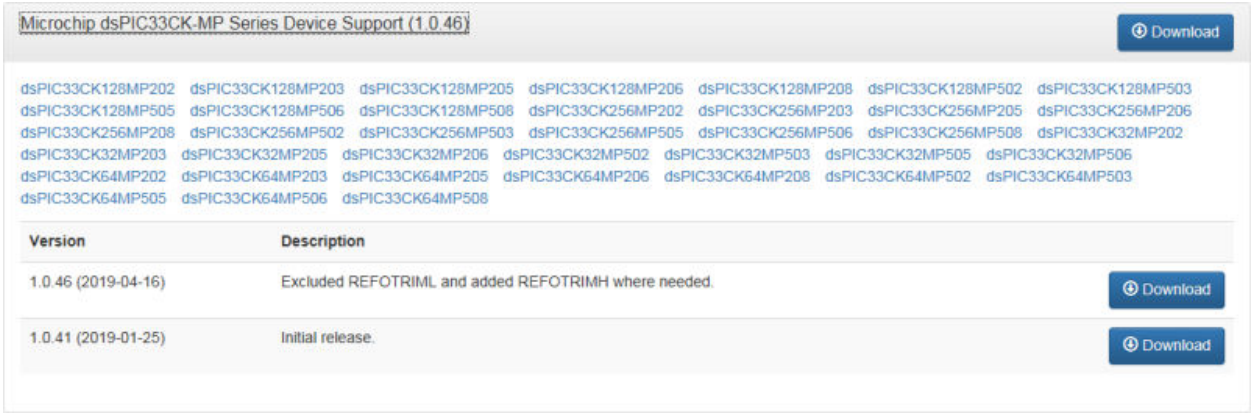
5.1.4 Packs on the Web

To find and download pack versions from the web, go to:

https://packs.download.microchip.com/

Click on the name of a pack to see pack information, such as supported devices and pack version history.

Figure 5-2. Microchip Packs Repository



5.2 Open an Existing MPLAB X IDE Project

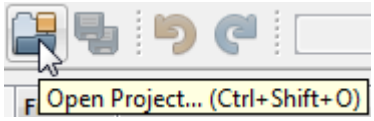
Once you have an existing project, either one that you created or a downloaded example, there are a several ways to open this project.

Open from IDE Menus

Select *File>Open Project* or *File>Open Recent Project*.

Open by Icon

Click on the **Open Project** icon.



Open by Drag-and-Drop of Project Folder

Select a project folder in your file manager window. Drag and drop the folder into the **Editor Pane** (see 4.2 View Changes to Desktop). The project will open in the Projects window.

Figure 5-3. Drag and Drop Folder

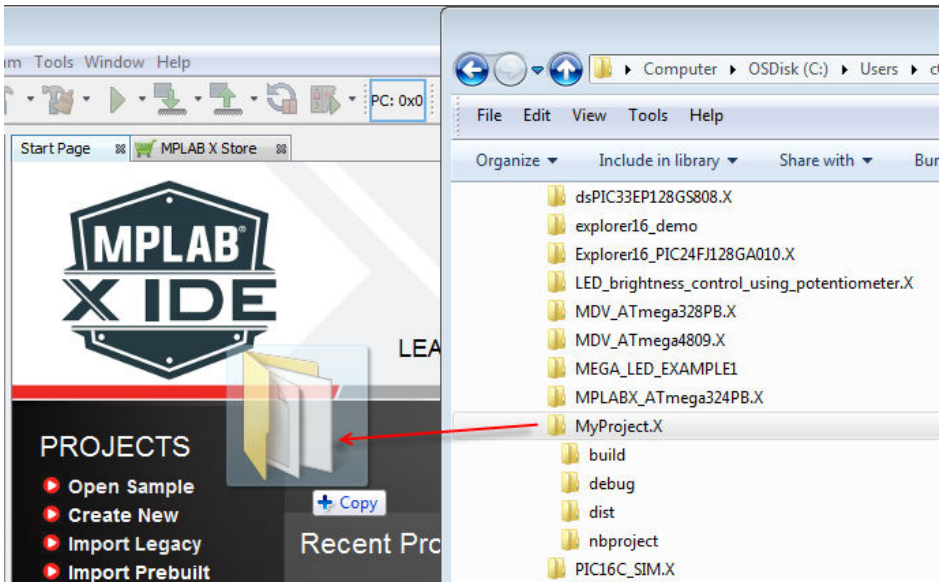
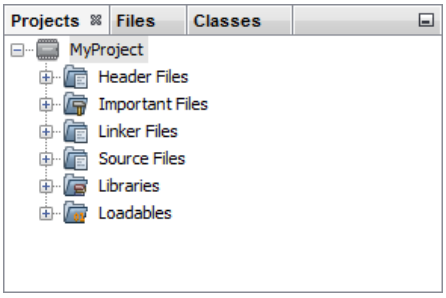


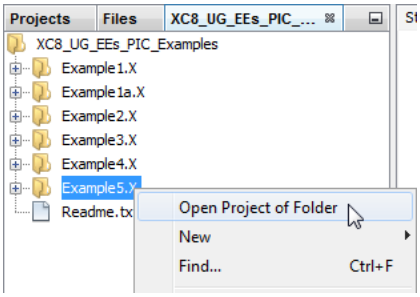
Figure 5-4. Opened Project



Open by Drag-and-Drop of Folder containing Projects

To open a folder containing projects, select the folder in your file manager window. Drag and drop the folder into the **Editor Pane** (see previous step). A window with the name of the folder on the tab will open in the **File Pane**. The folder and related projects will be shown in this window. To open a project in this folder, right click on the project name and select "Open Project of Folder." The project will open in the Projects window.

Figure 5-5. Opened Folder containing Projects



5.3 Import an Existing MPLAB IDE v8 Project

Use the New Project wizard to import an MPLAB IDE v8 project into an MPLAB X IDE project with the following considerations:

MPLAB IDE v8 workspace settings not transferred

Settings that were saved in a workspace in MPLAB IDE v8 (such as tool settings) will not be transferred to the new MPLAB X IDE project. Refer to the MPLAB IDE v8 help for what is stored in a workspace (*MPLAB IDE Reference>Operational Reference>Saved Information*).

Project settings, such as compiler, linker, and assembler options, will be transferred to the new MPLAB X IDE project.

MPLAB IDE v8 project possible file changes

An MPLAB IDE v8 project using the "MPLAB C Compiler for PIC24 MCUs and dsPIC DSCs" (aka MPLAB C30) and a COFF debug file format may have changes.

- The MPLAB X IDE project will be converted to the ELF/DWARF debug file format unless the project uses COFF libraries, in which case the project format will continue to be COFF.
- MPLAB IDE v8 is case insensitive to file extensions such as `.c` and `.C`. However, MPLAB X IDE is case sensitive and associates `.c` to C code files and `.C` to C++ code files. Therefore, if you import an MPLAB IDE v8 project with C code specified as `.C`, MPLAB X IDE will rename the `.C` files to `.c` to avoid incorrect compiler behavior.

MPLAB IDE v8 project possible build changes

An MPLAB IDE v8 project imported into MPLAB X IDE may not build the same as the original project.

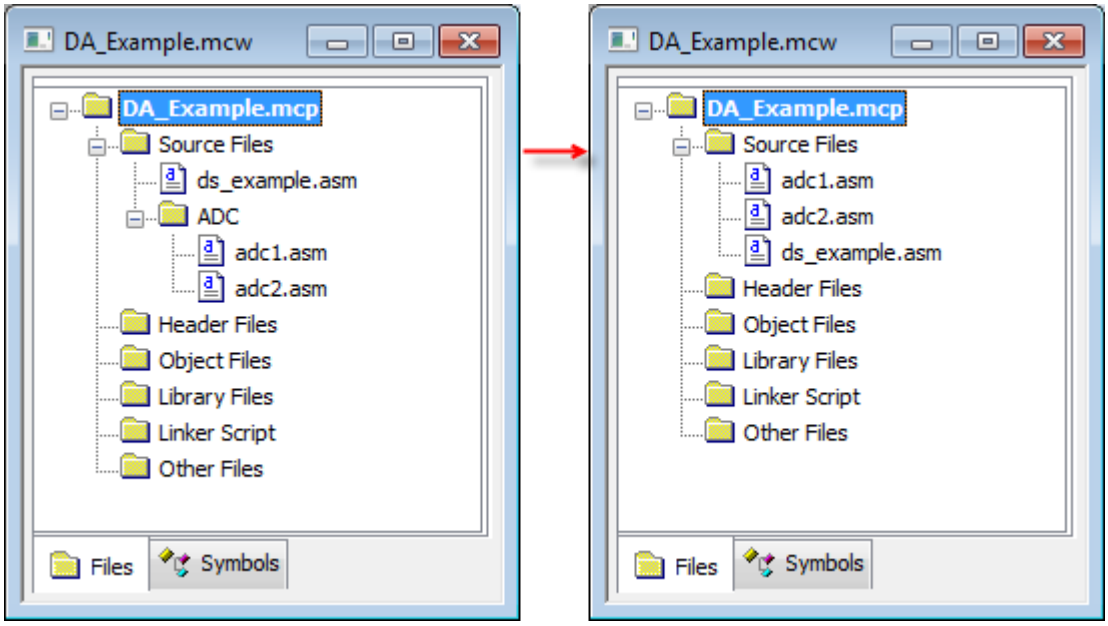
- Do not use `__FILE__`, `assert()`, `__TIME__`, or `__DATE__` in your project code or the build (and checksum) will be different.

- An MPLAB IDE v8 project that uses logical subfolders will not have the same checksum when imported into MPLAB X IDE. To build with the same checksum, move the files in the logical subfolders into the main logical folder (i.e., Source Files, Header Files, etc.) and then import into MPLAB X IDE.

To remove a logical subfolder under Source Files:

- Drag to files under Source Files.
- Select the folder and hit <Delete>.
- Alternately, you can right click on the folder and select "Remove from Project" to remove the folder from the project. But it exists on the PC, it will not be deleted.

Figure 5-6. Move Files and Delete Subfolder



5.3.1 Open the Import MPLAB IDE v8 Project Wizard

There are two ways to open the wizard, described below.

Import Legacy Project Option

To open the Import Legacy Project wizard, do one of the following:

- On the **Start Page**, click on the **Learn & Discover** or **My MPLAB X IDE** tab, "Projects" section, "Import Legacy" link.
- Select *File>Import>MPLAB IDE v8 Project*.

New Project Option

To open the New Project wizard, do one of the following:

- On the **Start Page**, click on the **Learn & Discover** or **My MPLAB X IDE** tab, "Projects" section, "Create New" link.
- Select *File>New Project* (or Ctrl+Shift+N).
- Click on the "New Project" icon on the toolbar. 

A wizard will launch to guide you through new project set up.

Step 1. Choose Project: select the "Microchip Embedded" category and choose from the project type "Existing MPLAB IDE v8 Project." Click **Next**.

5.3.2 Use the Import MPLAB IDE v8 Project Wizard

Follow the steps below to import your MPLAB IDE v8 project. The steps are numbered according to the "Import Legacy Project" Option (see 5.3.1 Open the Import MPLAB IDE v8 Project Wizard.)

Click **Next** to move to the next step.

Step 1. Locate MPLAB IDE v8 Project *.mcp File. Enter or browse to the legacy project.

Step 2. Select Device. Select the device you will be using in your application from the "Device" drop-down list. To narrow your selection list, choose a Family first.

Step 3. Select Header. This step will appear if a header is available for your selected device. To determine if a header is required for debug or if your device has on-board debug circuitry, consult one of the documents below. Then choose whether or not to use a header.

- Processor Extension Pak and Debug Header Specification
- Emulation Extension Pak and Emulation Header User's Guide

Step 4. Select Tool. Select the development tool you will be using to develop your application from the list. To determine support for your device, see 4.1.6.1 Project Tool Support.

Step 5. Select Plugin Board. Select a plugin board, if using one.

Step 6. Select Compiler. Select the language tool (compiler) you will be using to develop your application from the list. To determine support for your device, see 4.1.6.1 Project Tool Support.

Step 7. Select Project Name and Folder. It is recommended that you do not change the default name and location to preserve maintainability of both projects. See example in the figure below.

File Locations:

The new project will not copy the source files into its folder, but instead will reference the location of the files in the v8 folder.

To create an independent MPLAB X IDE project, create a new project and copy the MPLAB IDE v8 source files to it.

Main Project:

Check the checkbox to make this project the main project on import.

Project Location:

The MPLAB X IDE project location is not in the MPLAB IDE v8 project folder, so this should be unchecked.

File Formatting:

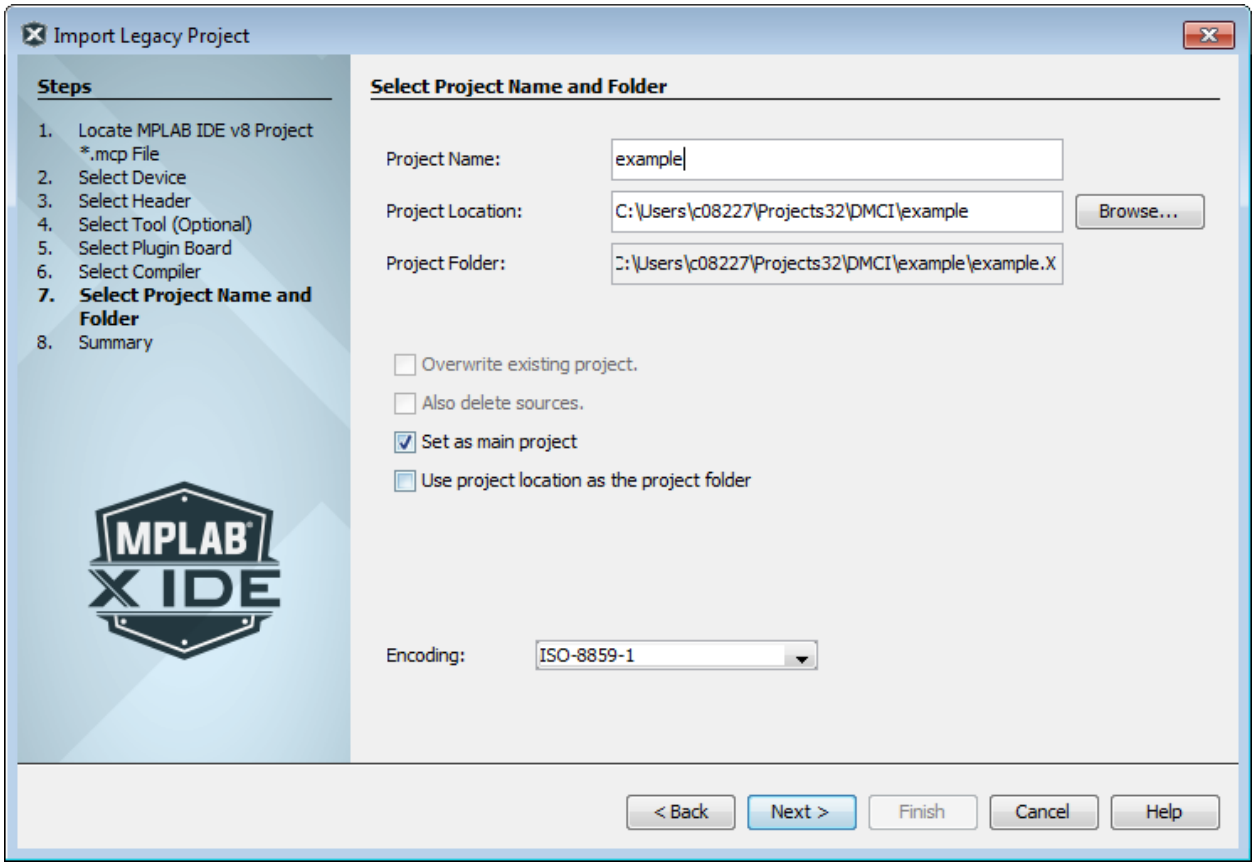
"ISO-8859-1" is the default character encoding used when importing a project from MPLAB IDE v8.

You should select the encoding that matches the one that is used in the imported project. For example, if the MPLAB IDE v8 format is "950 (ANSI/OEM – Traditional Chinese Big5)", then select "Big5" from the drop-down list.

Step 8. Summary. Review the summary before clicking **Finish**. If anything is incorrect, use the **Back** button to go back and change it.

The legacy project will open in the Projects window

Figure 5-7. Step 7 Dialog Example



5.4 Prebuilt Projects

Create a project from a prebuilt loadable image (Hex, ELF or COF files) by using the Import Image File wizard.

Note: To program a prebuilt image into a device, you will click the **Make and Program Device** icon even though it will only program the device (no make).

There are two ways to open this wizard: the Start Page and the New Project options.

Import Prebuilt Project Option

To open the "Import Image File" wizard, do one of the following:

- On the **Start Page**, click on the **Learn & Discover** or **My MPLAB X IDE** tab, "Projects" section, "Import Prebuilt" link.
- Select *File>Import>Hex/ELF (Prebuilt) File*.

New Project Option

For ways to open the New Project wizard, see 4.1.2 Launch New Project Wizard.

The wizard will launch to guide you through new project set up.

- **Step 1. Choose Project**: select the "Microchip Embedded" category and choose from the project type "Prebuilt (Hex, Loadable Image) Project." Click **Next**.
- The "Import Image File" wizard opens.

Import Image File Wizard

Follow the steps below to import your image file. The steps are numbered according to the "Import Prebuilt Project" option.

Click **Next** to move to the next step.

Step 1. Import Image File. Select the name and location of your image file. You may browse to a location.

Step 2. Select Device. Select the device you will be using in your application from the "Device" drop-down list. To narrow your selection list, choose a "Family" first.

Step 3. Select Header. This step will appear if a header is available for your selected device. To determine if a header is required for debug or if your device has on-board debug circuitry, consult one of the documents below. Then choose whether or not to use a header.

- Processor Extension Pak and Debug Header Specification
- Emulation Extension Pak and Emulation Header User's Guide

Step 4. Select Tool. Select the development tool you will be using to develop your application from the list. To determine support for your device, see 4.1.6.1 Project Tool Support.

Step 5. Select Project Name and Folder. Select a name and location for your new project. You may browse to a location. Click **Finish** when done.

The new project will open in the Projects window.

For information on exporting a project as hex, see the "Project Menu" in 12.15 Projects Window.

5.5 Loadable Projects, Files and Symbols

Use loadable projects and files to combine projects, combine hex files, combine projects and hex files or replace the project hex file. The Hexmate application is used to merge project or loaded hex files into one file. For more information on using this application, see the "*MPLAB XC8 C Compiler User's Guide*" (DS50002053) in the docs folder of the installed MPLAB XC8 C compiler. See also Section "HEXMATE Conflict Report Address Error" in 9.5 Errors.

Use loadable symbols to load debug symbols during a production build or program, as when using RTDM with DMCI, Motor Control applications, etc.

Loadable projects or files are useful for creating combined bootloader and application code. See 5.6 Loadable Projects and Files: Bootloaders.

The combinations of current projects and loadables are listed below.

Table 5-2. Loadable Combinations

Current Project	Loadable	Caveat
Standalone, Existing MPLAB IDE v8, Library	Standalone	None
	Hex file	None
	ELF or COF file	Can debug, but not build. An error will be displayed in the Output window.
Prebuilt (Hex)	Hex file	No auto checking for overlapping memory areas. Can debug, but not build. Build button will be disabled.
	ELF or COF file	
Prebuilt (ELF or COF)	Hex file	
	ELF or COF file	

The options are listed below.

Table 5-3. Loadable Options

Add Loadable Project(s)	Load one or more existing projects into your current project. When you build your current project, all projects will be built and the hex files will be combined into one. All debug files will be combined as well (ELF or COF).
Add Loadable File(s)	Load one or more existing hex files into your current project. When you build your current project, the hex file will be combined with the other hex files into one file. Note: You will no longer be able to debug the project that contains hex file(s). Use Loadable Projects for debugging.
Add Alternate File	Load an alternate hex file to be used. This options provides a post-build step where you may copy or move your project hex file to another location, use a tool such as HEXMATE to merge your file with another hex file, and then load the file back into the IDE.

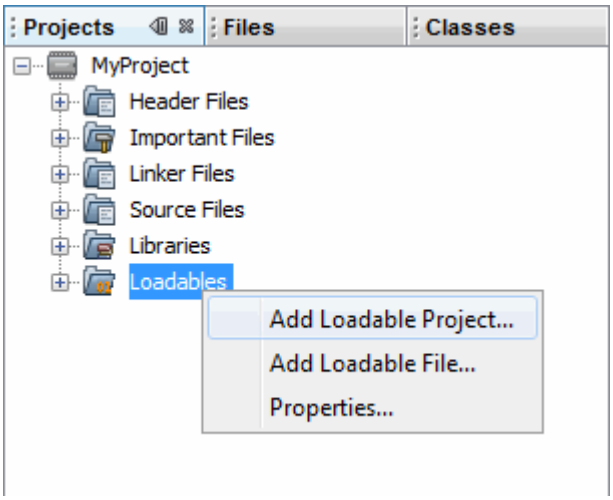
5.5.1 Projects Window – Loadables Setup

Right Click on the "Loadables" folder in the Projects window (see figure) and select an option:

- Add Loadable Project – select to add an existing project to your current project. Repeat to add additional projects.
- Add Loadable Files – select to add an existing hex file to your current project. Repeat to add additional hex files.
- Properties – Open the Project Properties window for Loading. See 5.5.2 Project Properties Window – Loading Setup.

Build your current project to build all projects and combine hex files into one. Any debug files will also be combined.

Figure 5-8. PROJECTS WINDOW – LOADABLES FOLDER

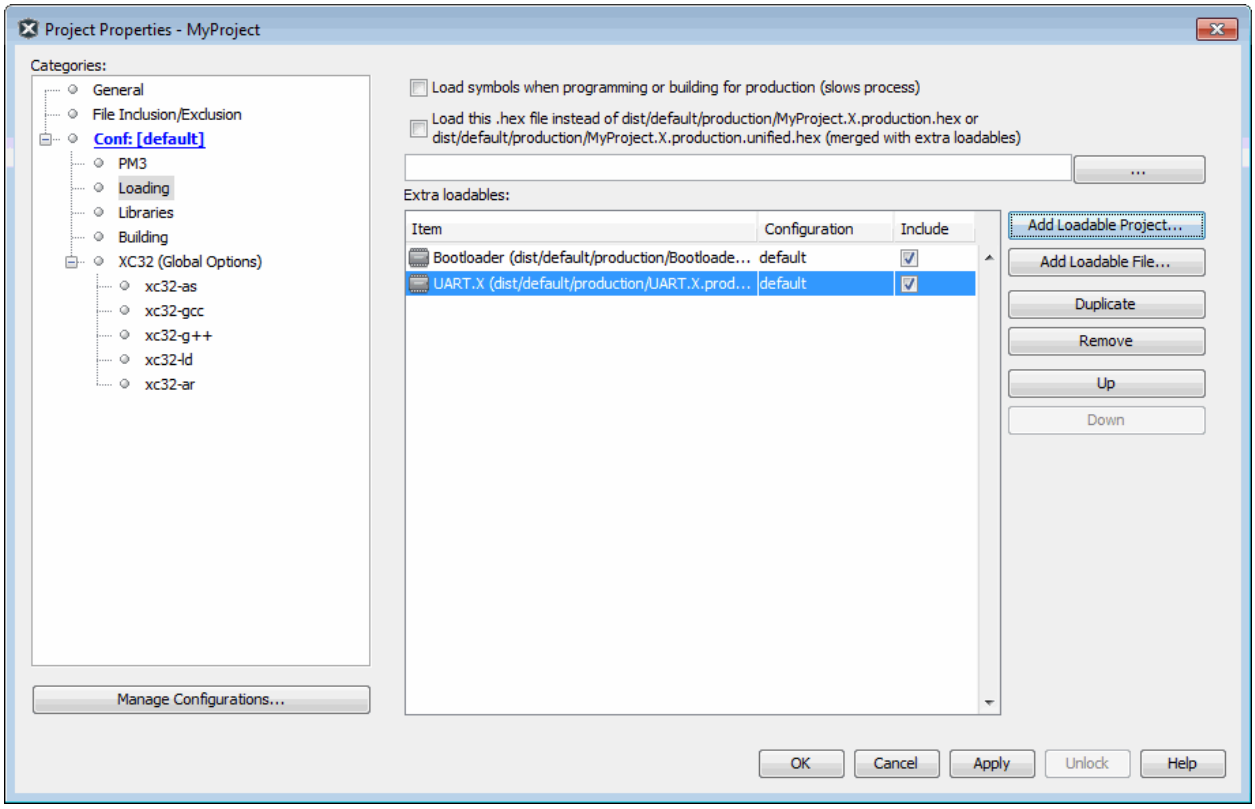


5.5.2 Project Properties Window – Loading Setup

Open the Project Properties window (*File>Project Properties*) and click on "Loading."

- Combining the Current Project with Other Projects
- Combining the Current Project Hex File with Other Hex Files
- Loading an Alternative Hex File
- Loading Debug Symbols During Program/Build

Figure 5-9. Project Properties – Loading

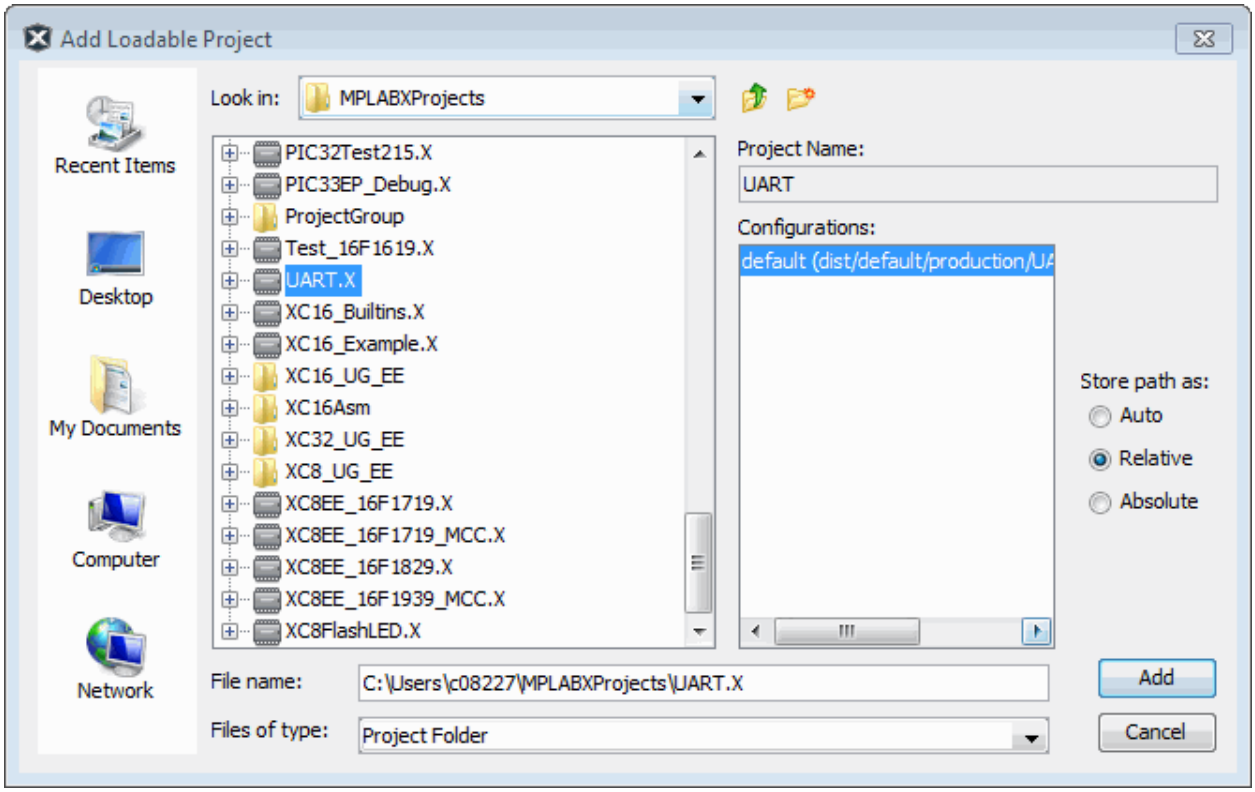


Combining the Current Project with Other Projects

1. Click **Add Loadable Project**. The Add Loadable Project dialog will appear.
2. Browse to a project and select it.
3. Under "Configurations" select a build configuration to use for this project. If you have not added different configurations to this project, you will only see "default."
4. Under "Store path as" select how to store information about the path to the loadable project. Choose from:
   - 4.1. Auto – allow MPLAB IDE to determine whether the loadable project path should be relative or absolute based upon the its location.
     If the main project directory contains the loadable project, the reference is a relative path. Otherwise, the reference is an absolute path.
   - 4.2. Relative – reference the loadable project with a relative (to the main project) path.
     This is the preferred way to add paths.
   - 4.3. Absolute – reference the loadable project with an absolute path.
     There are cases when this mode is useful, but these paths often require correction when a project is moved from one system to another.
5. Click **Add** to close the dialog.
6. In the Project Properties window, ensure the "Include" checkbox is checked if you want to build this project when you build the current project.
7. If you have added more than one project, the order shown here will determine the order in which the hex files will be added to the current project's hex file. Use **Up** and **Down** to change the order.
8. Click **Apply** or **OK** to accept the changes.

The next time you build the current project, the projects listed here will also be built (if "Include" is checked) and their hex files will be combined with the current project's hex file to create a single output hex file. Any debug files will also be combined.

Figure 5-10. Add Loadable Project



Combining the Current Project Hex File with Other Hex Files

1. Click **Add Loadable File**. The Add Loadable File dialog will appear.
2. Browse to a hex file and select it.
3. Under "Store path as" select how to store information about the path to the loadable file. Choose from the following options:
   - 3.1. Auto – Allow MPLAB IDE to determine whether the loadable file path should be relative or absolute based upon the its location.
     If the main project directory contains the loadable file, the reference is a relative path. Otherwise, the reference is an absolute path.
   - 3.2. Relative – Reference the loadable file with a relative (to the main project) path.
     This is the preferred way to add paths.
   - 3.3. Absolute – Reference the loadable file with an absolute path.
     There are cases when this mode is useful, but these paths often require correction when a project is moved from one system to another.
4. Click **Add** to close the dialog.
5. In the Project Properties window, if you have added more than one file, the order shown here will determine the order in which the hex files will be added to the current project's hex file. Use **Up** and **Down** to change the order.
6. Click **Apply** or **OK** to accept the changes.

The next time you build the current project, the hex files listed here will be combined with the current project's hex file (`CurrentProject.X.Production.hex`) to create a single output hex file. (`CurrentProject.X.unified.hex`) in the same folder as the current project hex file.

Loading an Alternative Hex File

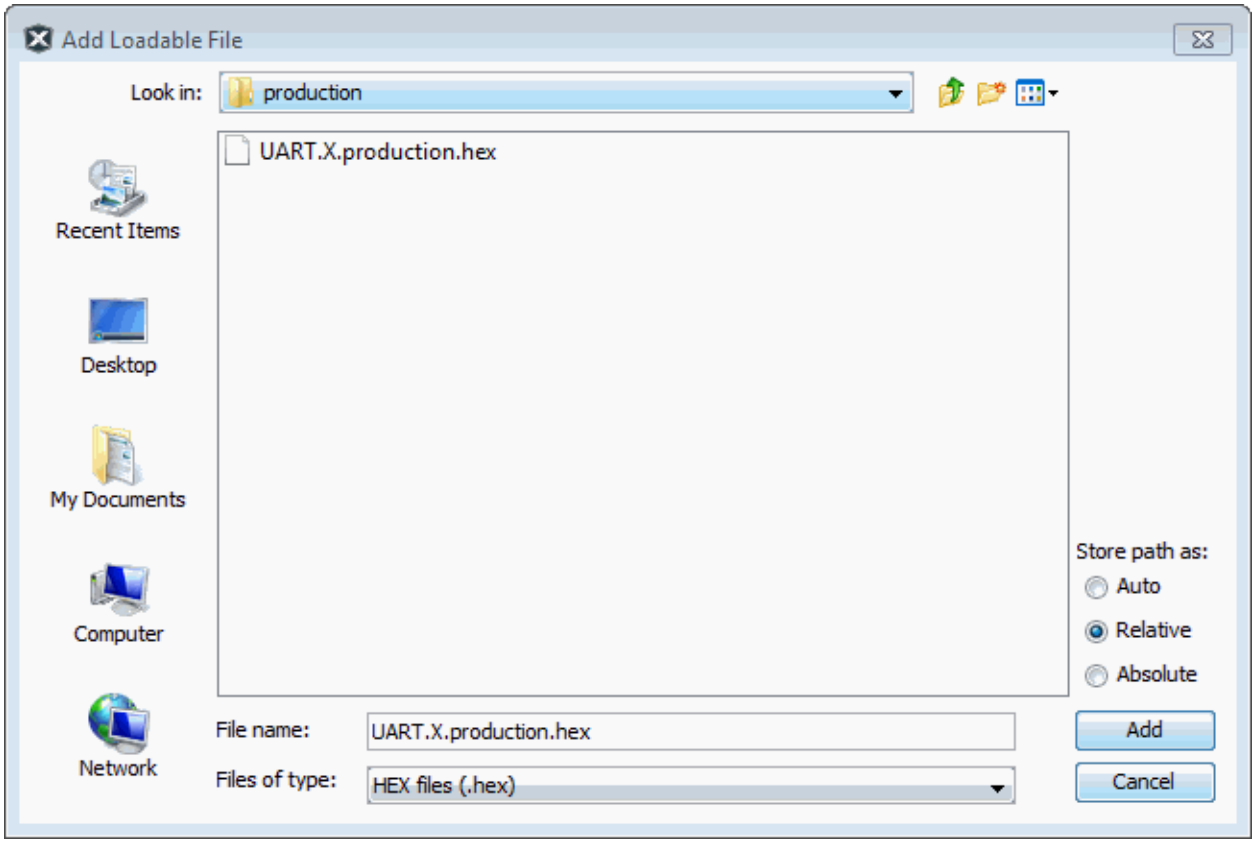
1. Click to check "Load this .hex file instead of ..."
2. Click the ellipses (...) button to open the Add Loadable File dialog. For details on adding the hex file using this dialog, see the previous topic.

The next time you build the current project, the alternative hex file will load on build complete.

Loading Debug Symbols During Program/Build

The checkbox to "Load symbols when programming or building for production (slows process)" allows you to load debug symbols when programming, as when using RTDM with DMCI, Motor Control applications etc. Otherwise the checkbox should be left unchecked to speed up programming and building (default).

Figure 5-11. Add Loadable File



5.5.3 The Preferred Method to Use Loadables

The recommended way to use loadables is:

1. Select the project that sets the device configuration bits and initialization as the main project (right click and select "Set as Main Project"). The loadable should not have device configuration settings as this will conflict with the main project.
2. Add the other projects as loadables to this main project. If the project being loaded has more than one project configuration, be sure to specify that when loading.

Specifying the project containing the loadables as the main project ensures that a change in any loadable will be picked up by the build when the Build button is clicked.

5.6 Loadable Projects and Files: Bootloaders

To combine a bootloader with application code:

1. Create one project for your application and one project for your bootloader.
2. Load the bootloader project or hex file into the application project. See 5.5.1 Projects Window – Loadables Setup or 5.5.2 Project Properties Window – Loading Setup for how to do this.

The next time you build your application project, the resulting hex file will be a combined bootloader/application hex file. Any debug files will also be combined.

For build errors, see 5.5.3 The Preferred Method to Use Loadables or the sections below.

MPLAB C Compiler for PIC18 MCUs (MPLAB C18)

This compiler provides application start-up code (`c018x.o`) that begins at the reset vector (address 0) for use in initializing the software stack, optionally initializing the `idata` section, and jumping to `main()`. If this start-up code is left in an application, there will always be a conflict with the bootloader code reset and you will get a linker error message about a data conflict.

A resolution would be to edit the start-up code to begin at an address other than 0.

MPLAB XC8 – PIC18 MCU Example

The following Microchip webinar details how to combine a bootloader with application code for a PIC18 MCU using MPLAB XC8 and MPLAB X IDE:
Linking PIC18 Bootloaders and Applications

MPLAB XC16 Bootloader Example

See the "*MPLAB XC16 C Compiler User's Guide*" (DS50002071) or Help file for an example bootloader.

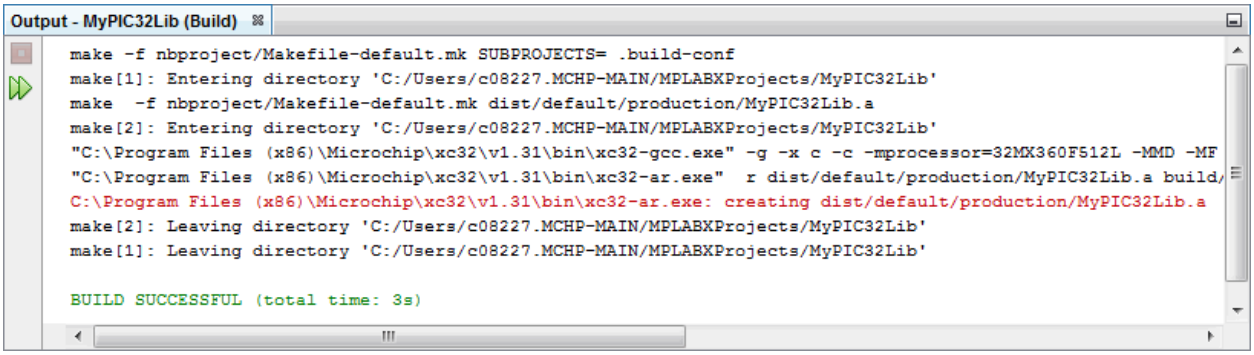
MPLAB XC32 Bootloader Example

See the "*PIC32 Bootloader Application Note*" (DS01388) on the Microchip website.

5.7 Library Projects

Create a project that will generate a library file (`*.lib` or `*.a` depending on which compiler you use) instead of a hex file (see text in red). This project will not be able to run on its own as it will not have a `main()` function. The resulting library will be used in another project that has its own `main()` function.

Figure 5-12. Library Project Example



For ways to open the New Project wizard, see 4.1.2 Launch New Project Wizard.

The wizard will launch to guide you through new project set up. Click **Next** to move to the next step.

Step 1. Choose Project. Select the "Microchip Embedded" category and choose from the project type "Library Project."

Step 2. Select Device. Select the device you will be using in your application from the "Device" drop-down list. To narrow your selection list, chose a Family first.

Step 3. Select Header. This step will appear if a header is available for your selected device. To determine if a header is required for debug or if your device has on-board debug circuitry, consult either:

- Processor Extension Paks and Debug Headers
- Emulation Extension Paks and Emulation Headers

Step 4. Select Tool. Select the development tool you will be using to develop your application from the list. To determine support for your device, see 4.1.6.1 Project Tool Support.

Step 5. Select Compiler. Select the language tool (compiler) you will be using to develop your application from the list. To determine support for your device, see 4.1.6.1 Project Tool Support.

Step 6. Select Project Name and Folder. Select a name and location for your new project. You may browse to a location. Click **Finish** when done.

The new project will open in the Projects window.

You can change your Library project to a Standalone (Application) project if you wish. For details, see 4.10.1 Change Project Configuration Type.

5.8 Import Atmel Studio 7 or Atmel START Project

Use the New Project or Import wizard to import either an Atmel Studio 7 Project or an Atmel START export file for MPLAB X IDE.

There are two ways to open the wizard, described below.

Import Project Option

To open the Import Atmel Studio or START Project wizard, select either:

- `File>Import>Atmel Studio Project`
- `File>Import>START MPLAB Project`

New Project Option

To open the New Project wizard, do one of the following:

- On the Start Page, click on the **Learn & Discover** or **My MPLAB X IDE** tab, "Projects" section, "Create New" link.
- Select *File>New Project* (or Ctrl+Shift+N).
- Click on the "New Project" icon on the toolbar.

A wizard will launch to guide you through new project set up.

Step 1. Choose Project: select the "Microchip Embedded" category and choose from the project type either "Import Atmel Studio Project" or "Import Atmel START MPLAB Project." Click **Next**.

5.8.1 Import Atmel Project - Locate Project File

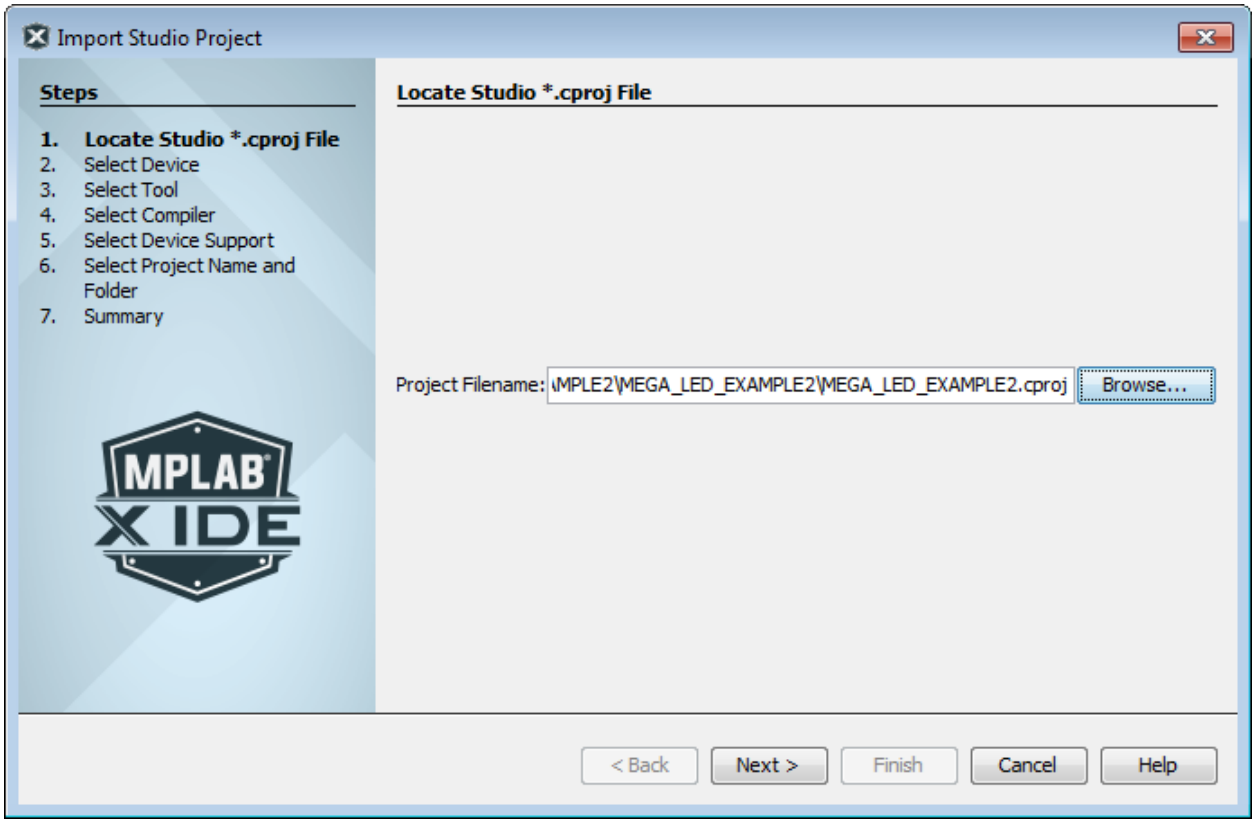
Step 1 or 2. Locate Atmel Studio or START Project File

Enter or browse to the requested file. For example, on Windows 7 OS, project files may be found under:

`C:\Users\Admin\Documents\Atmel Studio\7.0\`

For Atmel Studio, find the `*.cproj` or `*.cppproj` project file. For Atmel START, find the `*.atzip` project file.

Figure 5-13. Import Atmel Studio Project

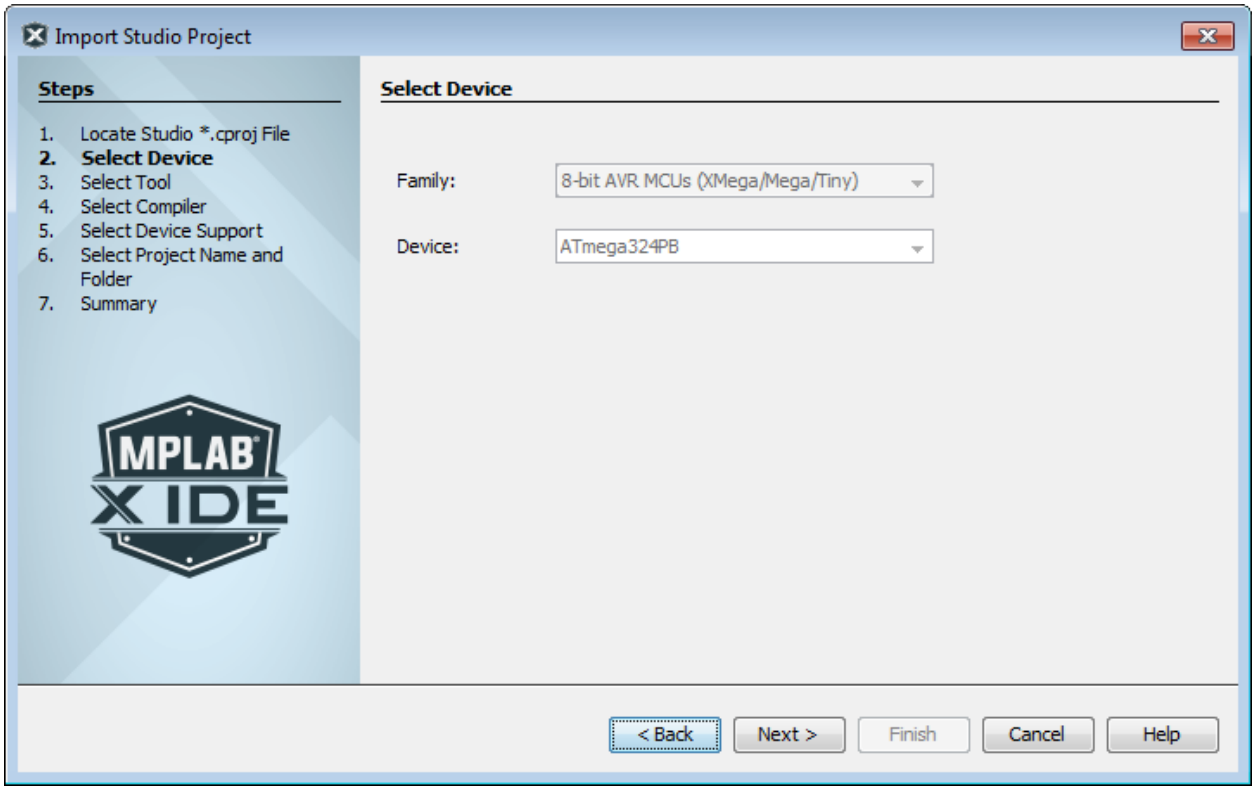


5.8.2 Import Atmel Project - Select Device

Step 2 or 3. Select Device

This window will be populated with the device information from the imported project.

Figure 5-14. Import Atmel Project - Select Device



5.8.3 Import Atmel Project - Select Tool

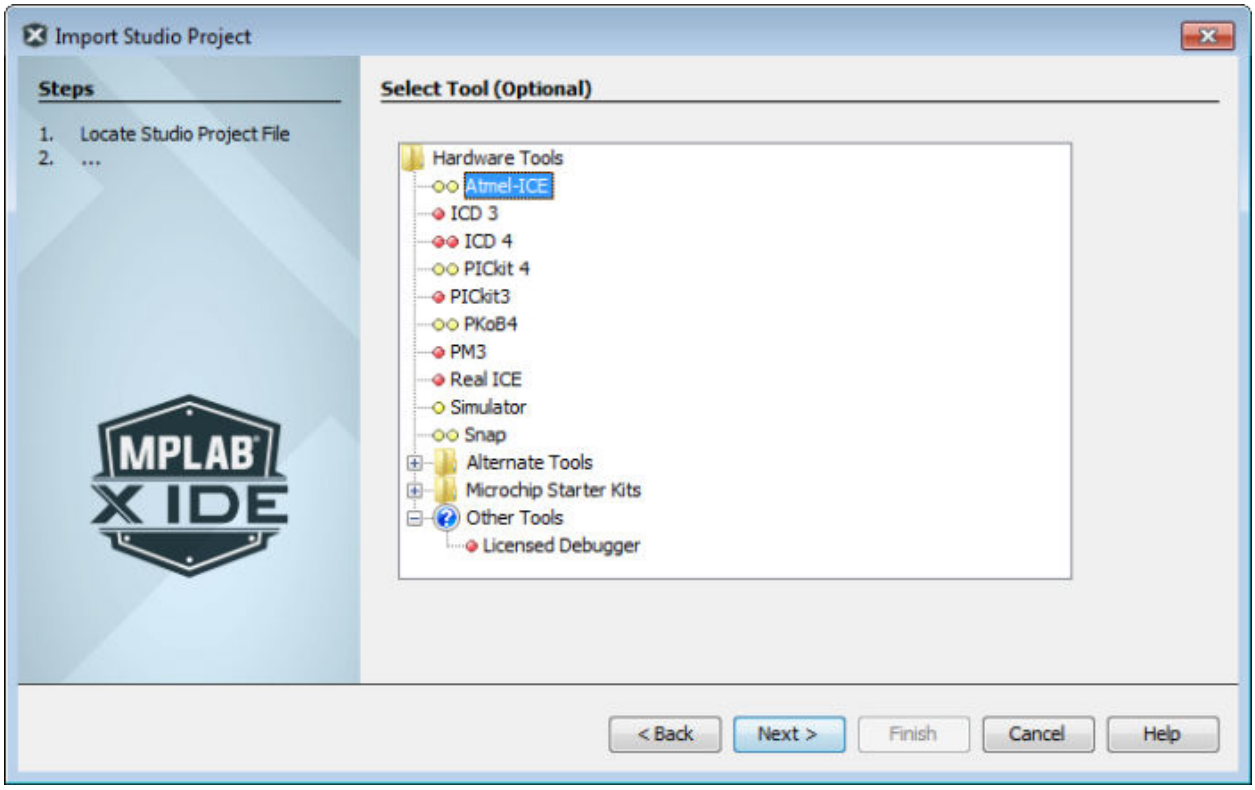
Step 3 or 4. Select Tool

This window will be populated with the debug tool from the Atmel project, if it is available in MPLAB X IDE. You may select a different tool from the list, but you may want to keep the original project tool until after import to ensure your project is working correctly. Then you can select a different tool for your project if you want.

The level of tool support for the selected device will be displayed next to the tool name.

For more on this display, see "Basic Tasks>Create New Project."

Figure 5-15. Import Atmel Project - Select Tool



5.8.4 Import Atmel Project - Select Compiler

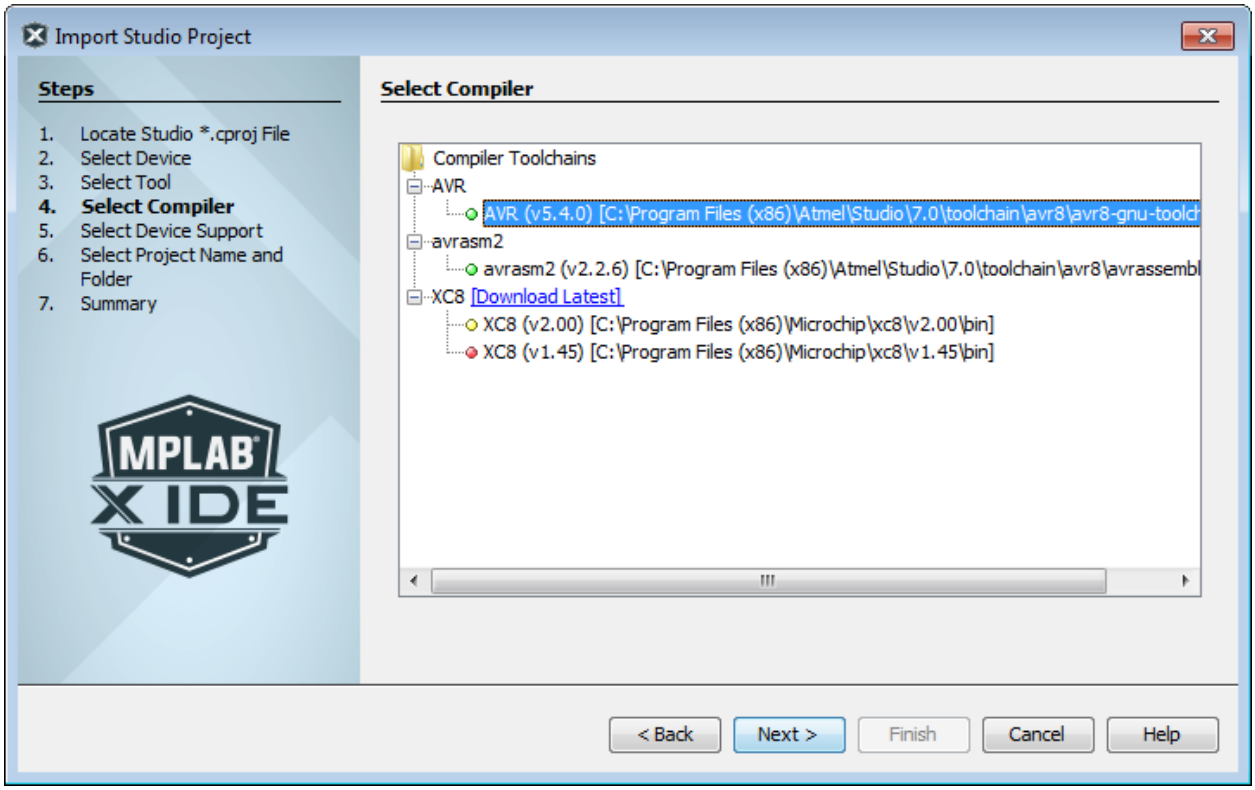
Step 4 or 5. Select Compiler

This window will be populated with the compiler from the Atmel project, if it is available in MPLAB X IDE. You may select a different compiler from the list, but you may want to keep the original project compiler until after import to ensure your project is working correctly. Then you can select a different compiler for your project if you want.

The level of tool support for the selected device will be displayed next to the tool name.

If you do not see a compatible compiler listed, click **Cancel** and follow the directions under "Basic Tasks>Create New Project."

Figure 5-16. Import Atmel Project - Compiler Selected



5.8.5 Import Atmel Project - Select Device Support

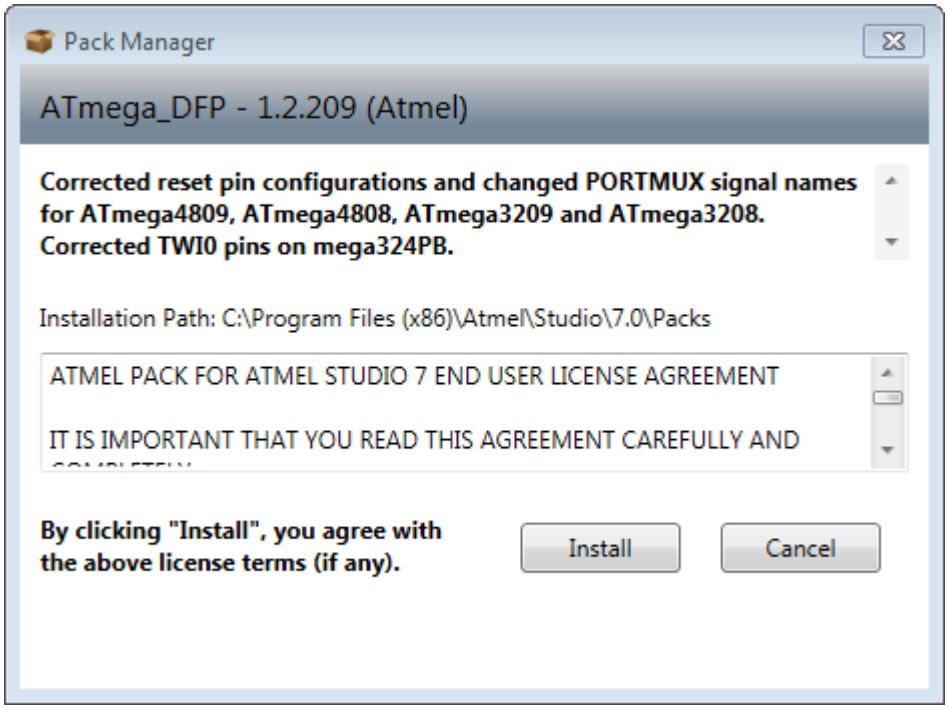
Step 5 or 6. Select Device Support

As of MPLAB X IDE v5.00, devices are supported in versioned device packs. If the device pack version is not supported in MPLAB X IDE:

1. For "Override Default Device Support" checked, you can download the pack support by clicking on the help icon (question mark) and following the instructions. When you download the pack, it will install in Atmel Studio (see figure).

Figure 5-17. Import Atmel Project - Select Device Support

Figure 5-18. Device Pack Installation



2. For "Override Default Device Support" checked, you can look in the Atmel Studio pack directory for the correct version. For example:

   `C:\Program Files (x86)\Atmel\Studio\7.0\packs\atmel\ATmega_DFP\1.2.209`

3. For "Override Default Device Support" unchecked, you can use the MPLAB X IDE device pack version (the latest) in the project. For example, the MPLAB X IDE project packs would be located under:

```
C:\Program Files (x86)\Microchip\MPLABX\v5.xx\packs\Atmel\ATmega_DFP
```

5.8.6 Import Atmel Project - Select Project Name and Folder

Step 6 or 7. Select Project Name and Folder

It is recommended that you do not change the default name and location to preserve maintainability of both projects.

Project Files Location:

For Atmel Start, source files will be copied into the MPLAB X IDE project.

For Atmel Studio, the new project will not copy the source files into its folder, but instead will reference the location of the files in the Atmel project folder.

To create an independent MPLAB X IDE project, create a new project and copy the source files to it.

Main Project:

Check "Set as main project" to make this project the main project on import.

Project Location:

For Atmel Studio, it is not recommended to check "Use project location as the project folder," but to keep the MPLAB X IDE project with the Atmel project. For Windows OS systems, if the Atmel project path is too long (see warning below), change the path of this project first before importing into MPLAB X IDE.

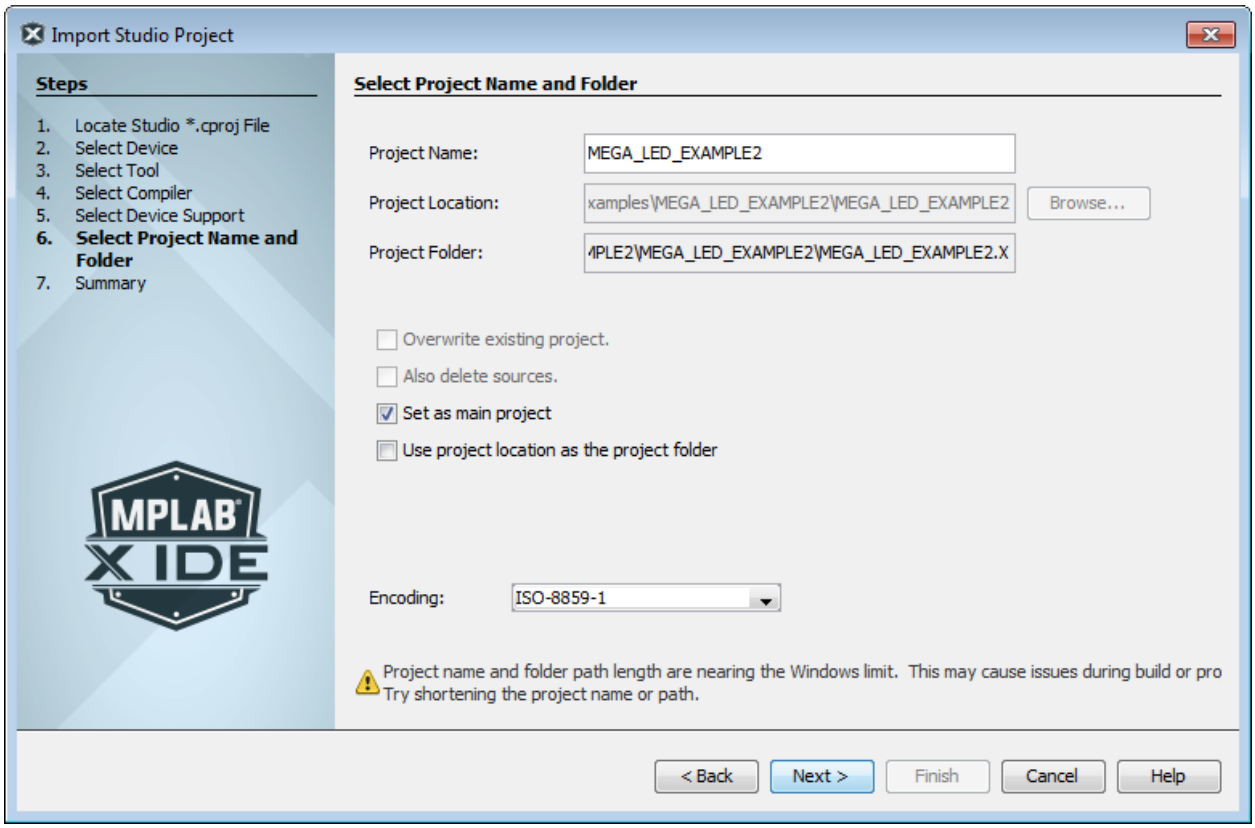
File Formatting/Encoding:

You should select the encoding that matches the one that is used in the imported project. For example, if the format is "950 (ANSI/OEM – Traditional Chinese Big5)," then select "Big5" from the drop-down list.

Path Length (Window OS):

If the path to your project is too long, you may need to shorten it to avoid issues during build.

Figure 5-19. Import Atmel Project – Select Project Name and Folder



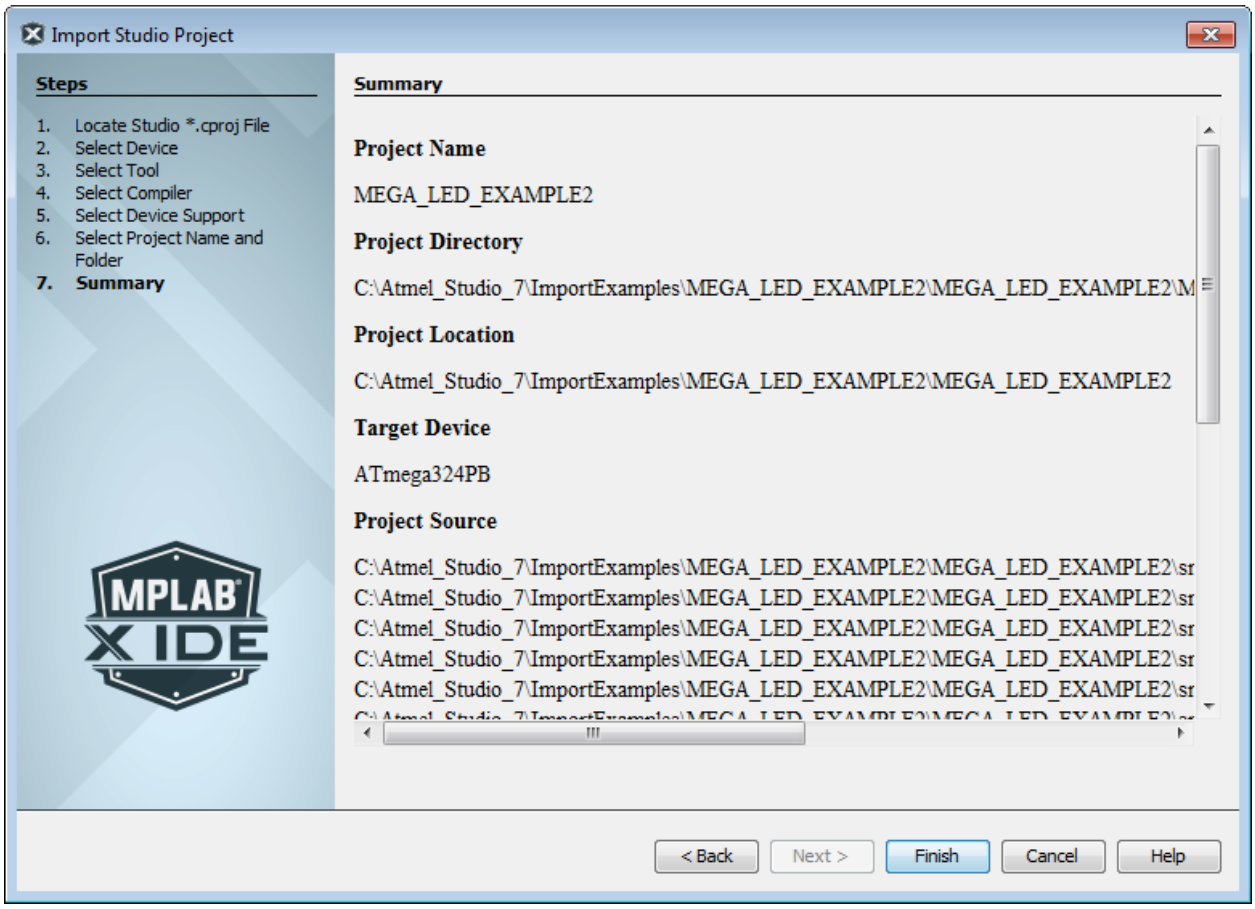
5.8.7 Import Atmel Project - Summary

Step 7 or 8. Summary

Review the summary before clicking **Finish**. If anything is incorrect, use the **Back** button to go back and change it.

When the project is being imported, observe any warnings or other messages in the Output window. You may need to make changes so your new project will build.

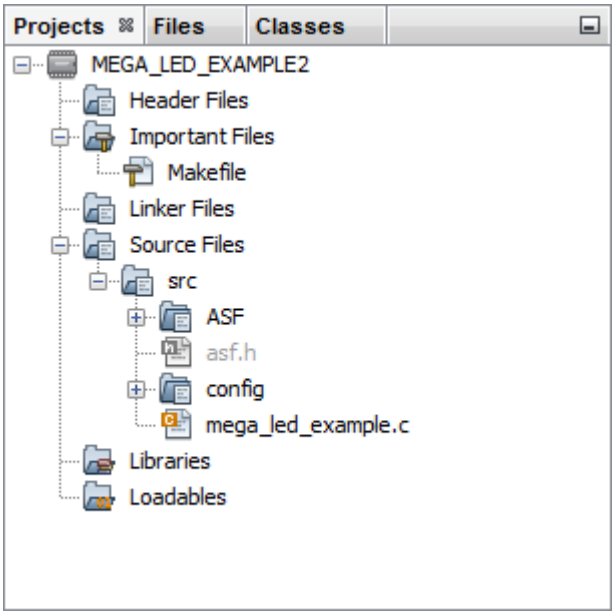
Figure 5-20. Import Atmel Project – Summary



The new project will open in the Projects window.

Note: The source files from the original Atmel project are linked, as opposed to copied. So modifications to the source files in either the Atmel or MPLAB X IDE projects will be noticed in both.

Figure 5-21. Import Atmel Project – Projects Window



5.9 Other Embedded Projects

MPLAB X IDE can create a project from selected, other embedded projects.

1. Select *File>New Project*.
2. Click on "Other Embedded" under "Categories" and select from a list of available embedded projects.
3. Continue to create an MPLAB X IDE project.

This feature imports your existing files into an MPLAB X IDE project. Conversion of other embedded project settings or code is not yet available.

For information on how to work with MPLAB X IDE, see:

- Tutorial
- Basic Tasks

For information on available compilers, see:

https://www.microchip.com/mplab/compilers

5.10 Sample Projects

Create a sample project to help you learn about Microchip devices, tools and MPLAB X IDE.

1. Select *File>New Project*.
2. Click on "Samples>Microchip Embedded" under "Categories" and select from a list of available embedded projects (that blink demo board lights) or template projects. Read the Description for more information.

Part numbers for demo boards are as follows:

- Explorer 16 Demo Board: DM240001
- PICDEM 2 Plus: DM163022-1

5.11 Work with Other Types of Files

When selecting *File>New File*, you are presented with many types of files. Using Microchip compiler files has been explored previously. However, there are other types of files you can select depending on your project language tool or need to create a specific file.

Table 5-4. File Types

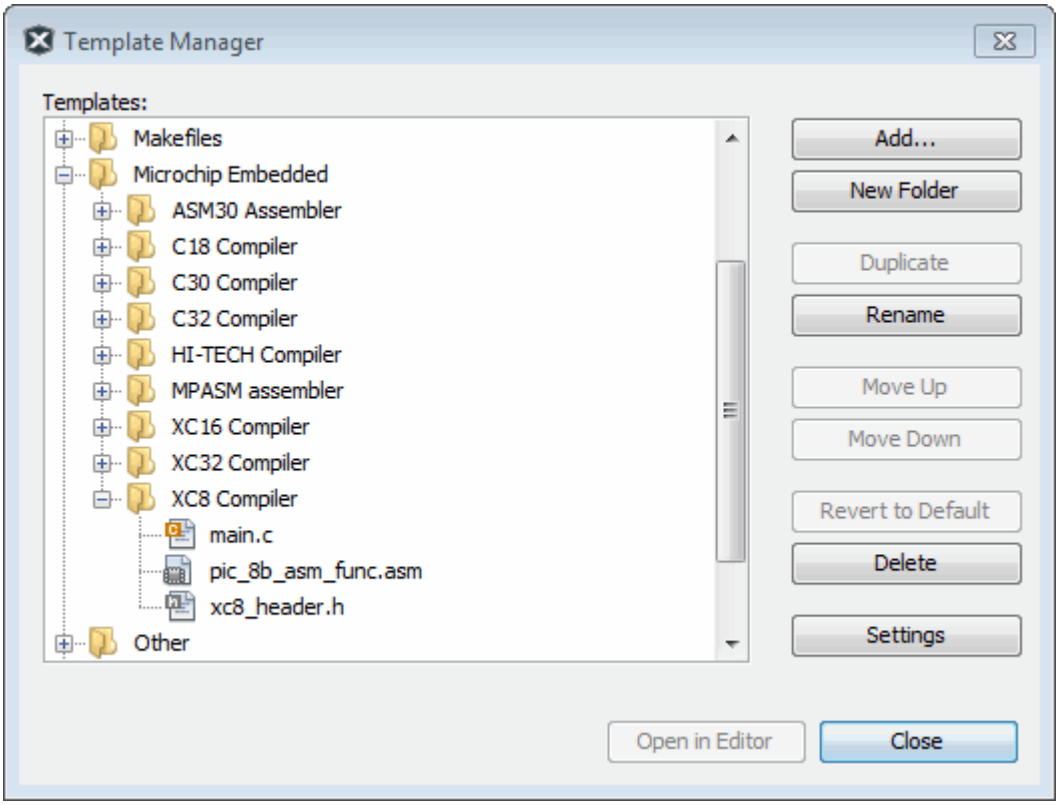
Category	File Type
Microchip Embedded	Files for language tools that are supported in MPLAB X IDE Select your compiler folder to see the available file types.
C	Generic C files
C++	Generic C++ files
Assembler	Generic assembly files
Shell Scripts	Shell script files: Bash, C, Korn, etc.
Makefiles	Makefile files
XML	XML files
Other	Other types of files, such as HTML, JavaScript, etc. If you do not see the type of file you want listed here, select "Empty File." In the next window, name the file with the desired extension.

5.12 Modify or Create Code Templates

When you create a file to add to your project (4.9.2 Create a New File), a template is used for the new file. To change this template, select *Tools>Templates* and then "Open in Editor" to edit a template. You may also use "Add" or "Duplicate" in this dialog to create new templates.

"New Folder" can be used to create a new folder to hold templates. Be aware that MPLAB X IDE filters out all but Microchip Embedded, Shell Scripts, Makefiles and Other, so files or folders should be created under those folders.

Figure 5-22. Template Manager



You may set template options by selection *Tools>Options* (*MPLAB X IDE>Preferences* for macOS), **Editor** button, **Code Templates** tab (as shown below).

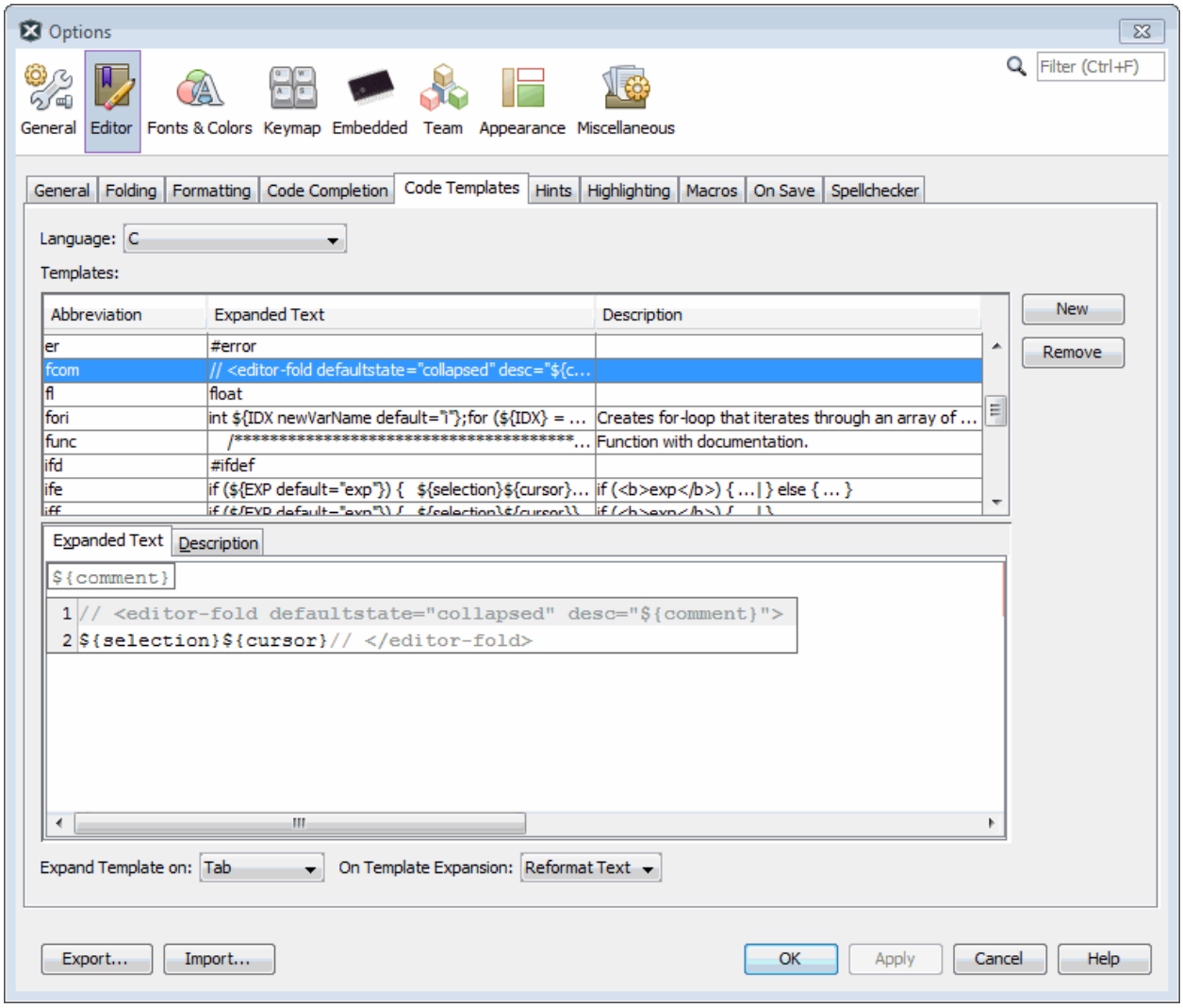
An option of note for C code is the "fcom" option. In an Editor window, type "fcom" and then press "Tab" to insert the following text into the source code:

```
// <editor-fold defaultstate="collapsed" desc="${comment}">
${selection}${cursor}// </editor-fold>
```

This option allows you to hide/view sections of code.

In the **Code Template** tab, selecting "fcom" will display "${comment}" in the "Expanded Text" section. Hover over this text to see the full text.

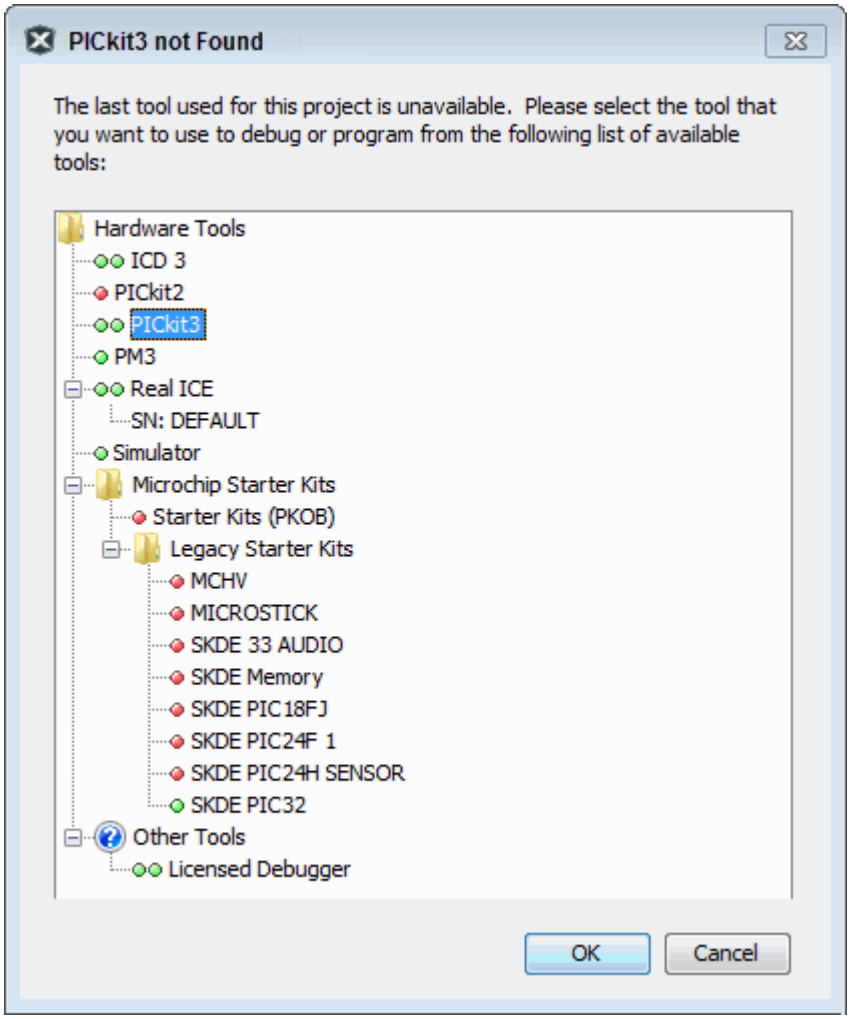
Figure 5-23. Template Options



5.13 Switch Hardware or Language Tool

When attempting to Run or Debug a project that has a debug tool selected, that is not the one currently connected, a dialog displays, allowing you to select another hardware tool for the project.

Figure 5-24. Switch Hardware Tool Dialog



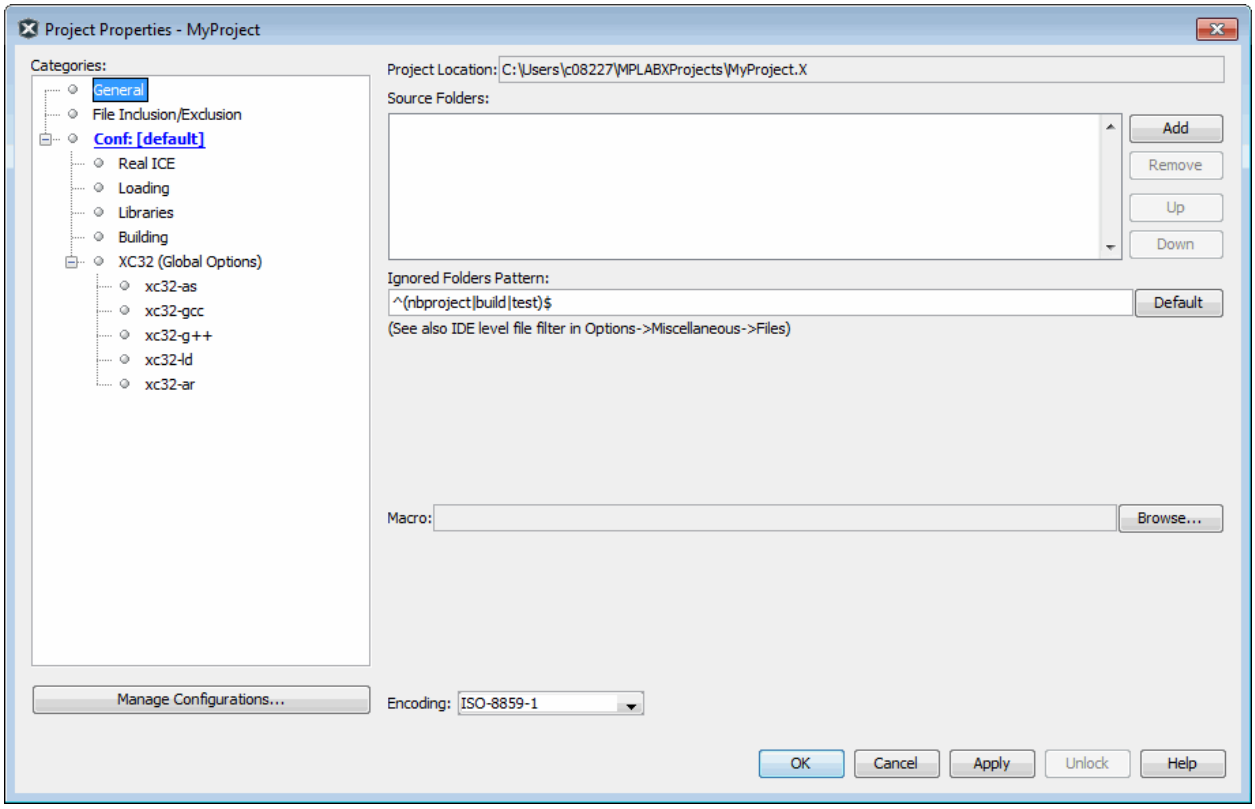
You may also plug in two or more hardware tools and switch between them in the Project Properties window (see 4.3 Open Project Properties).

To switch between different versions of compiler toolchains (language tools), use the Project Properties window.

5.14 Modify Project Folders and Encoding

When you created your project, you specified the project folder and encoding. Once the project is created, you may add or ignore project folders and change the project encoding using the "General" category in the Project Properties window.

Figure 5-25. Project Properties – General



Project Location

View the current project location. See 8.9 Moving, Copying or Renaming a Project to change the project location.

Source Folders

Add folders for MPLAB X IDE to search when looking for project files.

Note: Adding files outside the project folder may make the project less portable.

Ignored Folders Pattern

Ignore folders in the project folder according to the regular expression pattern specified.

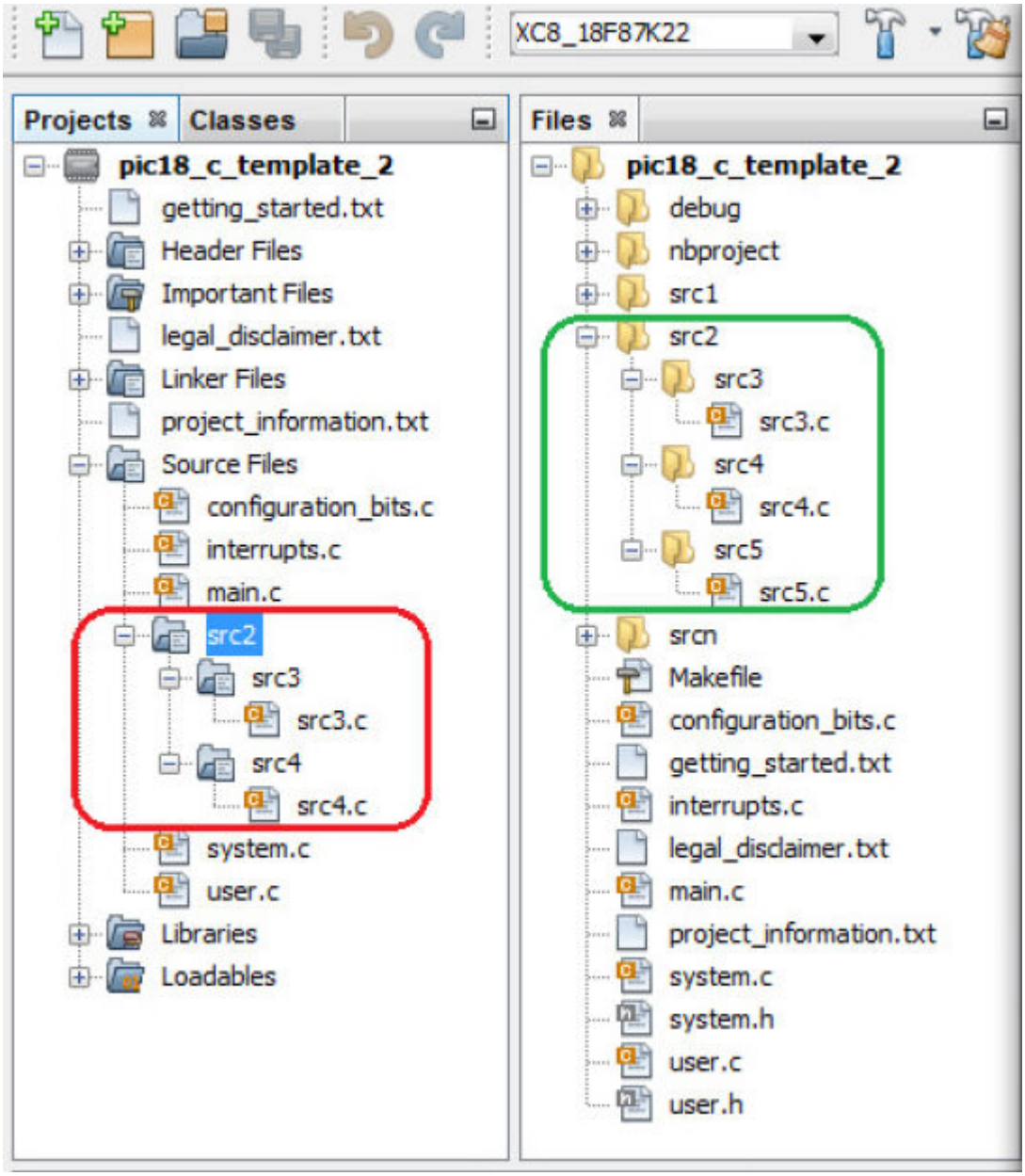
***Example*:** Exclude folder `src5`

To exclude the `src5` folder and its contents from the project, enter the following regular expression in the "Ignored Folders Pattern" text box:

```
^(src5|nbproject|build|test)$
```

View the Projects window to see no `src5` folder. View the Files window to see it does exist, but has been excluded.

Figure 5-26. Exclude Folder from Project



When the project is built, ensure that there is no mention of the `src5` folder or `scr5.c` file contained in the Build tab of the Output Window.

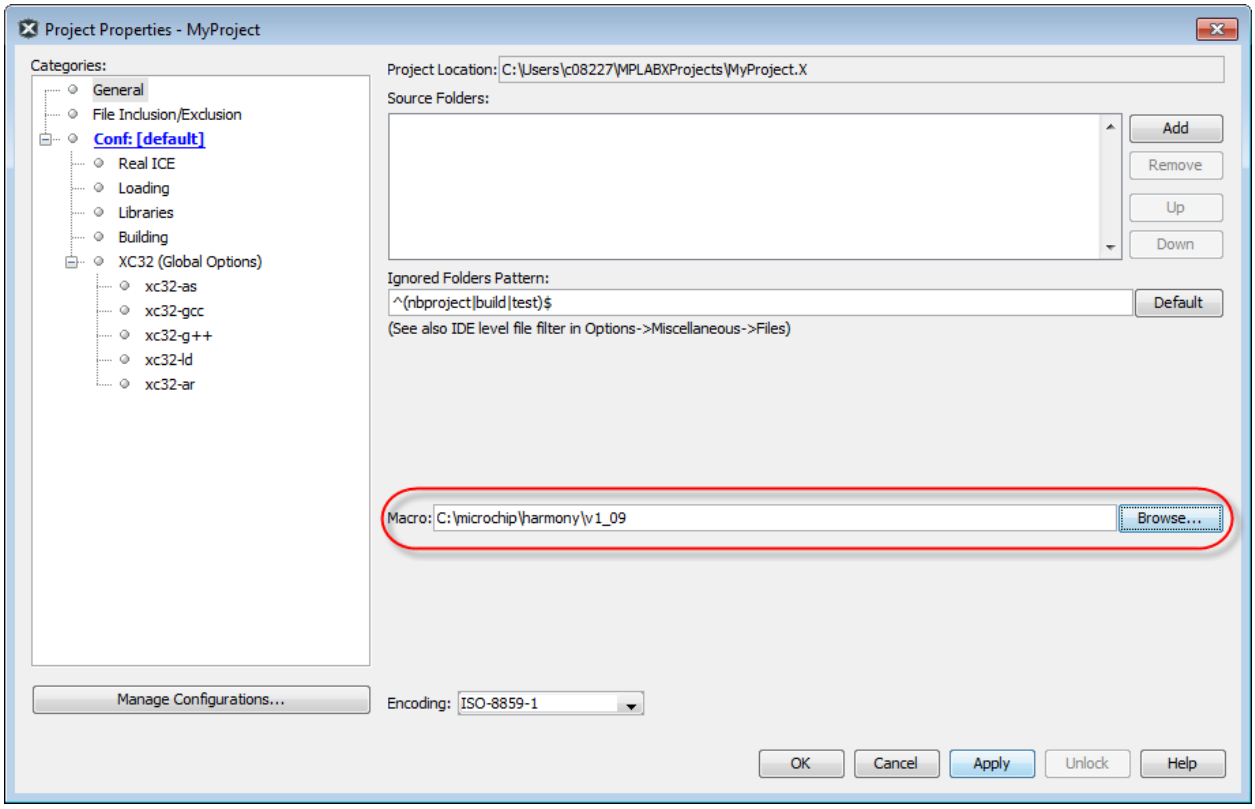
Macro

Enter a path to source files to be used in the "Add Existing Items" selection dialog to link to those specified source files.

Example: MPLAB Harmony Files

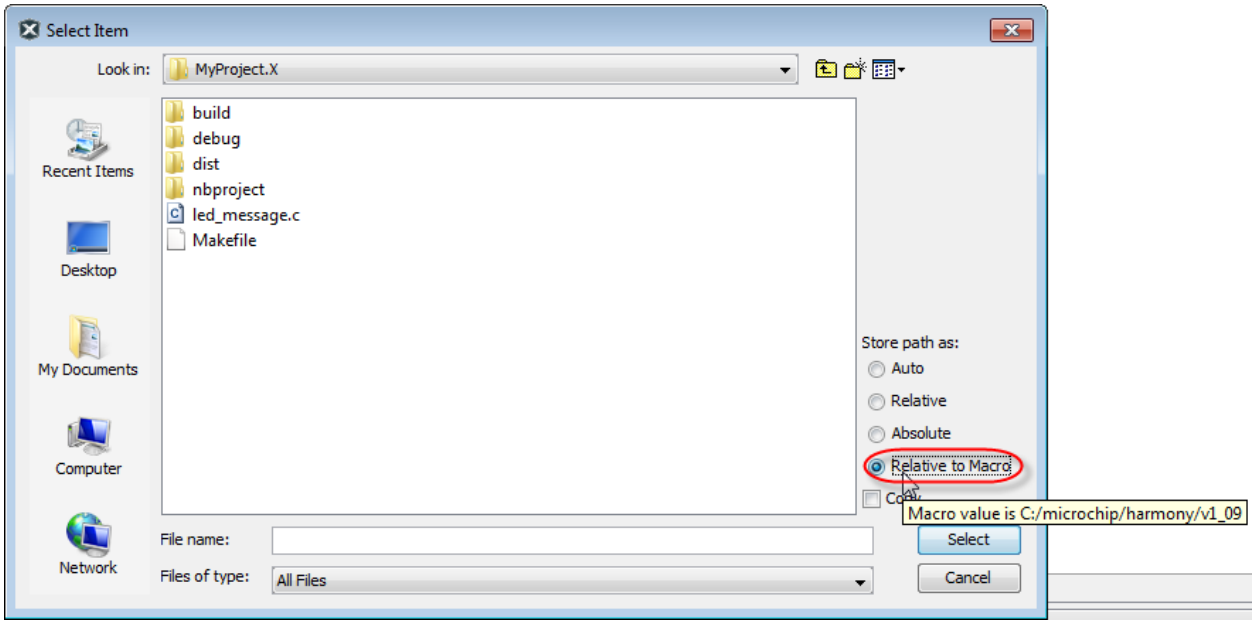
Select a path to MPLAB Harmony source files.

Figure 5-27. Enter Path to be Used for Macro



Go to the Projects window, right click on the "Source Files" folder and select one of the options for "Add Existing Items." In the dialog is an option to choose files from the Macro. Hover over "Relative to Macro" to see macro value.

Figure 5-28. Choose to Select Files from Macro



Encoding

Change the project encoding. This selection will specify the code syntax coloring, which can be edited under *Tools>Options* (*MPLAB X IDE>Preferences* for macOS), **Fonts and Colors** button, **Syntax** tab.

5.15 Use the Stopwatch

Use the stopwatch to determine the timing between two breakpoints.

To use the Stopwatch:

1. Add a breakpoint where you want to start the stopwatch.
2. Add another breakpoint where you want to stop the stopwatch.
3. Select *Window>Debugging>Stopwatch*. Click on the **Properties** icon on the left of the window and select the start and stop breakpoints.
4. Debug the program again to get the stopwatch timing result.

Figure 5-29. Stopwatch Setup

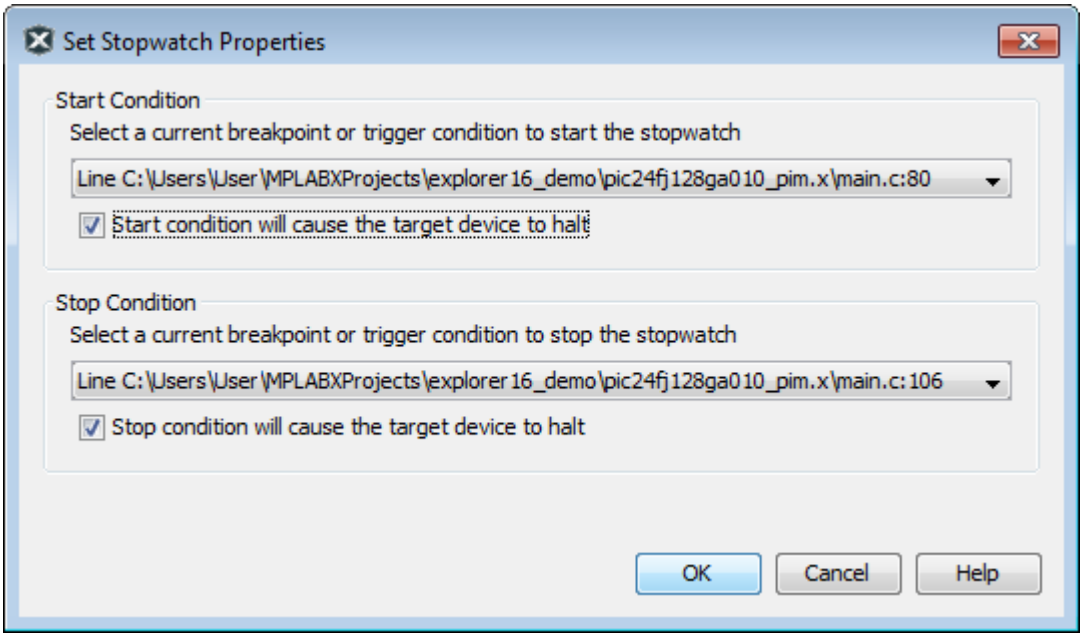
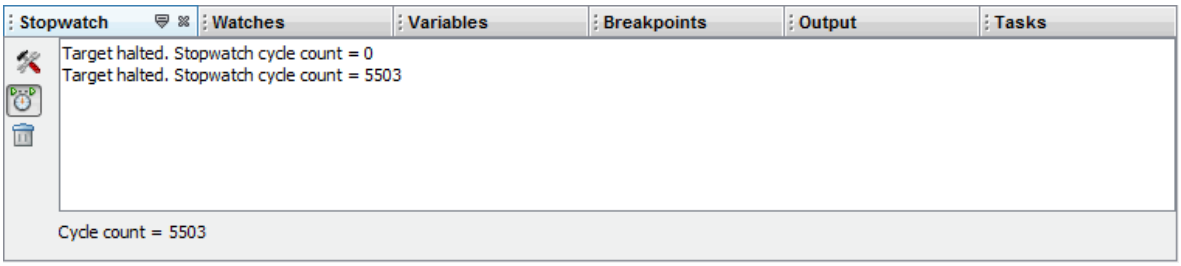


Figure 5-30. Stopwatch Window with Content



The stopwatch has the following icons on the left side of the window:

Table 5-5. Stopwatch Icons

Icon	Icon Text	Description
	Properties	Set stopwatch properties. Select one current breakpoint or trigger to start the stopwatch and one to stop the stopwatch.
	Reset Stopwatch on Run	Reset the stopwatch time to zero at the start of a run.
	Clear History	Clear the stopwatch window.

.....continued		
Icon	**Icon Text**	**Description**
	Clear Stopwatch	(Simulator Only) Reset the stopwatch after you reset the device.

5.16 View the Disassembly Window

View disassembled code in this window. Select *Window>Debugging>Output>Disassembly Listing File* to open the window. You must Pause (halt) in a debug session (Debug>Debug Project) to see the window contents.

This information may also be found in the listing file produced by the linker. Open this file by selecting *File>Open File* and browsing for the `ProjectName.lst` file.

A quick way to view the entire disassembly file is to right click in the Disassembly window and select "Disassembly Listing File."

Note: The Disassembly window will disassemble each instruction, but has no history of banking associated with the instruction. Therefore, SFR names displayed in the window will be for Bank 0.

Figure 5-31. Disassembly Windows with Variable Mouseover

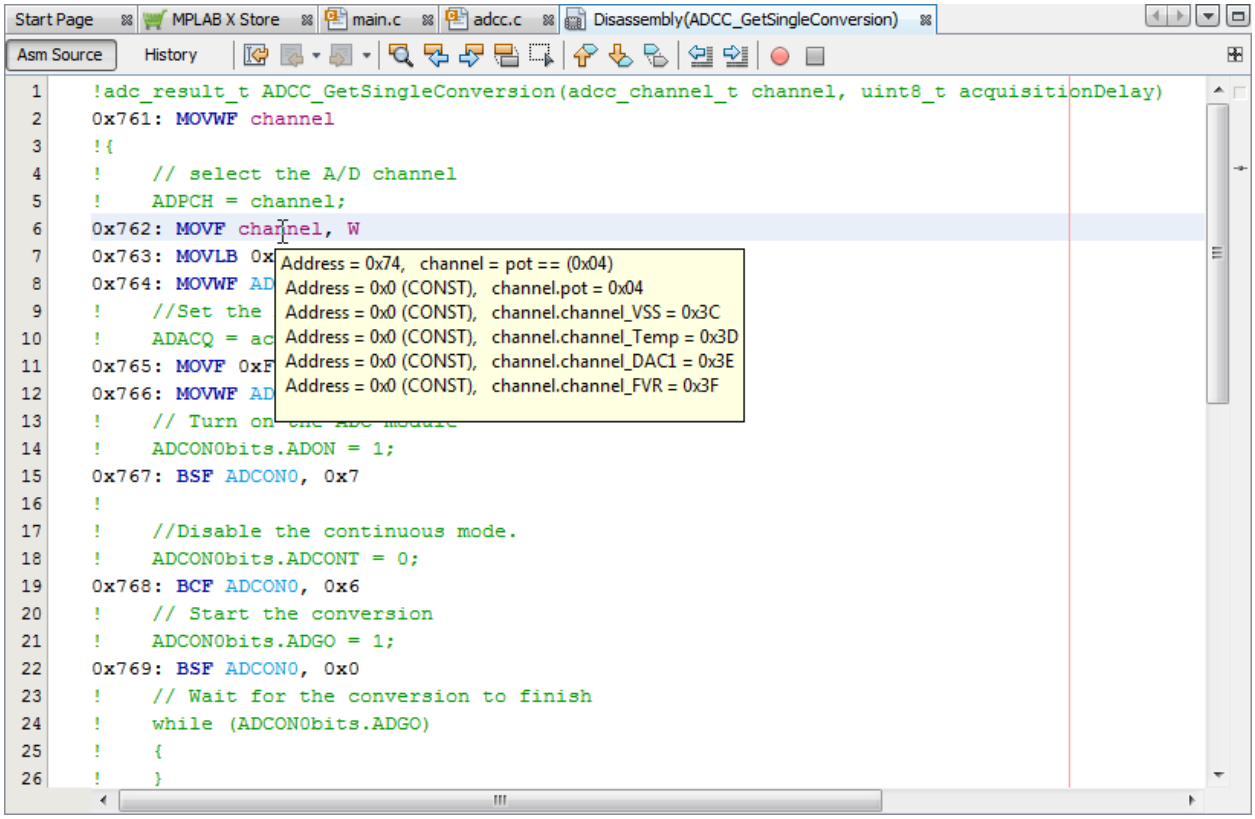


Table 5-6. Disassembly Window Context Menu

Menu Item(s)	Description
Step Instruction, Step Over, Run to Cursor	See 4.15 Step Through Code.
Cut, Copy, Paste	Cut, copy, or paste selected text in the window.
Select in Projects	Opens Projects window and selects the document containing selection.

.....continued	
Menu Item(s)	**Description**
Disassembly Listing File	See the contents of the disassembly listing file.

5.17 View the Call Stack

For 16- and 32-bit devices, a software Call Stack window is available to view CALLs and GOTOs in executing C code. This window is not applicable for assembly code (it is recommended that code optimization be turned off when using the call stack).

The Call Stack window displays functions and their arguments listed in the order in which they were called in the executing program.

To view the call stack:

1. Debug and then Pause your program.
2. Select *Window>Debugging>Call Stack*. A Call Stack window will open. See also 12.4 Call Stack Window.

For more on the call stack, see the NetBeans video:

https://www.youtube.com/watch?v=iS8OOUPmTUc

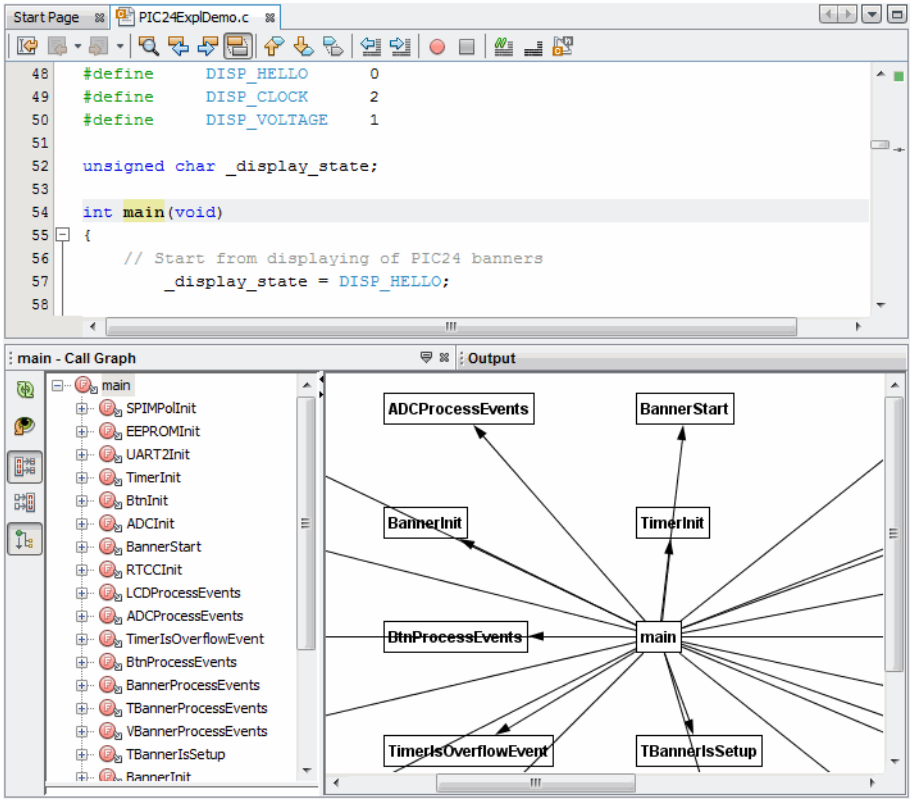
5.18 View the Call Graph

The Call Graph window displays a tree view of either the functions called from a selected function, or the functions that call that function.

To view the call graph:

- Right click on a function and select "Show Call Graph" from the drop-down menu.

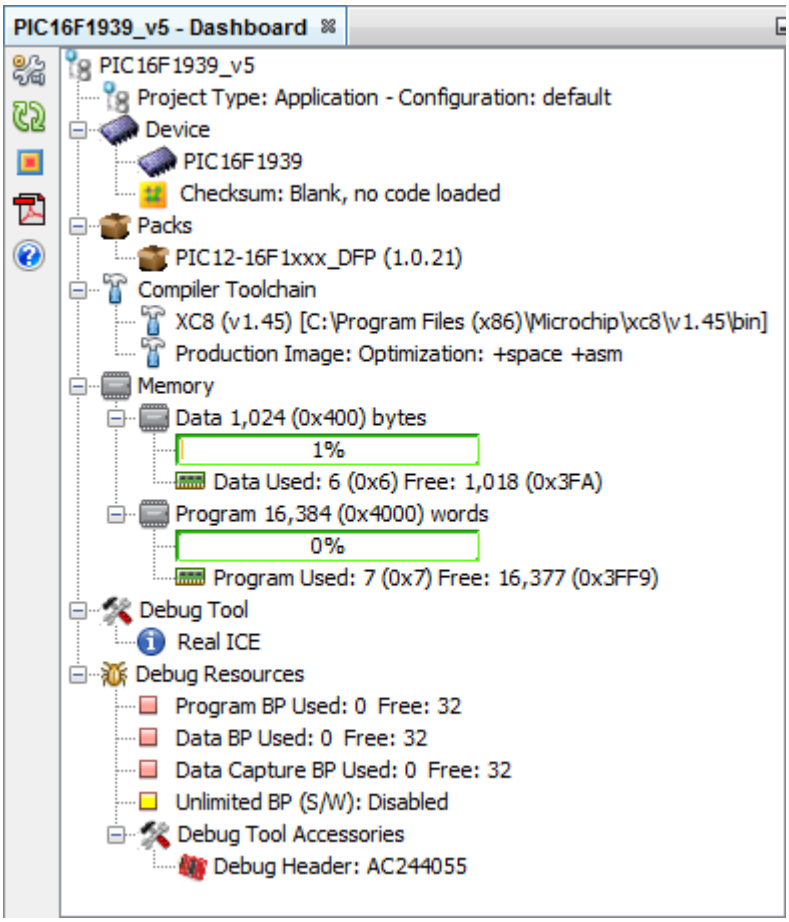
Figure 5-32. Call Graph of Main Function



5.19 View the Dashboard Display

Select *Window>Dashboard* to display project information.

Figure 5-33. Dashboard Display



The project displayed will either be:

- The active project in the Projects window, if no main project is selected. Click on a project in the Projects window to make it active.
- The main project. Right click a project in the Projects window and select "Set as Main Project."
- No other project, active or inactive, will be displayed.

5.19.1 Dashboard Groups

Dashboard window project information is grouped into specific sections.

Table 5-7. Dashboard Groups (Continued)

Group	Definition and Content
Project Name	• name of the **active/main project** • **project type** (application or library) and the **project configuration** (default or user-defined)
Device	• project device • any **status flags** generated during Run or Debug • **checksum** of what is loaded into device memory. You must build to see this. Refer to 6.14 Checksums.

.....continued

Group	Definition and Content
 Packs	Packs containing versioned device information -`.PIC` file and other files, depending on device.
 Compiler Toolchain	• **project toolchain name** (e.g., XC8), the tool **version number** in ( ) and the **path** to the executable files in [ ]. • **image type** (Production or Debug) and toolchain **optimization** level, which corresponds to license type. For more on optimizations, see your language tool documentation. • information on the **compiler license***: – license type: Workstation or Network – license mode: FREE, STD or PRO – license seats: total number of seats available – licenses available: the number of licenses available – HPA – days until HPA ends – Expire – days until the license expires – Press for status – See status in Output window * You must have at least MPLAB XC8 v1.34, MPLAB XC16 v1.25, or MPLAB XC23 v1.40 for this to be visible. For more on compiler licenses, see: https://www.microchip.com/mplab/compilers
 Memory	• usage symbols disabled. Click on message to enable Load Symbols. In the Project Properties window, Loading category, check "Load symbols when programming or building for production (slows process)". • **type of memory** (Data or Project) and the **amount available** on the device. • **Memory Used** should be a guide to the amount of memory remaining. The compilers output a Memory Summary that details usage for Program Space, Configuration Bits, ID location, and EEPROM (if on the device). The sum of these memory spaces, allowing for word sizes, should agree with the Dashboard Program Used. The Map file should be examined when your device has smaller amounts of memory.
 Debug Tool	**Hardware Debug Tools** • debug tool **name** and serial number • debug tool **connection**. By default, the tool connection is active at all times. To change this, go to *Tools>Options* (*MPLAB X IDE>Preferences* for macOS), **Embedded** button, **Generic Settings** tab, "Maintain active connection to hardware tool". • click the **Refresh Debug Tool Status** button to see hardware debug tool firmware versions and current voltage levels. **Simulator** • debug tool **name**, i.e., Simulator. • **Click for Simulated Peripherals** to see a list of supported device peripherals.

.....continued	
Group	**Definition and Content**
Debug Resources	**Hardware Breakpoints** The number of breakpoints currently being **used** and the number of breakpoints **free** for the project device. **Software Breakpoints** Whether software breakpoints **supported** on the project device. **Debug Tool Accessories** Which, if any, debug tool accessories are used in this project, such as debug headers.

5.19.2 Dashboard Icons

Icons on the left side of the Dashboard window provide access to functions.

Table 5-8. Sidebar Icons

Icon	Function
	Project Properties Displays the Project Properties window.
	Refresh Debug Tool Status Click this to see hardware debug tool details.
	Toggle Software Breakpoint – Enabled/Disabled Click to alternately enable or disable software breakpoints. The state of this feature is shown in the icon: • Red center: Disabled • Green center: Enabled • Black center: Not supported – when you have exceeded the number of break points available for your device or tool.
	Open Device Data sheets Get a device data sheet from the Microchip web site: https://www.microchip.com/ Click to either open a saved, local data sheet or open a browser to go to the Microchip web site to search for a data sheet.
	Compiler Help Click to open the Master Index (if available) for documents in the docs folder of the compiler installation directory.

5.20 View Registers for the Project (I/O View)

Use the I/O View window (*Window>Debugging>IO View*) to see an overview of registers of the target device for the current project. It serves as a quick reference during design and is capable of displaying register values when the project is in debug mode.

The I/O View is a split pane view with peripherals in top pane and registers of the selected peripheral in bottom pane.

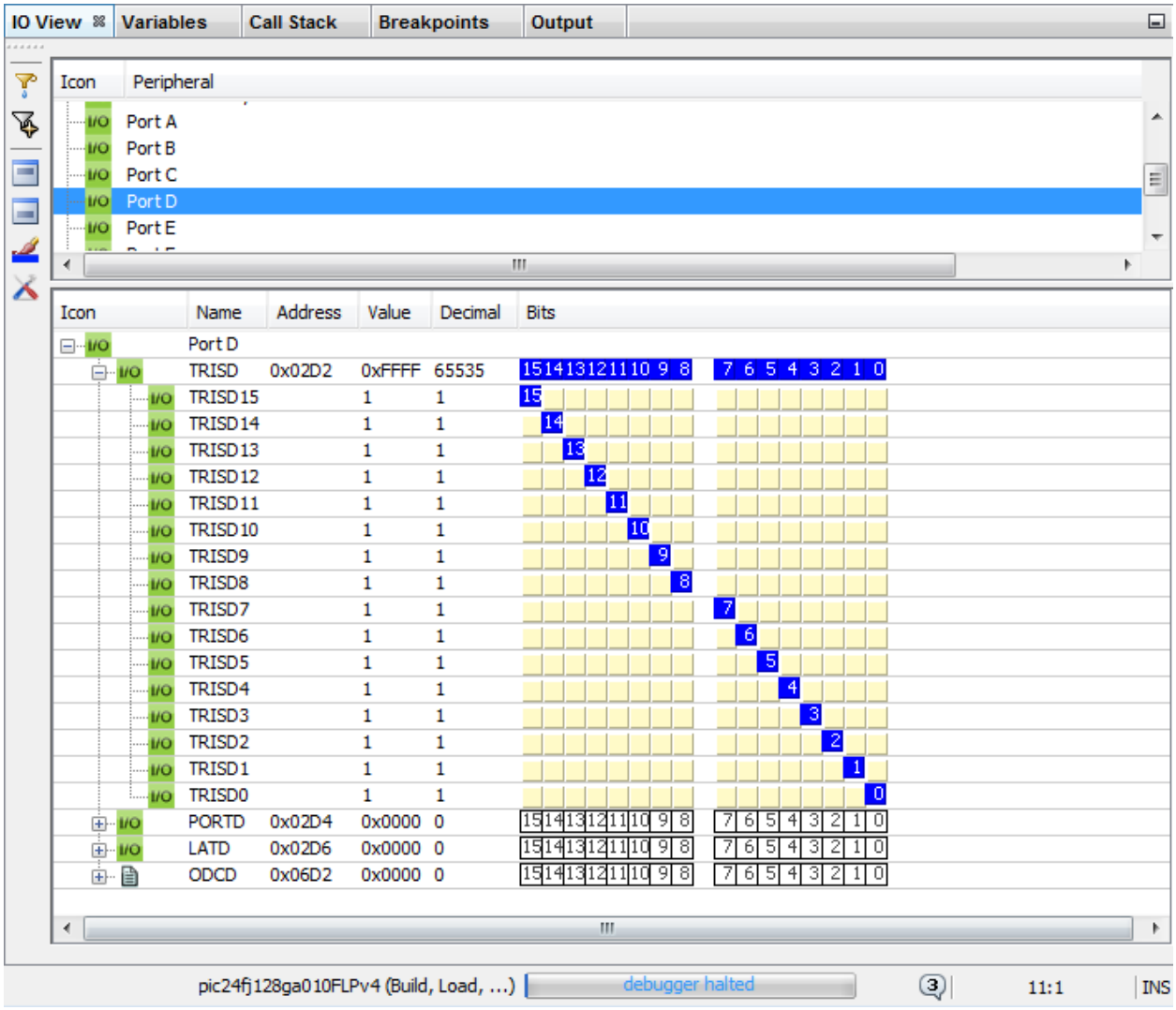
- The top pane can show registers with named editing values in combo, like Config combo.
- The bottom pane shows register words with custom bit control for editing and changed colored bits.

For more on this window, see 12.8 I/O View Window.

Edit During Debug Pause

When debug mode has been paused, you can edit values in the window.

Figure 5-34. I/O View for Debug Pause



To change a bit, simply click it, and the value will be toggled. Each bit in the register is displayed in a separate column. Bits which are set will have a dark color by default, while cleared bits will have no color (default white).

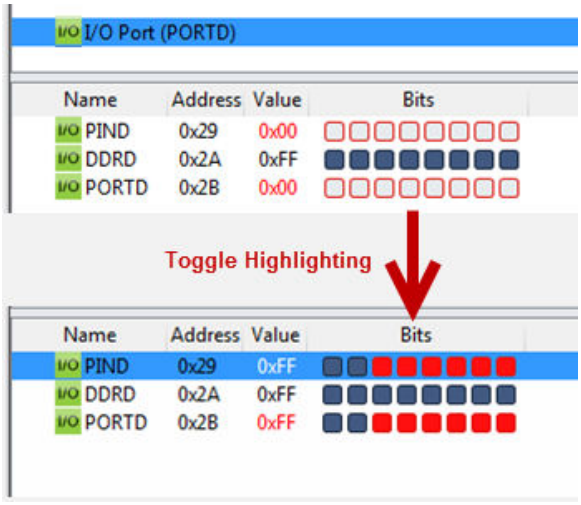
Any value can be changed by clicking the value field and writing a new value.

Some values and bits cannot be modified as they are read-only and some bits may be write-only. See the documentation for each device for more information. When a bit or value is set, it is immediately read back from the device, ensuring that the I/O View only displays actual values from the device. If a new value is set, but the I/O view does not update as expected, the register might be write-only or simply not accessible.

Change Colors

When a register has changed since last time it was displayed, it will indicate so with a red colored value and bits in the display. If a bit has been set since last time, it will be solid red. If it has been cleared it will simply have a red border. This feature can be toggled on or off in the toolbar.

Figure 5-35. I/O View - Toggle Highlighting of Modified Bits



5.21 Improve Your Code

Improve your code by using code refactoring and/or profiling.

Refactoring code is a method of making code simpler without changing its functionality. Currently you can do the following with C code:

- Find function usages throughout files
- Rename functions and parameters throughout files

For more information, see 7.6 C Code Refactoring.

Profiling code is an examination of CPU Usage, Memory Usage, and Thread Usage tools, all observed while the program is running. The profiling tools run automatically whenever you run your C project.

If you are looking for features found in NetBeans IDE menus (but not normally used or visible in MPLAB X IDE), go to *Tools>Plugins>Installed>MPLAB Hidden Menu Filter*, select this item, and click "Deactivate."

5.22 Control Source Code using Local History

MPLAB X IDE has a built-in local file history feature, a benefit of the NetBeans platform. This feature provides built-in versioning support for local projects and files that is similar to conventional version control systems. Available tools include a local DIFF and file restoration.

To see local history for a file, either:

- Click on the **History** button on an Editor window
- right click on the file in the Projects or Files window and select *History>Show History*

Any past changes to the file should be listed there.

Figure 5-36. Local History in Editor Window

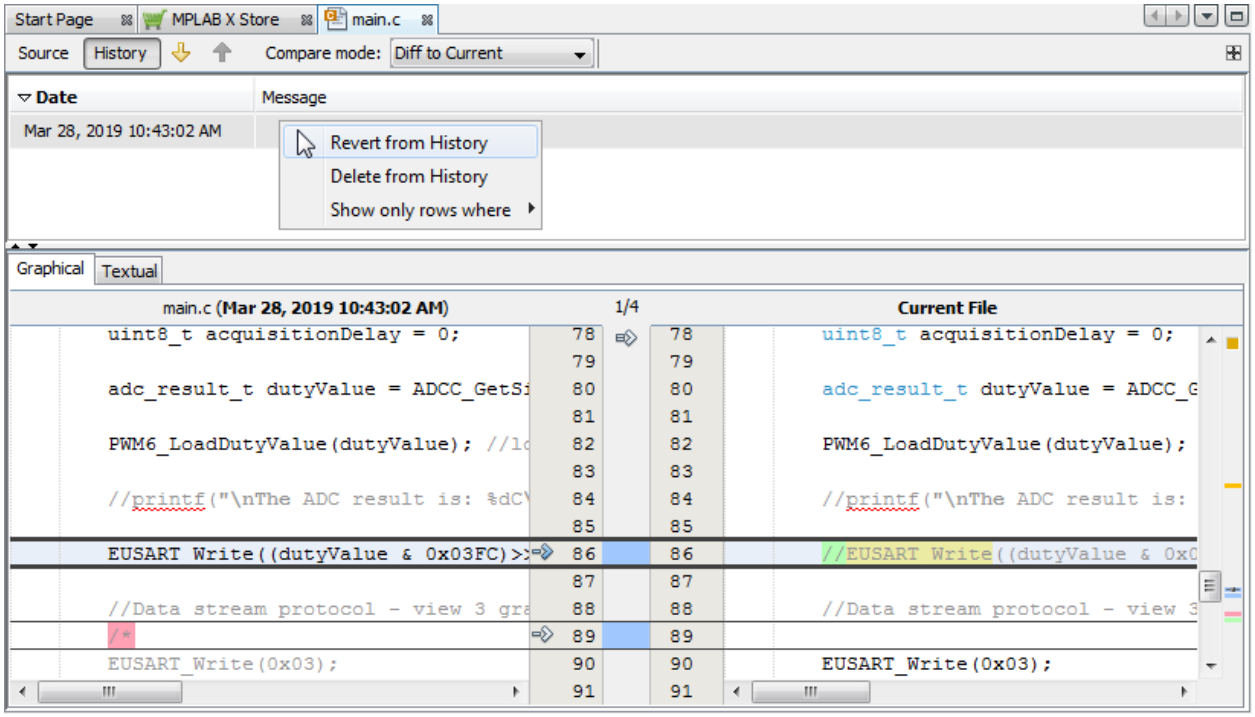


Table 5-9. Local History Context Menu

Menu Item	Description
Revert from History	To revert to a previous version. The Revert to dialog opens with any previous versions of the document. Select one and click **OK** to revert to that version.
Delete from History	To delete changed from history. To revert items deleted from history, right click on the file in the Projects or Files window and select *History>Revert Deleted*.
Show only rows where	Right click under the Date column to select date filtering (Date == or Date < >). Right click under the Message column to select message filtering (Message == or Message < >).

5.23 Control Source Code using a Revision Control System

When developing a complex application that consists of many files, especially as part of a team, controlling source code changes becomes a necessity. MPLAB X IDE provides a built-in method of source file versioning and supports external revision (source or version) control programs.

Several revision control systems (RCSs) may be used with MPLAB X IDE. Once set up, revision control actions can be accessed through a context menu.

Currently supported source/version control systems are Git, Subversion and Mercurial. Other systems may be supported through plugins (e.g., CVS).

When using a revision control system for your MPLAB X IDE projects:

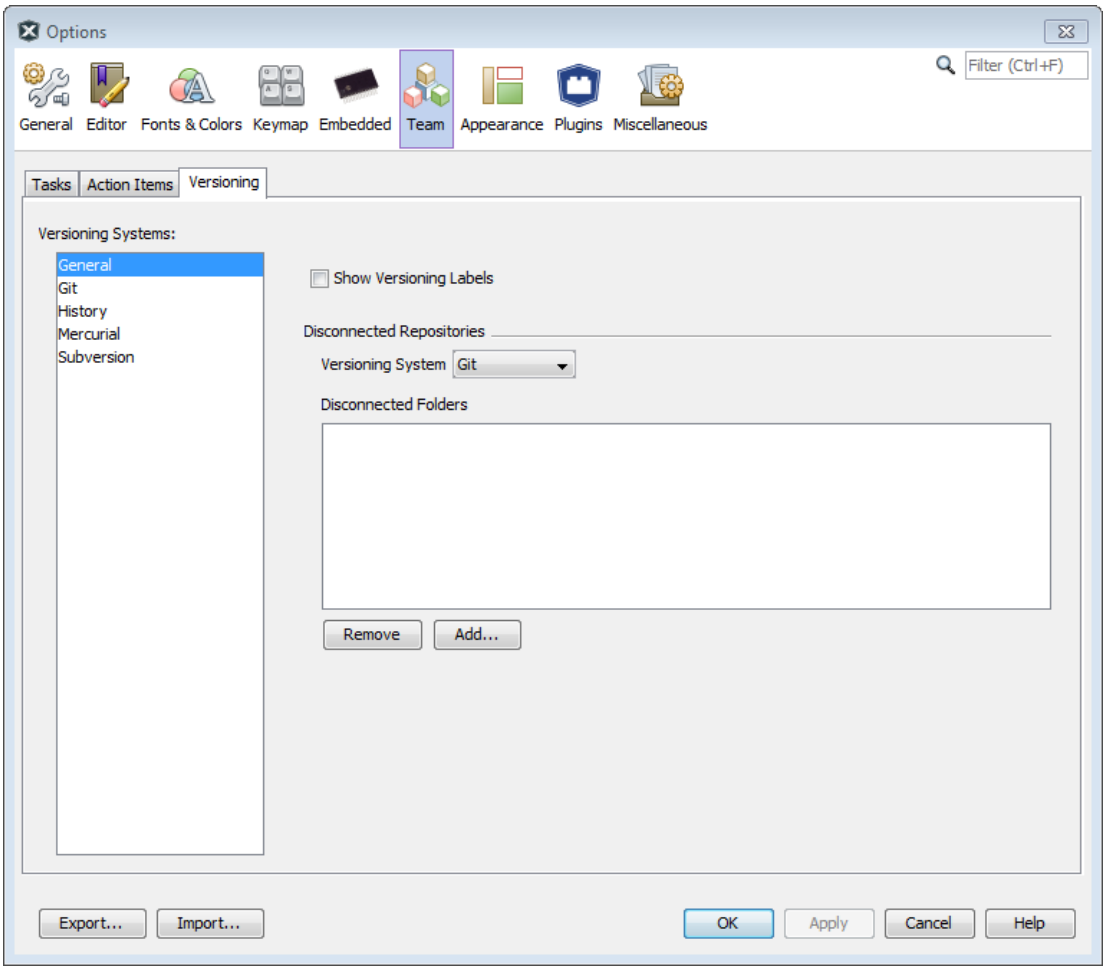
1. Try to keep all project files within the project. Files outside the project will also need to be included in the repository so the project can find them.
2. In MPLAB X IDE, commit only the files from the project. This is important as inclusion of unwanted files can prevent the project from compiling on other machines. See "Saving Project Files."

3. Resolved any conflicts when checking in using the Merge Conflicts Resolver window. See "Resolving Conflicts in Revision Controlled Files."
4. Check read-only files in per "Building a Project with Read-Only Files."

To set up source/version control:

1. Team menu – select a version control program from the submenus and set up that version control program (see the figure below).
2. *Tools>Options* (*MPLAB X IDE>Preferences* for macOS), Team, Versioning – set up version control options.
3. *Window>Versioning* – open version control windows.

Figure 5-37. Version Control Options



Working with Git

Follow these links for more on Git:

- Git Website: http://git-scm.com
- Wikipedia: Git: https://en.wikipedia.org/wiki/Git
- Git Tutorial: https://git-scm.com/docs/gittutorial
- Using Git Support in NetBeans IDE: https://netbeans.org/kb/docs/ide/git.html

Working with Subversion

Follow these links for more on Subversion:

- Apache™ Subversion® Website: http://subversion.apache.org/
- Wikipedia: Apache™ Subversion®: http://en.wikipedia.org/wiki/Apache_Subversion
- Version Control with Subversion®: http://svnbook.red-bean.com/

- Using Subversion Support in NetBeans IDE: https://netbeans.org/kb/docs/ide/subversion.html

Working with Mercurial

Follow these links for more on Mercurial:

- Mercurial Website: http://mercurial.selenic.com/
- Wikipedia: Mercurial: http://en.wikipedia.org/wiki/Mercurial
- Hginit: Mercurial tutorial: http://hginit.com/
- Using Mercurial Support in NetBeans IDE: https://netbeans.org/kb/docs/ide/mercurial.html

Working with CVS

To use CVS with MPALB X IDE, you must install the CVS plugin. See 5.26 Add Plugin Tools.

- CVS - Concurrent Versions System: http://www.nongnu.org/cvs/
- Wikipedia: Concurrent Versions System: http://en.wikipedia.org/wiki/Concurrent_Versions_System
- Source Forge: What is CVS?: http://sourceforge.net/apps/trac/sourceforge/wiki/CVS
- Using CVS Support in NetBeans IDE: https://netbeans.org/kb/docs/ide/cvs.html

Working with Other Revision Control Systems

Other revision control systems may be added as plugins. See 5.26 Add Plugin Tools.

5.23.1 Saving Project Files

There are project files that need to be saved into a repository.

The following table lists project files that either need or do not need to be committed to a version control repository.

Table 5-10. Project Files Saved to Repository

Directory or File(s)	Commit?
Project directory	
`Makefile`	✔
Source files	✔
build directory	✘
dist directory	✘
nbproject directory	
`configurations.xml`	✔
`project.properties`	✔
`project.xml`	✔
`Makefile-*`	✘
`Package-*`	✘
private directory	✘

.....continued	
Directory or File(s)	**Commit?**
Green Check: required to generate the project image Red X: these directories/files are regenerated and therefore do not need to be saved	

See "Files Window View" for more on the project structure.

5.23.2 Resolving Conflicts in Revision Controlled Files

Inevitably, conflicts will arise between a file you are trying to check in and the same file in the revision system repository that others have modified.

To understand how these issues can be resolved, see the example below.

Example: Conflict in `configurations.xml`

To resolve conflicts on the file `configurations.xml`:

1. Conflicts will be announced in:
   1.1. warning dialogs
   1.2. the output window
2. Click on the **Files** tab in the Projects window. Expand the `nbproject` folder. Find the file named `configurations.xml`. It should be in red font, signifying conflicts.
3. Right click on configurations.xml and select *RCS>Resolve Conflicts*, where *RCS* is your revision control system. This opens the Merge Conflicts Resolver window.
4. The Merge Conflicts Resolver window allows you to fix conflicts.
5. After fixing the errors, close the project. If the right click context menu for the project does not work, close the project using *File>Close Project*.
6. Reopen the project. The conflicts should be resolved.

5.23.3 Building a Project with Read-Only Files

To build a project where files have been checked into a repository and some may then be read only:

1. Commit the minimum number of files to source control. This is explained in 5.23.1 Saving Project Files.
2. Make the files `nbproject/configurations.xml` and `nbproject/project.xml` writeable.

5.24 Collaborate on Code Development and Error Tracking

Collaborate on code development with your group using a team server supported inside MPLAB X IDE. Use the "Team" menu to log into your account, create or open your project, share your project, get resources, send a chat message or show your contact list. Collaborate on tracking bugs by using issue tracking systems

Supporting menu items include:

- *Window>Services* – The Services window is the main entry point to your runtime resources. It shows a logical view of important runtime resources such as the servers, databases, and web services that are registered with the IDE. Search for "Services Window" under *Help>Help Contents* for more details.
- *Team>Find Task* – In the "Find Tasks" window, select a project task, select criteria, and click **Search**.
- *Team>Report Task* – in the "Report a New Task" window, select the new project task, specify the issue details, and click **Submit**.

For more on team projects and issue tracking, see the NetBeans Help topic:

https://netbeans.org/kb/docs/ide/team-servers.html

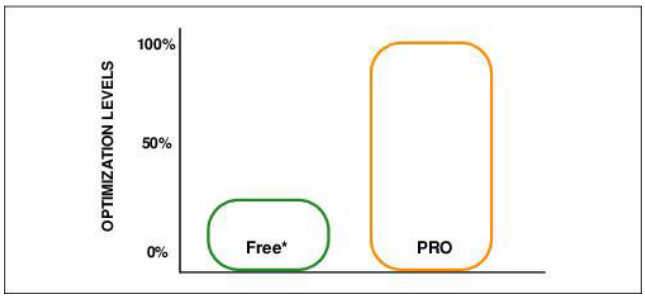
To find out more about these tools, see the following:

- Git – http://git-scm.com
- Mercurial – http://mercurial.selenic.com/
- Subversion – http://subversion.apache.org/

5.25 Compare MPLAB XC Compiler Free vs. PRO Licenses

When you download an MPLAB XC C compiler from the Microchip website, the compiler executes as a PRO license for 60 days and then reverts to a Free license. You can purchase the PRO license if you want to continue to use the most optimizations.

Figure 5-38. Optimization Levels per License

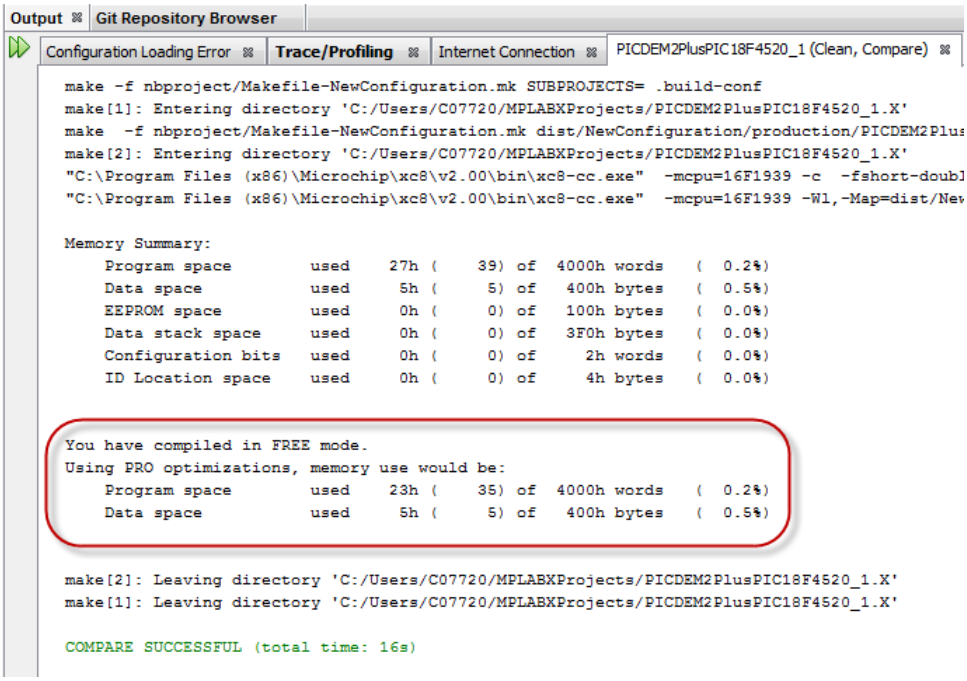


To determine if you need to buy a PRO license, you can run a comparison tool for your application in MPLAB X IDE. On the Run toolbar, you can select to either "Build with PRO comparison" or "Clean and Build with PRO comparison." An example output is below.

Note: If your code it too large to build in Free mode, the comparison will not work. However, for the MPLAB XC8 compiler, you can use the --MAXIPIC option and try again.

These options build the project normally, in whatever optimization is selected, but also create a non-working PRO build and then compare the two results. You still get the original build as the output. Any parts of the PRO build are removed.

Figure 5-39. PRO Comparison



5.26 Add Plugin Tools

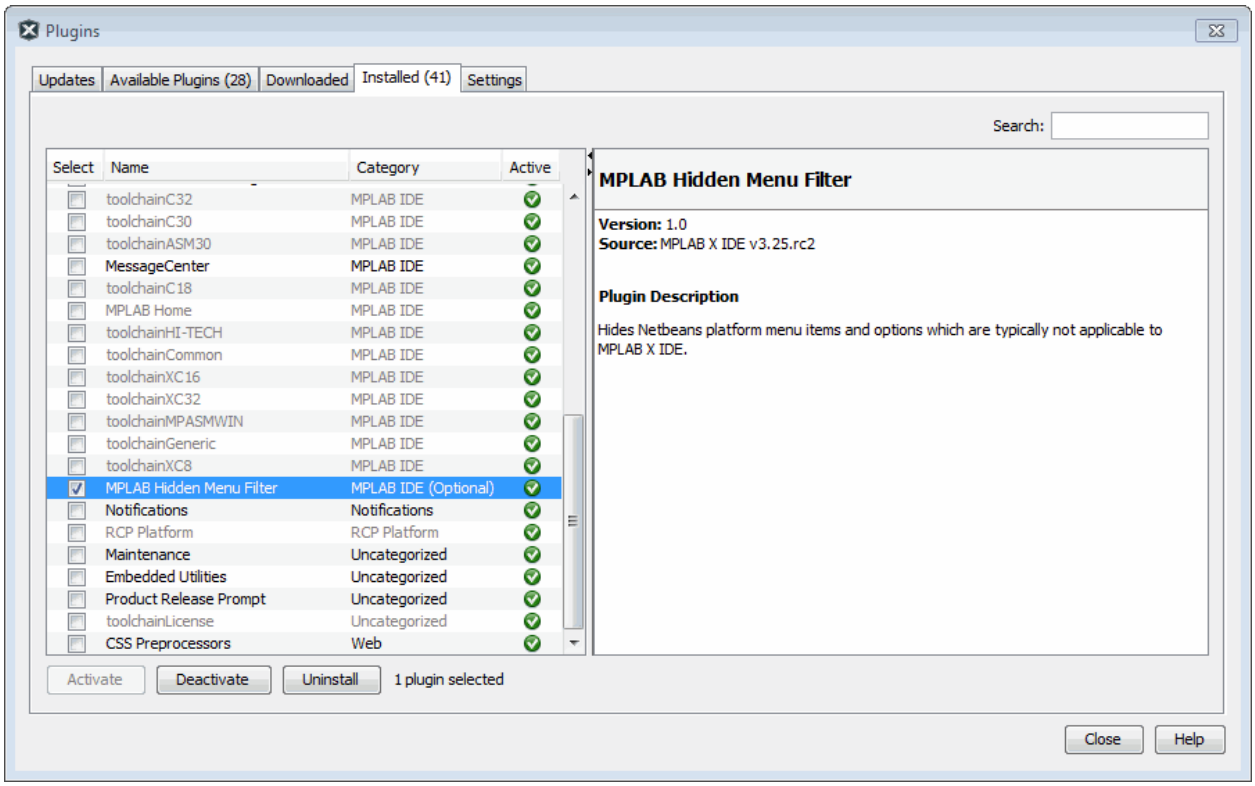
MPLAB X IDE plugin tools, like DMCI, are available from either the Plugin Manager in the IDE or the Embedded Code Source web site.

Viewing Installed Plugins

MPLAB X IDE comes with a number of plugins installed. Although ungrayed items may be deactivated or uninstalled, this is not recommended unless the category is listed as optional.

To view installed plugins, select *Tools>Plugins* and click on the **Installed** tab.

Figure 5-40. Installed Microchip Plugin Tools



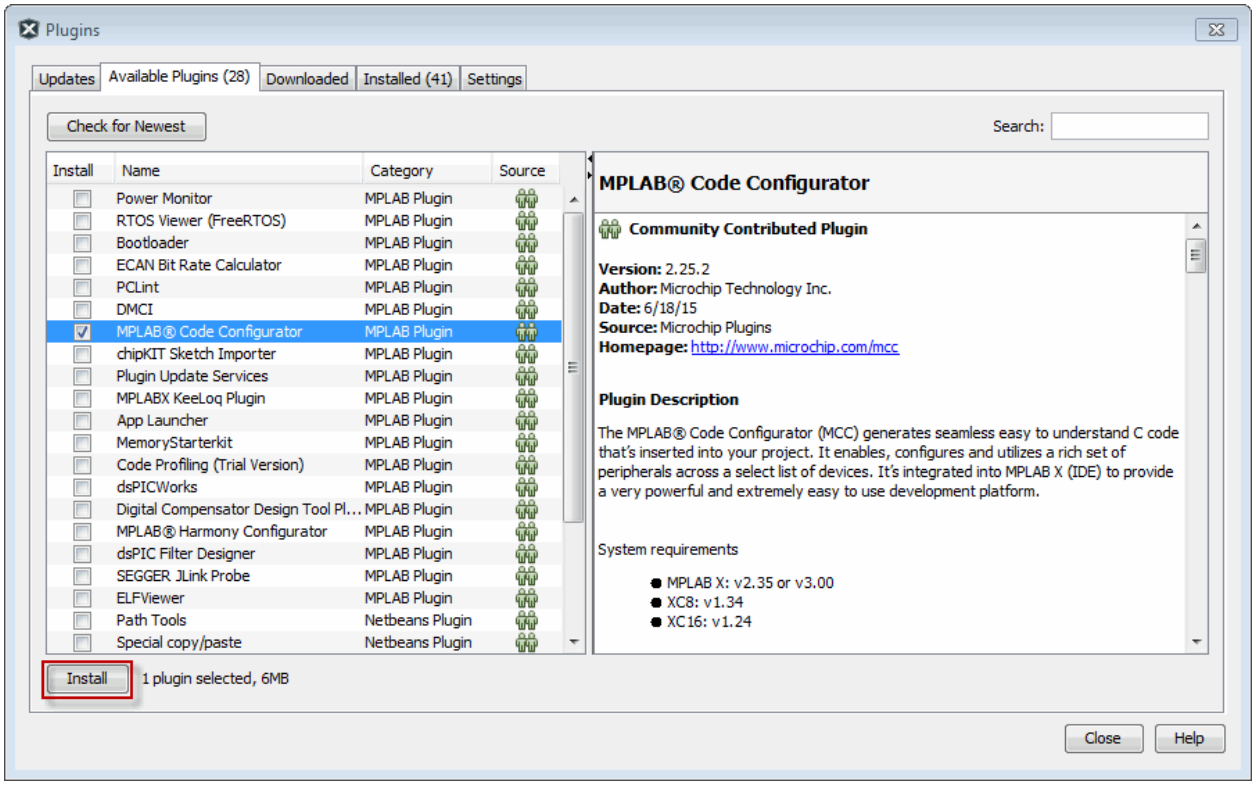
Add Plugins from the Plugin Manager

To view and install available plugins:

1. In MPLAB X IDE, select *Tools>Plugins* and click on the **Available Plugins** tab.
2. Select your plugin by checking its check box and then click Install (see figure below).
3. Follow the on-screen instructions to download and install your plugin.
   Note: Some plugins may be dependent on the modules in other plugins for their functionality to be implemented. The Plugins Manager warns you when this is the case.
4. Once installed, the plugin will be listed under the Installed tab, where it can be deactivated, reactivated or uninstalled.
5. Look for your tool under Tools>Embedded. If you do not see it, you may need to close and re-open MPLAB X IDE.

Click the **Help** button to read more about installing plugins.

Figure 5-41. Available Mlicrochip Plugin Tools

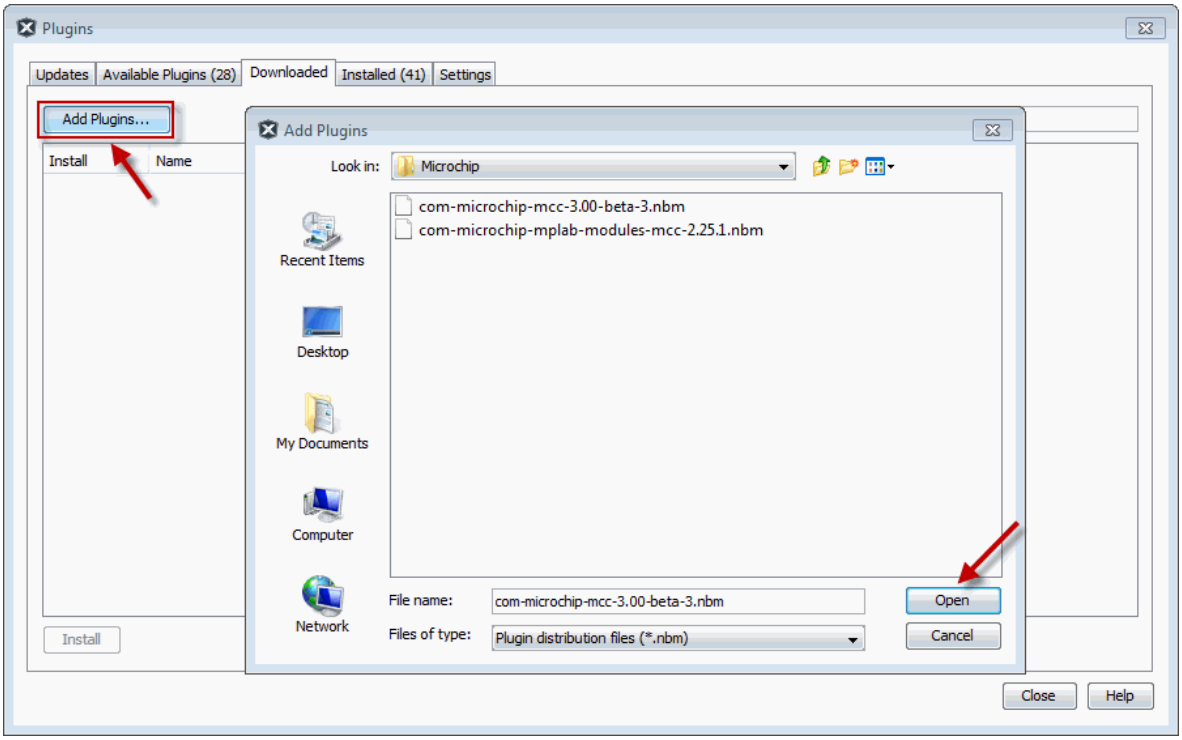


Add Plugins from Embedded Code Source

To view, download and install available plugins:

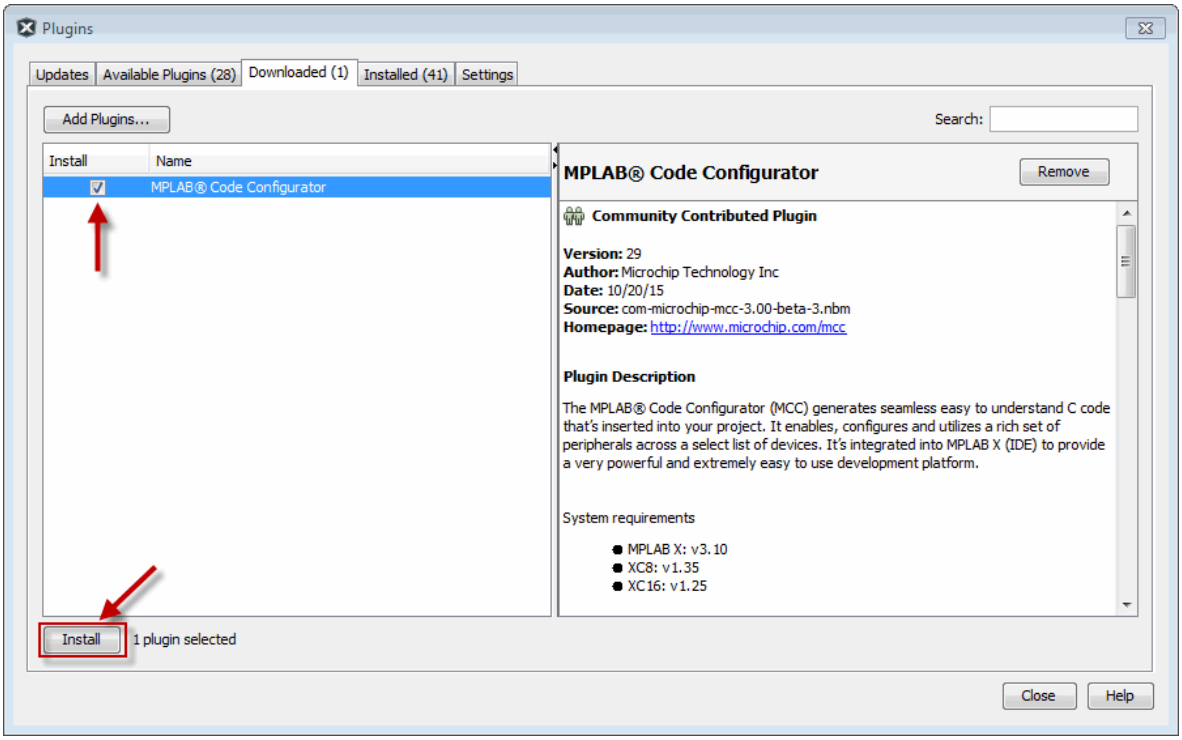
1. Go to the Embedded Code Source web site: http://www.embeddedcodesource.com/codedeveloper/microchip-technology
2. Select a plugin and follow the instructions on the web site to download to a folder on your computer.
3. Extract the `.nbm` file from the downloaded ZIP file.
4. In MPLAB X IDE, select *Tools>Plugins* and click on the **Downloaded** tab.
5. Click **Add Plugins**. Find, select, and open the `.nbm` file.

Figure 5-42. Downloaded Microchip Plugin Tools



6. Click Install to install the plugin. Ensure that the check box next to the plugin name is checked.

Figure 5-43. Install Downloaded Microchip Plugin Tools



7. Follow the on-screen instructions to download and install your plugin.
   Note: Some plugins may be dependent on modules in other plugins in order for the functionality to be implemented. The Plugins Manager warns you when this is the case.

8. Once installed, the plugin will be listed under the Installed tab, where it can be deactivated, reactivated or uninstalled.
9. Look for your tool under *Tools>Embedded*. If you do not see it, you may need to close and re-open MPLAB X IDE.

Upgrade Plugins

To determine if there is an update for your plugin:

1. In MPLAB X IDE, select *Tools>Plugins* and click on the Updates tab.
2. Click **Check for Updates**.
3. If any updates appear, ensure that the check box next to the plugin name is checked and then click **Update**.
4. Follow the on-screen instructions to update your plugin.

Installing a new version of MPLAB X IDE will NOT update your installed plugins. Plugins are tested against a versioned interface of a release. Not all plugins can be migrated between versions so they are not carried over. This is true for NetBeans, also. So, you may need to reinstall your plugins when you install a new version of MPLAB X IDE.

Configure Update Centers

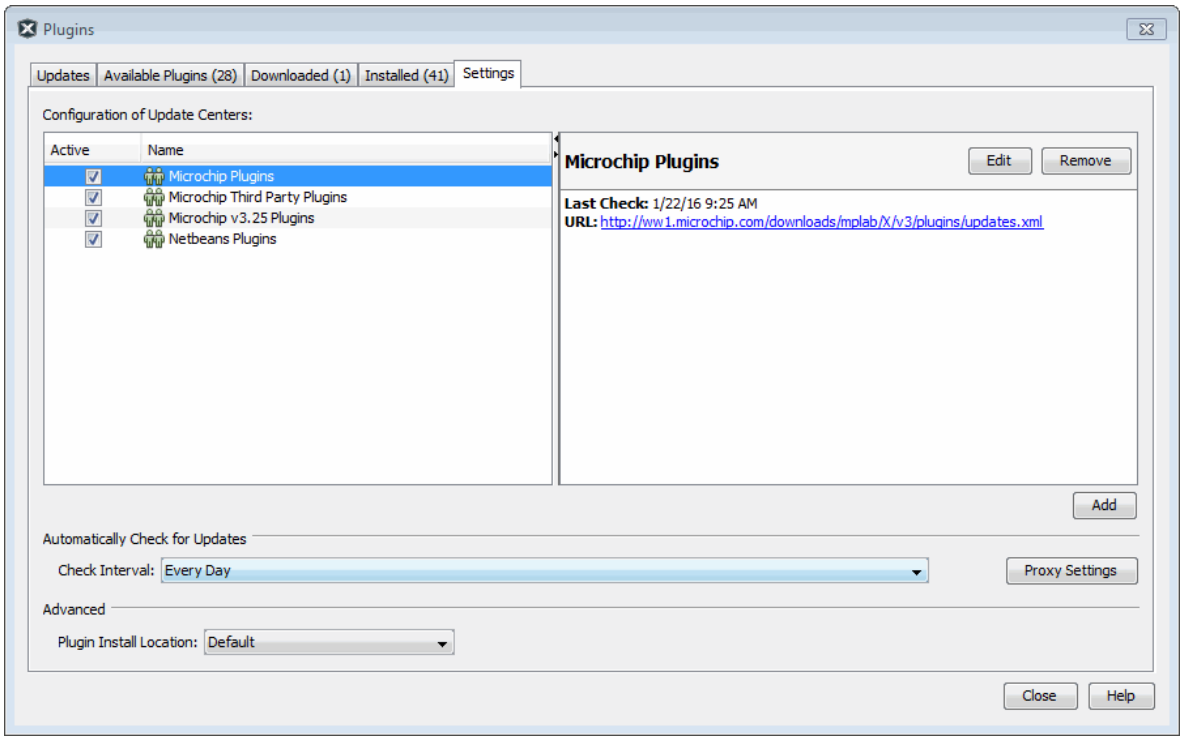
To see available plugins in the Plugins window, you must have one or more update centers configured. MPLAB X IDE comes with two Microchip Update Centers already configured:

- lhttp://ww1.microchip.com/downloads/mplab/X/plugins/updates.xml
- http://ww1.microchip.com/downloads/mplab/X/thirdpartyplugins/updates.xml

To configure another Update Center in the Plugin Manager:

1. Select *Tools>Plugins* and click the Settings tab.
2. Click the Add button to open the Update Center Customizer dialog.
3. Enter a name for the update center.
4. Enter a URL for the update center.
5. Click **OK**.

Figure 5-44. Configure Microchip Update Center



Configure Installed Plugin

Once a plugin is installed in MPLAB X IDE, you may need to configure it. Plugin configuration options are available under *Tools>Options*, Plugins. See "Tools Options Plugins Window."

Plugin Code Location

Plugin code is stored with MPLAB X IDE user configuration data. See 8.7 Viewing User Configuration Data.

6. Advanced Tasks & Concepts

The following topics discuss how to perform more advanced tasks and detail some useful concepts.

Table 6-1. Understanding Advanced Tasks and Concepts

Speed Things Up	1. Speed Up MPLAB X IDE if the IDE is operating too slowly. 2. Speed Up Build Times in MPLAB X IDE by using parallel make.
Work with Projects	1. Work with Multiple Projects when developing a complex application. 2. Work with Multiple Configurations to allow selection of different project properties for use with the same project code. 3. Create Dual Core Projects to work with dsPIC33CH dual core devices. 4. Create User Makefile Projects to use external makefiles in MPLAB X IDE. 5. Used Linked Resources for Source File Folders to change project source bases or library versions (MPLAB Harmony). 6. Work with Third Party Tools in an MPLAB X IDE project. 7. Use Code Coverage to test the percentage of code executed. 8. Log Data to capture execution and debug problems. 9. Package an MPLAB X IDE Project to zip up the files in a project. 10. Hardware tool connections to the target can sometimes determine debug capabilities. See Hardware Tool Connections and Debugging.
Concepts	1. What are Checksums. 2. Different kinds of Configurations.

6.1 Speed Up MPLAB X IDE

If MPLAB X IDE is operating too slowly, consider the suggestions in the follow sections.

Increase the Computer Heap

You can modify the amount of memory allocated to MPLAB X IDE in the file `mplab_ide.conf`. We recommend you back up this file before you start editing it. If you change the contents of this file, the changes will be operative the next time you run MPLAB X IDE.

- ***Windows OS 64 Bit*** - `C:\Program Files (x86)\Microchip\MPLABX\vx.xx\mplab_platform\etc`
- ***Windows OS 32 Bit*** - `C:\Program Files\Microchip\MPLABX\vx.xx\mplab_platform\etc`
- ***Linux OS*** - `/opt/microchip/mplabx/vx.xx/mplab_platform/etc`
- ***macOS*** - `/Applications/microchip/mplabx/vx.xx/Contents/Resources/mplab_platform/etc`

where `vx.xx` is the MPLAB X IDE version.

The following line contains the default values:

```
default_options="-J-Dnb.FileChooser.useShellFolders=false -J-Dcrownking.stream.verbosity=very-
quiet -J-Xms256m -J-Xmx512m -J-XX:PermSize=128m -J-XX:MaxPermSize=384m -J-XX:
+UseConcMarkSweepGC -J-XX:+CMSClassUnloadingEnabled"
```

The bolded areas are:

`-Xms256m` tells the JVM to start with at least 256 MB for the heap.

`-Xmx512m` tells the JVM to allocate as much as 512 MB for the heap, but no more.

-XX:PermSize=128m tells the JVM to allocates 128 MB for space needed to keep track of additional data that does not go on the heap.

-XX:MaxPermSize=384m tells the JVM to allocate no more than 384 MB for the additional data that does not go on the heap.

You should not need to modify the PermSize or the MaxPermSize unless you get the error `java.lang.OutOfMemoryError: PermGen space`.

In general, the most important area is `-Xmx512m`; this limits the maximum amount of heap that MPLAB X IDE will use. It might seem that having a large amount of heap could be helpful. However, memory used for MPLAB X IDE means less memory for other applications and system functions.

You can monitor how much memory the IDE is using by enabling the Memory monitor. Either right click on an empty space in the toolbar area and select memory, or select *View>Toolbars>Memory*.

The upper number will not go above 512 MB unless you change the value in `mplab_ide.conf`. It is recommended that you change the `-Xmx512m` setting in 128 MB increments. If you have a lot of memory, you can increase it by more than 128 MB, but make sure you leave enough memory for the rest of the system.

Change Debug Window Usage

If using debug tools slows down MPLAB X IDE, try the following:

- Only display windows you need to view. The MPLAB X IDE debugger will examine each open window when a debug tool is used.
- Reduce updates in the Stack window during debug stepping. In the *Tools>Options* (*MPLAB X IDE>Preferences* in macOS) window, **Embedded** button, **Generic Settings** tab, check "Disable auto refresh for call stack view during debug sessions" and "On mouse over structure and array expressions, evaluate integral members only."

6.2 Speed Up Build Times

Depending on the configuration of your computer, you may be able to use parallel make (see 12.16.2 Project Options Tab) to speed up your project build times. Not all language tools support parallel make.

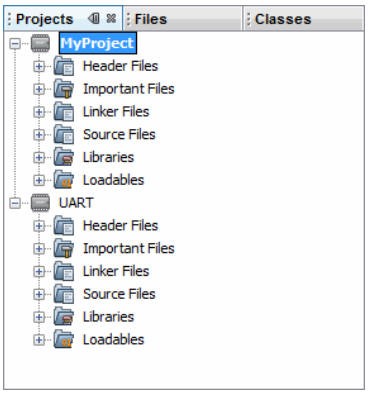
Another option is to consider your operating system (OS). Some OS's have faster file accessing. MPLAB X IDE supports Windows, Linux and Mac OS's. Research which one might be right for you.

6.3 Work with Multiple Projects

MPLAB X IDE allows you to work with more than one project.

Multiple projects can be opened in MPLAB X IDE and viewed in the Projects window. For more about this window, see 12.15 Projects Window.

Figure 6-1. Multiple Projects in the Projects Window



Open Multiple Projects

You can create (*File>New Project*) or open (*File>Open Project*) more than one project in MPLAB X IDE directly.

Alternatively, you can use the `--open` directive in a short cut, as in:

```
"C:\Program Files (x86)\Microchip\MPLABX\v3.15\mplab_platform\bin\
mplab_ide.exe" --open "C:\MPLAB_X_Projects\MyProject1"
```

Then you can create multiple short cuts to open multiple projects from many different locations.

Define Project Type

Each project can be made active, or the focus of development, as described below. However there should only be one main project at any time.

Main Project

A single project can be selected as the main project by right clicking the project name and selecting "Set as Main Project." The main project name will then appear in bold. In this case all the toolbar functions will be executed on the one main project.

Active Projects

You can also work without setting a main project. Active projects are the projects in focus when there is no main project.

Projects may be made active by clicking on them in the Projects window. In that case, the IDE will know which project you are working on by context, i.e., if you set the editor focus on a file, then the project that owns that file will become the active project. The active project receives all actions, e.g., when you click the Debug Project icon.

Develop Project Code

To develop and debug your code for each project:

1. Start debugging a project by clicking on the project in the Projects window and then selecting *Debug>Debug Project*.
2. If you are running multiple simultaneous debug sessions, open the Sessions window (*Window>Debugging>Sessions*) and you can switch between any currently running debug session.

When one debug session is switched to another, the Watches window and variables plus the memory will switch to show the currently selected project being debugged. The Status bits should also follow the debug project. The Dashboard will follow the last selected whether it is a debug project or a project in the project window. This is by design.

If you have the option "Maintain active connection to hardware tool" selected under *Tools>Options* (*MPLAB X IDE>Preferences* for macOS), **Embedded** button, **Generic Settings** tab, a reconnect **will be** performed when you switch sessions/projects. This is by design.

Group Multiple Projects

Another way to work with multiple projects is in groups. Select *File>Project Group* and pick or create a project group using the Create New Group dialog box.

6.4 Work with Multiple Configurations

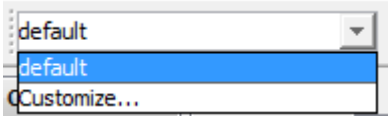
MPLAB X IDE allows multiple build configurations for the same project. This may be useful for code that can be compiled on multiple platforms (such as the Microchip Application Libraries Demo Projects.)

When you create a new project, a "default" configuration is created. To create additional configurations, follow the steps in these sections:

6.4.1 Manage Configurations Dialog

To create your own configuration, start by doing one of the following:

- Use the drop down menu on the toolbar and select "Customize". The Project Properties window will open.

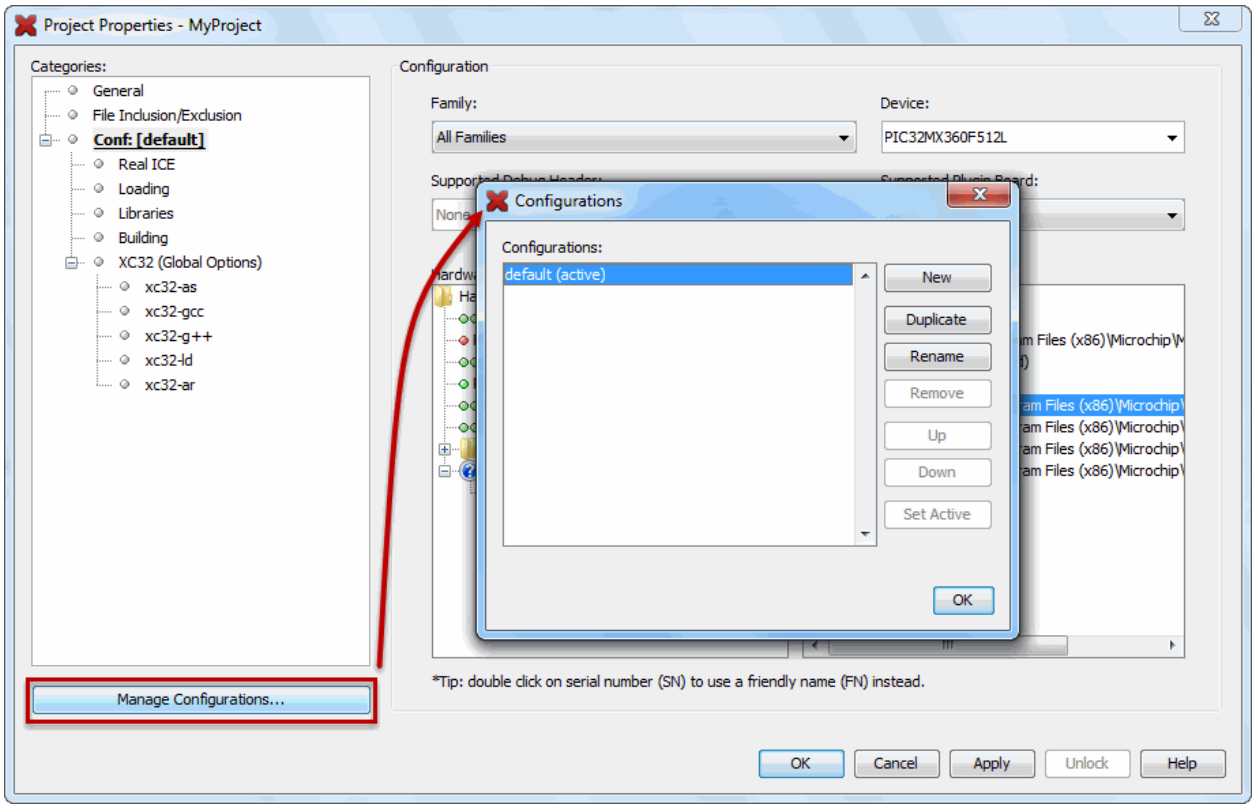


- Open the Project Properties window by right clicking on the project name and selecting "Properties".

In the Project Properties window, click **Manage Configurations** to open the Configurations dialog. Existing configurations can be renamed or a new configuration can be added or duplicated from an existing one. Any spaces in the name will be replaced with underscores.

When more than one configuration is created for a project, the active one can be selected from **Manage Configurations** or from the drop-down menu.

Figure 6-2. Project Properties – Configurations Dialog

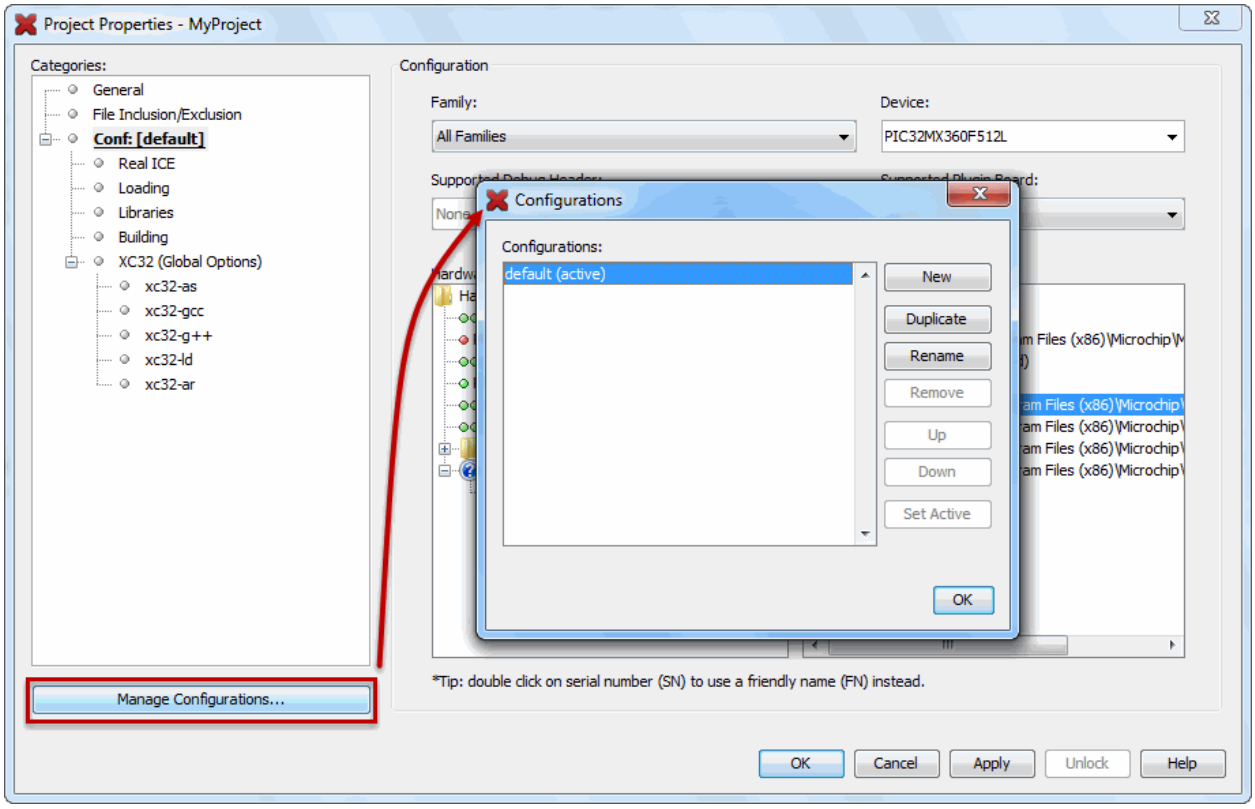


6.4.2 Add a New Configuration

To add a new configuration to your project:

1. Click **Manage Configurations** in the Project Properties window.
2. In the Configurations dialog, select a project configuration and click **New**.
3. Enter a name in the New Configuration Name dialog, such as "My Test" (see figure below). Click **OK** to close the dialog. Under "Configurations," the name will be changed to "My_Test" since spaces are not allowed.
4. Click **OK** to return to the Project Properties window. The debug configuration (Conf: My_Test) should now be visible.

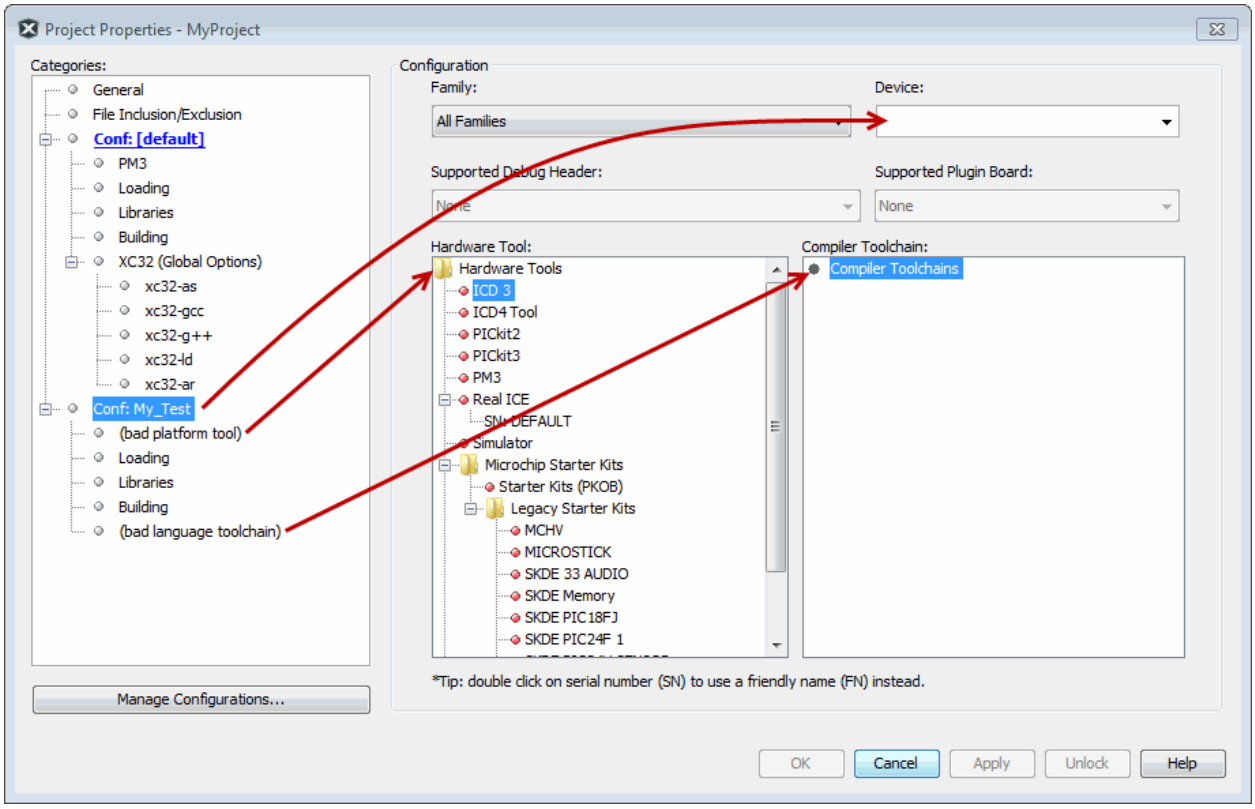
Figure 6-3. Create New Configuration



Once a new configuration is added, some items must be assigned in the Project Properties window (see figure below).

1. Device – You must select this first to see the hardware tool and compiler support.
2. Hardware Tool
3. Compiler Toolchain

Figure 6-4. New Configuration



6.4.3 Add a Duplicate Configuration

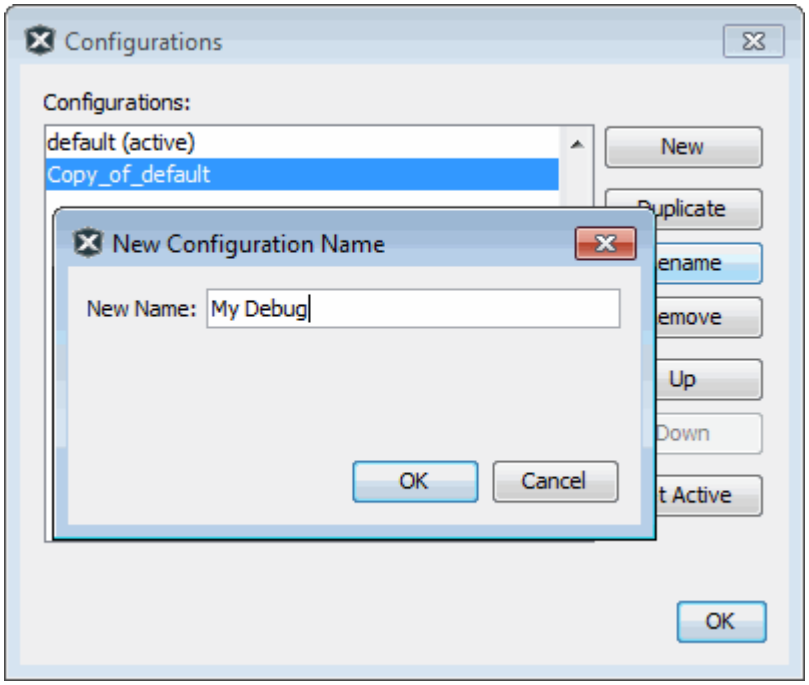
You can add a configuration that is the duplicate of an existing one and then make edits to it.

One reason to make a duplicate configuration is to create your own debug configuration. Although MPLAB X IDE provides debug macros for use with Microchip tools (see "Debug Macros Generated" in 4.13 Debug Code), you may want to use your own debug macros or you may want to set up the same debug capabilities with third-party tools.

To set up a **debug configuration**:

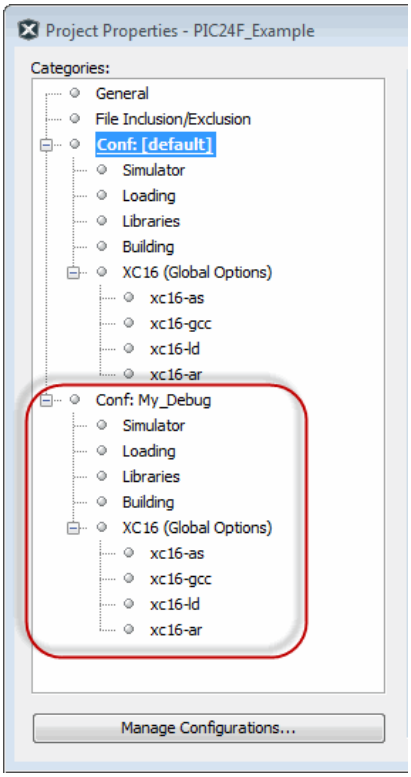
1. Click **Manage Configurations** in the Project Properties window.
2. In the Configurations dialog, select a project configuration and click **Duplicate**.
3. Click **Rename** and enter a name in the New Configuration Name dialog, such as "My Debug" (see figure below). Click **OK** to close the dialog. Under "Configurations," the name will be changed to "My_Debug" since spaces are not allowed.

Figure 6-5. Create Debug Configuration



4. Click **OK** to return to the Project Properties window. The debug configuration (Conf: My_Debug) should now be visible (see figure below).

Figure 6-6. Debug Configuration



To use macros in code to switch to other configurations, like debug, see 6.4.6 Configuration Macros.

6.4.4 Manage Files in Configurations

You can manage your project files by assigning them to individually or as a group to the configurations you have created.

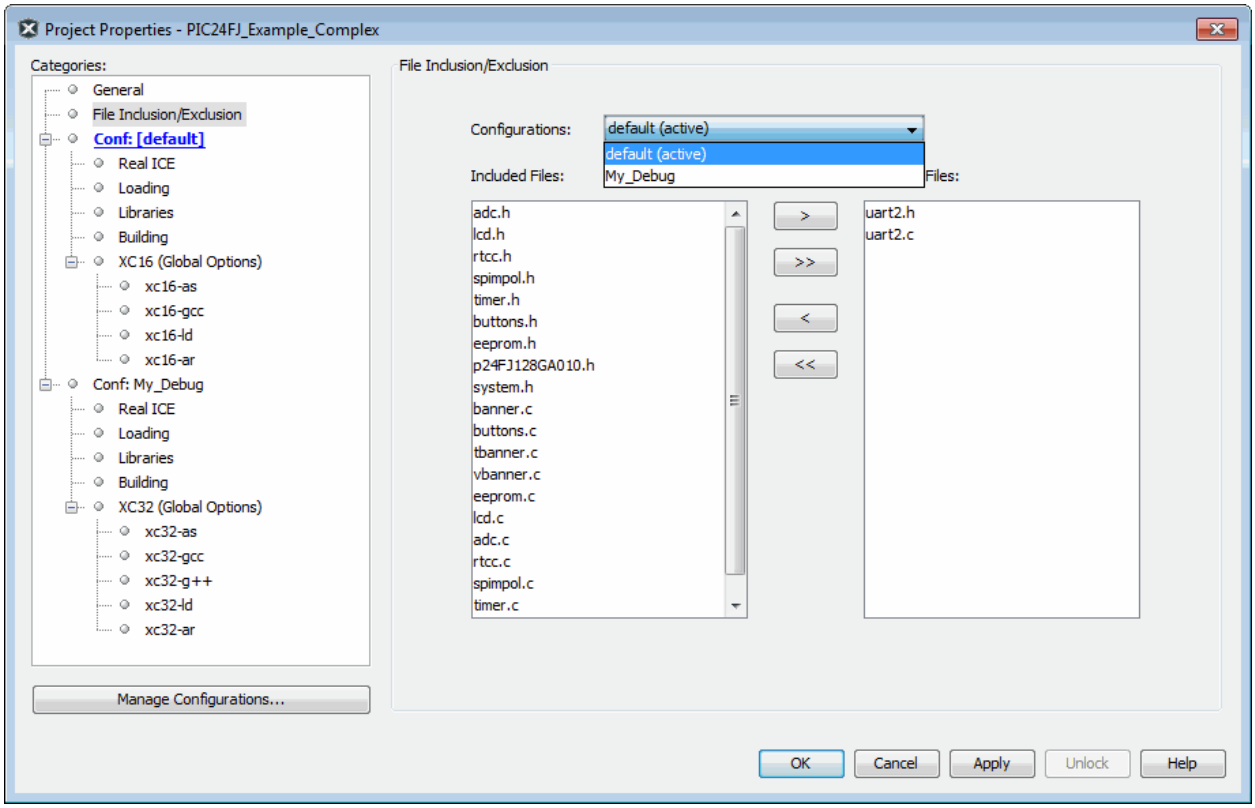
Projects Window

Select the configuration you want in the "Set Project Configuration" drop-down box on the toolbar. Then select the file or files (Shift-Click or Ctrl-Click) and right click on one of them to pop up a context menu. Select "Exclude file(s) from current configuration." To include a file or files again, repeat to see on the context menu "Include file(s) from current configuration."

Project Properties Window

To set the configuration you want to apply to many files, right click on the project name in the Projects window to open the Project Properties window. Select the "File Inclusion/Exclusion" category. Select a configuration from the Configurations drop down box and then specify which files to include or exclude from this configuration.

Figure 6-7. Project Properties Configuration



6.4.5 Configuration Names

Some special characters cannot be used in configuration names. This is due to being able to use the configuration name within the source code as a define macro ("Configuration Macros").

Special characters like a hyphen (-), ampersand (&), etc., will be automatically replaced with an underscore (_).

6.4.6 Configuration Macros

When you add a new or duplicate configuration, you can create your own configuration-related macros or use autogenerated macros for #ifdef or other statements in code. These are MPLAB XC C compiler preprocessor macros.

User-Defined Macros

To define your own preprocessor macro:

1. In the Project Properties window, click on a project configuration to make it active. Under that configuration, click on the compiler in the toolchain.
2. Under the "Preprocessing and messages" options category, locate the option that allows you to define a macro and click on the associated text box.
3. In the pop-up dialog, enter a macro name and click **OK**. This macro is now associated with the active configuration.
   You can name the macro the same as a configuration (such as "My_Debug" from 6.4.3 Add a Duplicate Configuration) or you can give it any other name (such as "DebugConfig").

   Macro names (and strings, discussed below) are case-sensitive.

You can now use the preprocessor macro in conditional text:

```
// If My_Debug configuration is active
#ifdef DebugConfig
fprintf(stderr,"This is a debugging message\n");
#endif
```

In addition, you can assign a string value that matches the configuration name to your macro(s) for use in regular code:

```
#ifdef DebugConfig
#define CFG_NAME "My_Debug";
// other defines
#endif
#ifdef DefaultConfig
#define CFG_NAME "My_Default";
// other defines
#endif
:
int main(int argc, char** argv)
{
printf(CFG_NAME);
return{EXIT_SUCCESS};
}
```

The `CFG_NAME` macro will always hold the active configuration's name.

Autogenerated Macros

When a new or duplicate configuration is created, for example `My_Test`, a related macro is also created, for example `XPRJ_My_Test`. The macro will have a string value which matches the name of the configuration ("My_Test").

You can use the macro for `#ifdef` statements as in the previous section, or you can use the value of the macros at compile time as follows:

```
int main(int argc, char** argv)
{
printf(CFG_NAME);
return{EXIT_SUCCESS};
}
```

The `CFG_NAME` macro will automatically hold the active configuration's name.

6.5 Create Dual Core Projects

A dual core device (e.g., dsPIC33CH DSCs) has a Master core and Slave core that can operate independently and can be programmed and debugged separately, during application development. Both processor (Master and Slave) subsystems have their own interrupt controllers, clock generators, ICD, port logic, I/O MUXes and PPS. The device is equivalent to having two complete dsPIC® DSCs on a single die.

Dual core devices have several modes of operation which correspond to different ways of creating and using projects:

Master Only Programming and Debugging

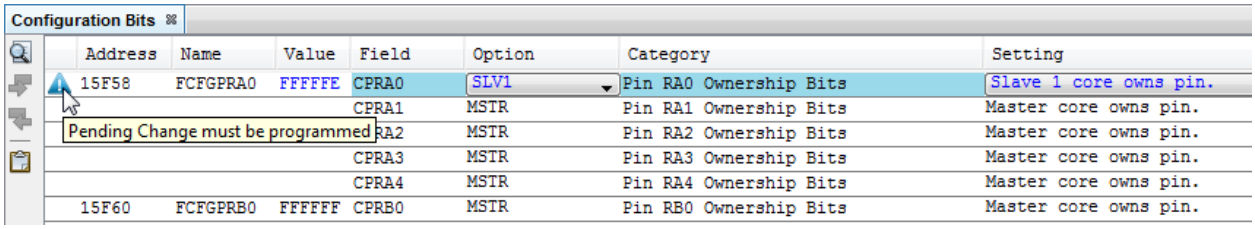
A dual core device is by default in Master Only mode; the Master Slave Interface (MSI) has the MSI1 Master Control Register (MSI1CON) Slave Enable bit (SLVEN) set to '0'.

A project for this device mode is no different from a project for any Microchip device. For details, see "Create a New Project."

Slave Only Programming

For the slave project to work independently, the ports for the slave need to be assigned. Therefore a master project (e.g., MasterStub) is necessary that programs the master with the configuration bit settings (like which PORTS are assigned to slave). For details, see "Set Configuration Bits in the Configuration Bits Window."

Figure 6-8. Master Configuration Bits



Two project are required for Slave Only Programming mode:

1. Create a MasterStub project per "Create a New Project" and be sure to enable the slave (SLVEN = '1'). Set the configuration bits as described previously.
2. Create a SlaveOnly project per "Create a New Project" but use the "S1" version of the device, e.g., dsPIC33CH128MP508S1.

MasterStub must be programmed first and then SlaveOnly.

Slave Only Debugging

To debug the SlaveOnly project in the previous section, enable background debug (`S1DEBUG`) in the configuration settings:

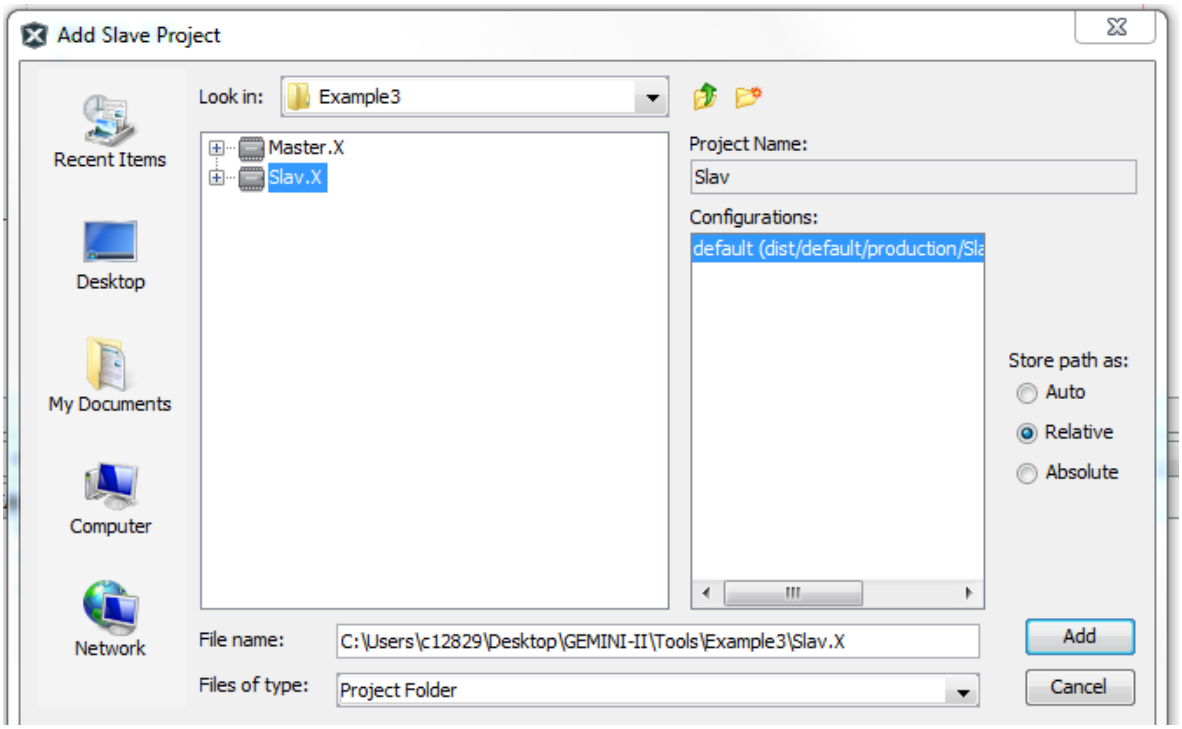
```
// FS1ICD
#pragma config S1ICS = PGD2 // ICD Communication Channel Select bits
                            // (Communicate on PGEC2 and PGED2)
#pragma config S1DEBUG = ON // Background Debug (Enabled)
 :
```

Master and Slave Programming and Debugging

In this mode, the Master project will be linked to the Slave project. That is, the Slave code resides in Master and Master must transfer the code to slave.

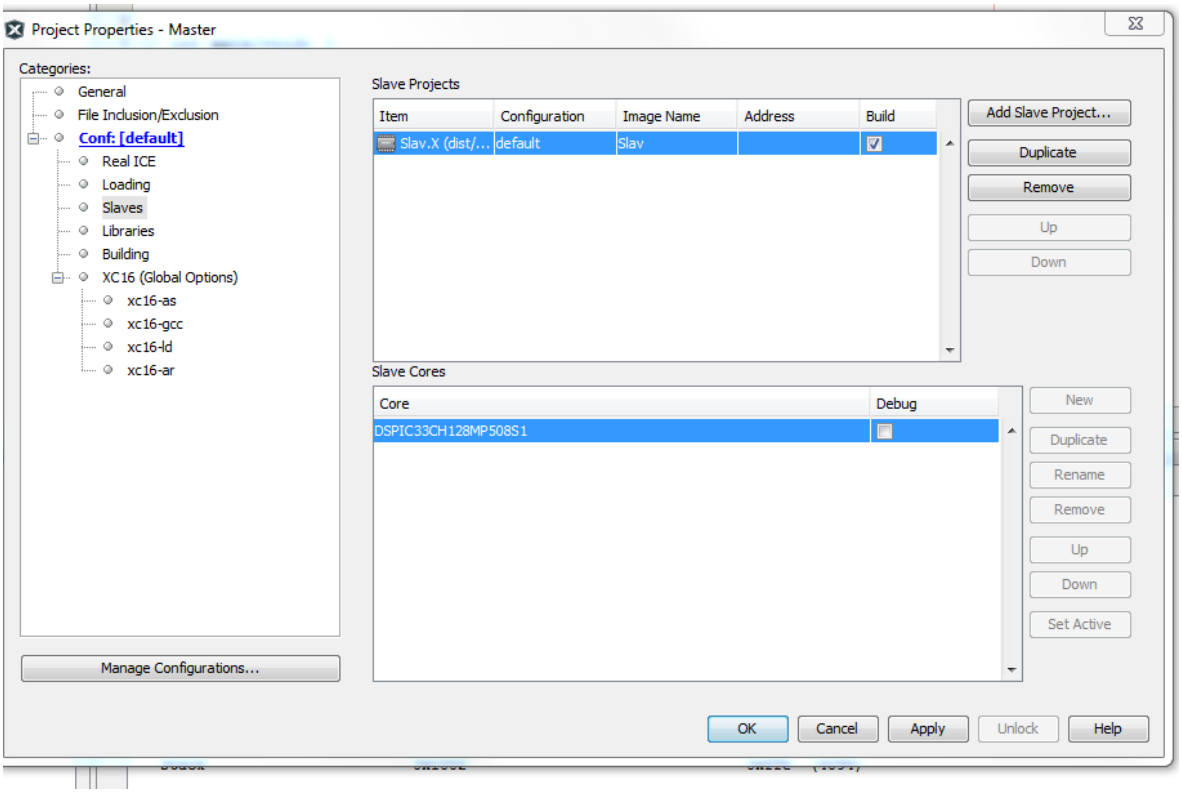
1. Create Master and Slave projects as in "Slave Only Programming."
2. In the Projects window, the Master project will have a folder named "Slaves." Right click on the folder and click "Add Slave Project." Then select the Slave project in the dialog and click **Add**.

Figure 6-9. Add Slave Project



3. Right click on the Master project and select "Properties" to open the Project Properties window. Click on the "Slaves" category and make sure to check the "Build" checkbox. Click **Apply**.

Figure 6-10. Master with Slave Project



4. In the Master code add:

```
#include "Slav.h"
//The function to transfer the code from slave to master and start the slave is below.
 _program_slave(1,0,Slav);
 _start_slave();
```

5. By building and programming Master, both Master and Slave projects will be built and both Master and Slave cores will be programmed.

6.6 Create User Makefile Projects

Create a project that makes use of an external makefile. This is useful if you have a project that was built outside of MPLAB X IDE, but now you want to use MPLAB X IDE to do debugging.

To create the makefile project, use the New Project wizard.

6.6.1 New Project - User Makefile

For ways to open the New Project wizard, see 4.1.2 Launch New Project Wizard.

The wizard will launch to guide you through new project setup.

Step 1. Choose Project: Select the "Microchip Embedded" category and choose from the project type "User Makefile Project."

Step 2. Select Device: Select the device used in your project from the "Device" drop-down list. To narrow your selection list, choose a Family first.

Step 3. Select Header: This step will appear if a header is available for your selected device. To determine if a header is required for debug, or if your device has on-board debug circuitry, consult one of the documents below. Then choose whether or not to use a header.

- Processor Extension Pak and Debug Header Specification
- Emulation Extension Pak and Emulation Header User's Guide

Step 4. Select Tool: Select the development tool you will be using to debug your application from the list. The level of tool support for the selected device will be displayed next to the tool name. That is, green for full support, yellow for beta support and red for no support, yet.

Step 5. Select Plugin Board: This step will appear if a plugin board is available for your selected device. Drop down the list under "Supported Plugin Board" to see what it available.

Step 6. Select Project Name and Folder: Select a name and location for your new project. See Table 1.

Step 7. Create User Makefile Project: Enter data to set up your makefile project (See Table 2). Instructions for General Data Entry in this window are:

- Forward (/) slashes are recommended in your path. However, you may use backward (\) slashes instead, though you may need to escape them (\\).
- A path and file name without spaces is recommended. However, if you do have spaces, use escape spaces (backward slash before the space) when you enter a path and file name in this window. GNU Make is known to cause weird issues with spaces as spaces are their separators.

Step 8. Finish: The new project will open in the Projects window.

Change Makefile project settings in the Project Properties window, "Makefile" category, after the project is created.

For information on exporting a project as hex, see "Project Menu" in 12.15 Projects Window.

Table 6-2. Table 1: Select Project Name and Folder Options

Item	Description
Project Name	Specify a name for your project.
Project Location	Enter or browse to a location for your project.

.....continued	
Item	**Description**
Project Folder	View the folder location based on the Project Name and Project Location. For information on the relation of the project folder to the working folder, see 6.6.2 User Makefile Project Folder and Working Folder.
Set as main project	Set project created as the main project. Checked by default.
Overwrite existing project	If project name has been used already, this checkbox will be active.
Also delete sources	If project exists already and contains source code, this check box will be active.
Use project location as project folder	Create the project in the same folder as the project location. This is useful if you are importing an MPLAB IDE v8 project and want to maintain the same folder for your MPLAB X IDE project (as when you want the build to be the same). MPLAB X IDE uses a folder to store a project information whereas MPLAB IDE v8 uses an .mcp file. Therefore you can only have one MPLAB X IDE project share a folder with an MPLAB IDE v8 project (.mcp file).
Encoding	By default set as ISO-8859-1. There is usually no need to change this.

Table 6-3. Table 2: Create User Makefile Project Options

Item	Description
Working Directory	Specify the location of the external project. This is the default location from where the build, debug build, and clean commands will be executed. For information on the relation of the project folder to the working folder, see 6.6.2 User Makefile Project Folder and Working Folder. Browsing to the directory is recommended as this will take care of formatting issues for you*.
Build command	When the command to Run is requested in MPLAB X IDE (icon or menu), perform the build according to this command instead. Build path*: For Windows, use quotes. For Linux or Mac OS, any spaces must be escaped.
Debug build command	When the command Debug is requested in MPLAB X IDE (icon or menu), perform the debug build according to this command, instead. Debug build path*: For Windows, use quotes. For Linux or Mac OS, any spaces must be escaped. Also see: http://microchipdeveloper.com/mplabx:work-outside-compile-for-debug
Clean command	When the command to Clean is requested in MPLAB X IDE (icon or menu), perform the clean according to this command, instead. Clean path*: For Windows, use quotes. For Linux or Mac OS, any spaces must be escaped.
Image name	Enter the path and name of the production image file (hex). Image path*: Any spaces must be escaped.
Debug image name	Enter the path and name of the debug image file (ELF or COF). Debug image path*: Any spaces must be escaped.
User Include Paths	Enter or browse to user include file(s). Manage include paths in the dialog (Figure 1).

.....continued

Item	Description
User Macros	Add or parse-and-add macros. Edit: Enter a macro or browse to a file containing one. Manage macros in the dialog (Figure 2). Parse: Automatically parse a macro (MPLAB XC compilers only). Follow the instructions on the dialog. Once complete, choose to Replace any current macro(s) listed or Append to any current list of macros. See 6.6.3 Example of a Makefile Project for an example of how to use this dialog.

Figure 6-11. Select Include Paths Dialog

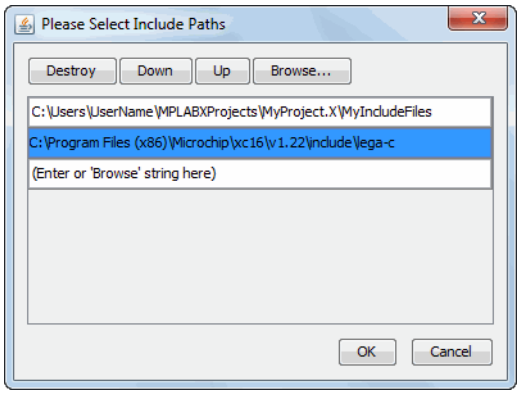
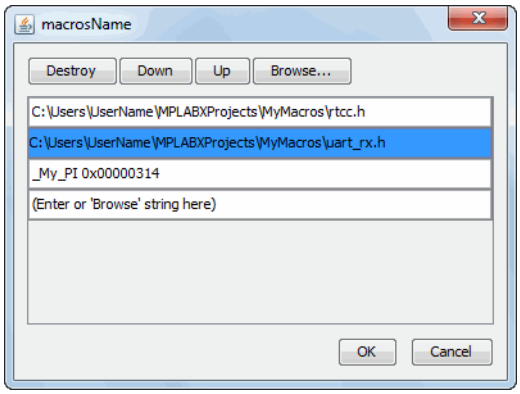


Figure 6-12. macrosName Dialog



6.6.2 User Makefile Project Folder and Working Folder

The project folder contains its own `Makefile` and `nbproject/Makefiles`. MPLAB X IDE will run the makefile in the project folder, then change to the working folder (directory) and run the "Build command" or "Debug build command." In effect, GNU Make is run and then whatever is specified as a build command is run.

The project folder may be located anywhere, with relation to the working folder, if absolute paths are used to define locations. However, this will make your project less portable. Make the working folder relative to the project folder (e.g., one level below the project folder) for your project to be more portable.

6.6.3 Example of a Makefile Project

This example shows how to create a user makefile project in MPLAB X IDE that will run either a batch file or make on user code that was created outside of the IDE. The user code consists of a single file `t.c`. The simple code contained in this file is listed below.

Example: `t.c` Code

```
#include <xc.h>
int main(void) {
while(1) {
```

```
}
return 0;
}
```

For production the code is linked into `t_production.hex` and for debug the code is linked into `t_debug.elf`.

For this example, `t.c`, the batch file `makeme.bat` and the user makefile `Makefile` are assumed to be in the (Windows OS) directory:

```
C:\Users\MCHP\MPLABXProjects\makestuff\myOwnCodeWithItsOwnMakefile
```

Substitute your *`UserName`* for `MCHP` to recreate the example on your PC.

- User Makefile Project – Create Code from a Batch File
- User Makefile Project – Create Code from User Makefile
- User Makefile Project – Complete
- User Makefile Project – Parse Macros

6.6.3.1 User Makefile Project – Create Code from a Batch File

To compile the code, the simple batch file `makeme.bat` takes as arguments `production`, `debug` or `clean`. If `production` is passed then `t_production.hex` is built. If `debug` is passed then `t_debug.elf` is built. If `clean` is passed, then the `.elf` and `.hex` files are erased.

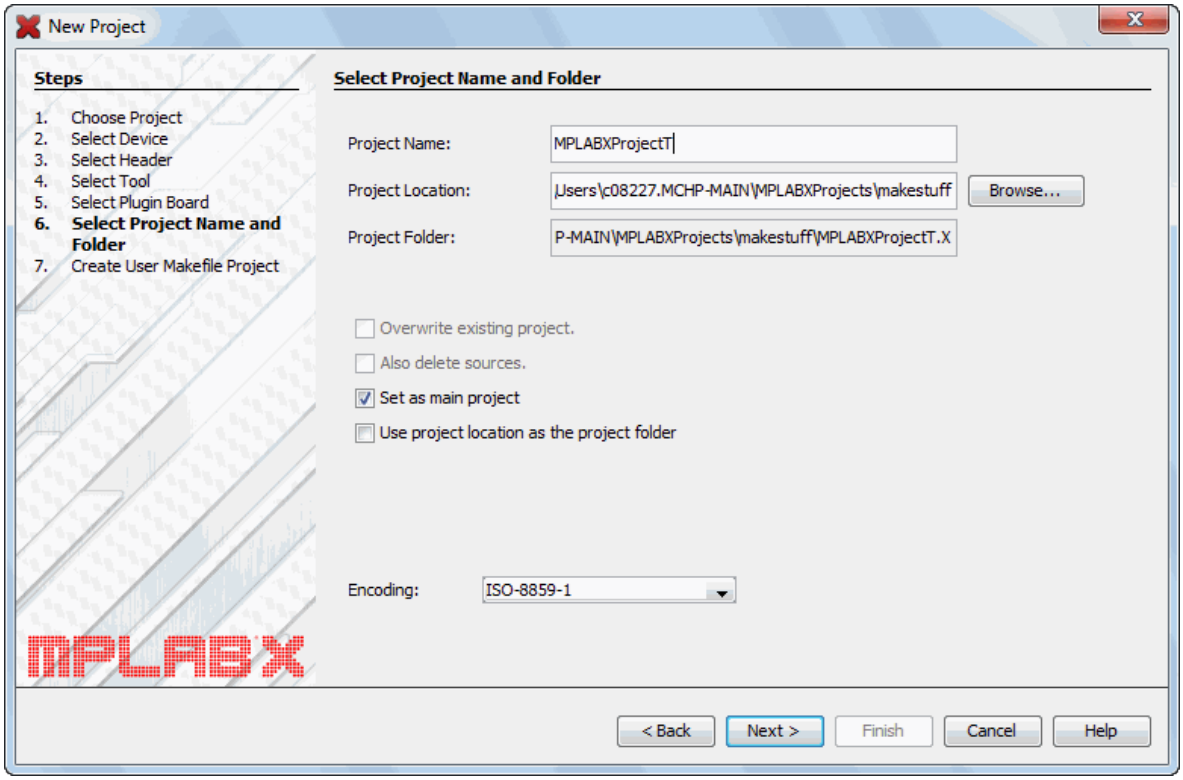
Example: `makeme.bat` Code

```
::assumes xc32-gcc and xc32-bin2hex are on the path
rem @echo off
set COMPILER=xc32-gcc
set BIN2HEX=xc32-bin2hex
set PROCESSOR=32MX360F512L
if "%1" == "production" goto production
if "%1" == "clean" goto clean
:debug
%COMPILER% -mprocessor=%PROCESSOR% -g -D_DEBUG -o t_debug.elf t.c
goto end
:production
%COMPILER%  -mprocessor=%PROCESSOR% -o t_production.elf t.c
%BIN2HEX% t_production.elf
goto end
:clean
del /F *.elf
del /F *.hex
goto end
:end
```

To create a new user makefile project in MPLAB X IDE, follow the steps in 6.6.1 New Project - User Makefile:

1. **Choose Project:** Select "Microchip Embedded," "User Makefile Project."
2. **Select Device:** Select PIC32MX360F512L as the device.
3. **Select Header:** None available for this device.
4. **Select Tool:** Select the hardware tool by SN or the Simulator. For this example, the Simulator was used.
5. **Select Plugin Board:** None used for this example.
6. **Select Project Name and Folder:** For this example, the Project Folder will be a sibling to `myOwnCodeWithItsOwnMakefile`. Therefore the Project Location will be: `C:\Users\MCHP\MPLABXProjects\makestuff`. The Project Name is MPLABXProjectT.

Figure 6-13. User Makefile Project – Select Project Nane and Folder



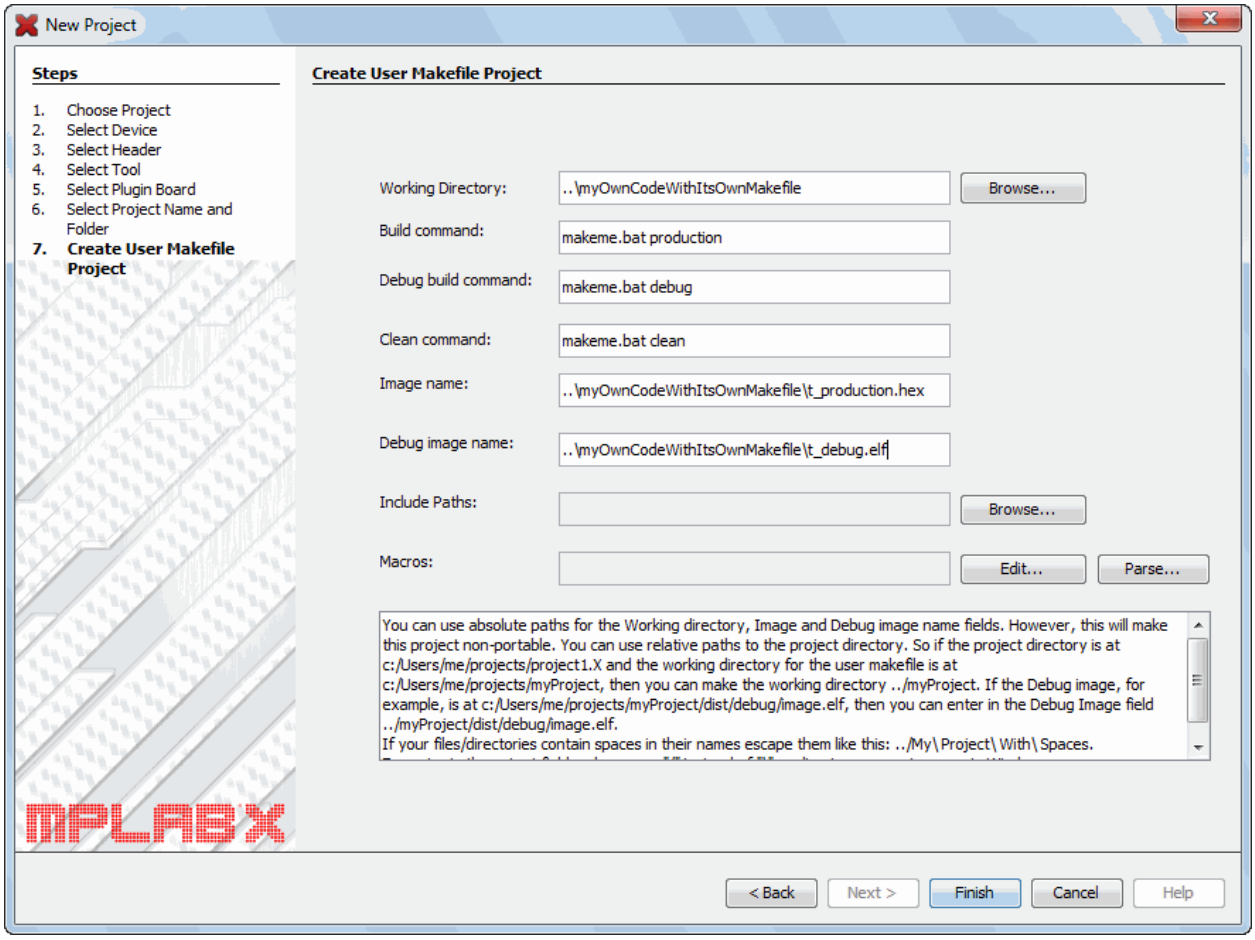
7. **Create User Makefile Project:** Enter information to set up your makefile project.
The Working Directory is where the batch file build, debug build and clean commands will be run. This directory could be set as the absolute path:

```
C:\Users\MCHP\MPLABXProjects\makestuff\myOwnCodeWithItsOwnMakefile.
```

Doing this, however, makes the MPLAB X IDE project less portable. Therefore the Working Directory, Image name and the Debug image name will be entered as relative paths with respect to the MPLAB X IDE project directory.

Click **Finish** to create the project.

Figure 6-14. User Makefile Project – Create from Batch



6.6.3.2 User Makefile Project – Create Code from User Makefile

To compile the code, the simple user makefile `Makefile` takes as arguments `production`, `debug` or `clean`. If `production` is passed then `t_production.hex` is built. If `debug` is passed then `t_debug.elf` is built. If `clean` is passed then the `.elf` and `.hex` files are erased.

Example: User Makefile Code

```
# Assumes xc32gcc and xc32bin2hex are on the path
COMPILER=xc32gcc
BIN2HEX=xc32bin2hex
PROCESSOR=32MX795F512L
production: t_production.hex
debug: t_debug.elf
t_debug.elf: t.c
$(COMPILER) mprocessor=$(PROCESSOR) -g -D_DEBUG -o t_debug.elf
t.c
t_production.hex: t_production.elf
$(BIN2HEX) t_production.elf
t_production.elf: t.c
$(COMPILER) mprocessor=$(PROCESSOR) o t_production.elf t.c
clean:
del /F *.hex
del /F *.elf
```

To create a new user makefile project in MPLAB X IDE, follow steps 1-6 from "User Makefile Project – Create Code From a Batch File". For Step 7, set up as per figure below.

The Working Directory is where the user Makefile build, debug build and clean commands will be run. The Working Directory, Image name and the Debug image name are entered as relative paths with respect to the MPLAB X IDE project directory.

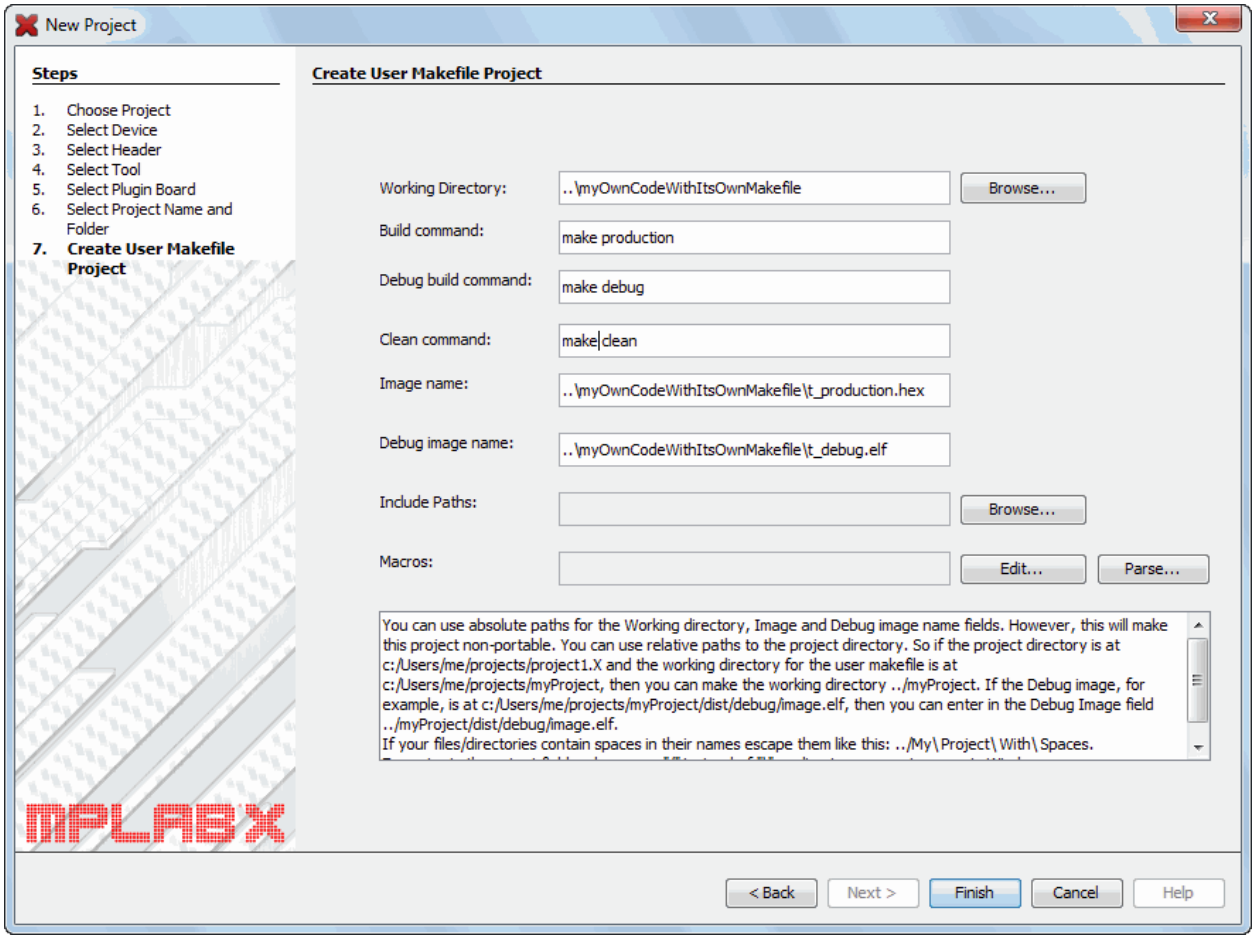
MPLAB X IDE uses GNU Make to build, debug build and clean. As per "User Makefile Project Folder and Working Folder", MPLAB X IDE will run the Makefile in the project folder, which will change directories to the working folder. There, make will look for a makefile and find the user-defined Makefile, which will define how to build, debug build, and clean.

Note: Another way to define commands, for example the Build command, would be to use the `-f` option to specify the user makefile, as in:

```
make -f Makefile production.
```

Click **Finish** to create the project.

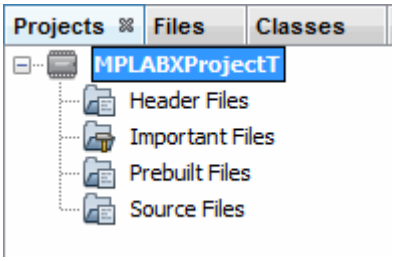
Figure 6-15. User Makefile Project – Create from Makefile



6.6.3.3 User Makefile Project – Complete

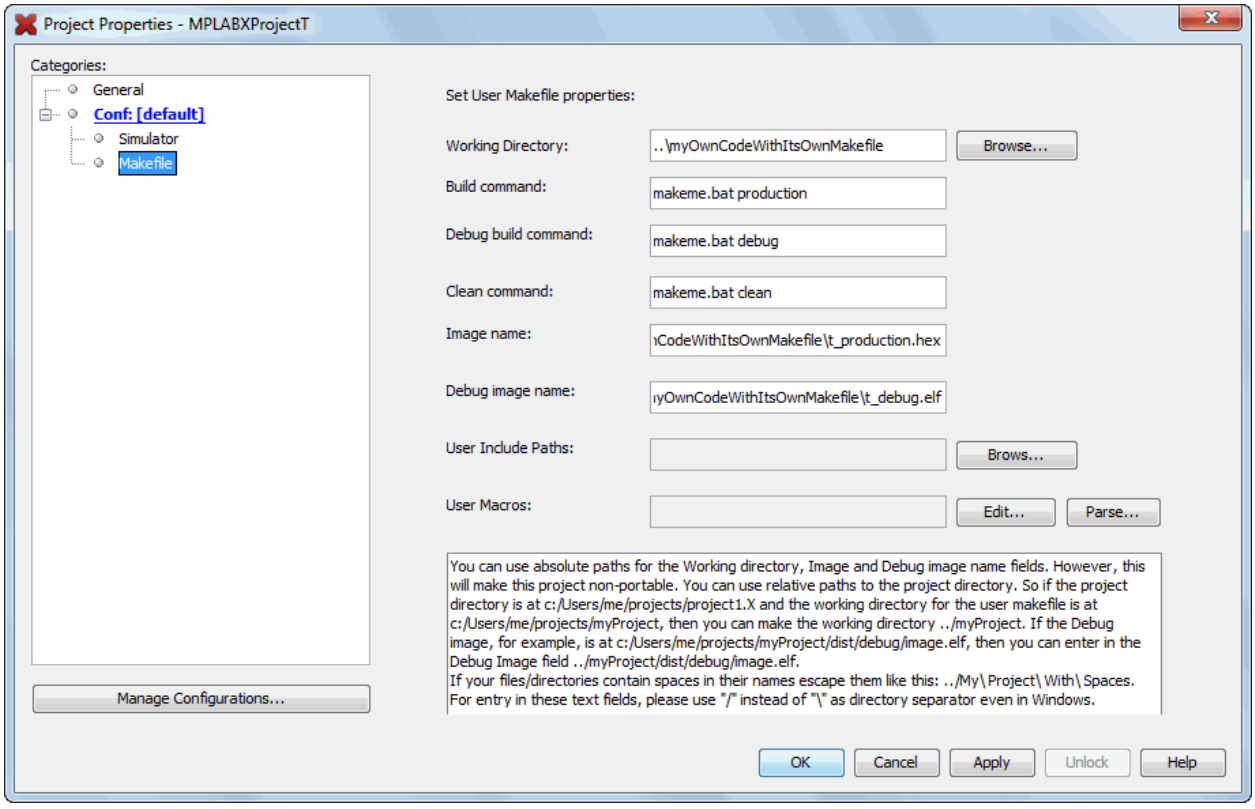
Once the project has been created, you should see the following in the Projects window. You can right click on the project name to build the project.

Figure 6-16. User Makefile Project – Project Tree



If you wish to change settings, right click on the project name and select "Properties."

Figure 6-17. User Makefile Project – Project Properties



For this example, the results of build (`t_production.hex`) and debug build (`t_debug.elf`) are found in the `myOwnCodeWithItsOwnMakefile` working directory. As only the MPLABXProjectT project directory is visible in MPLAB X IDE, you will not see any changes in the IDE Projects window after a build.

6.6.3.4 User Makefile Project – Parse Macros

You can parse and add macros for MPLAB XC C compilers to your makefile project. To do this you need to call the compiler on the command line with the relevant arguments and copy the whole output to the first panel of the "Parse macros" dialog.

To run the compiler on the command line, you will need a C file. For this example, use the code below and paste into a file named `xcmain.c`:

```
#include <xc.h>
void main(void) {
return;
}
```

Ensure that your computer knows the Path to your MPLAB XC compiler. Then execute the following on the command line:

```
xc32-gcc -dM -E xcmain.c > macros32.txt
```

The MPLAB XC32 C compiler is used, not the MPLAB XC32++, so xc32-gcc is the executable. The option `-dM` tells the preprocessor to output only a list of the macro definitions that are in effect at the end of preprocessing. The option `-E` stops execution after the preprocessing stage. The redirect (>) character sends the output into the file `macros.txt`.

For other MPLAB XC C compilers, you can use:

```
xc16-gcc -dM -E xcmain.c > macros16.txt
xc8 --pre -v --chip=8BitMCU xcmain.c > macros8.txt
```

where `8BitMCU` should be replaced with your 8-bit device. The `macros8.txt` file will also require additional editing to isolate the macros from other text in the file.

To open the "Parse macros" dialog:

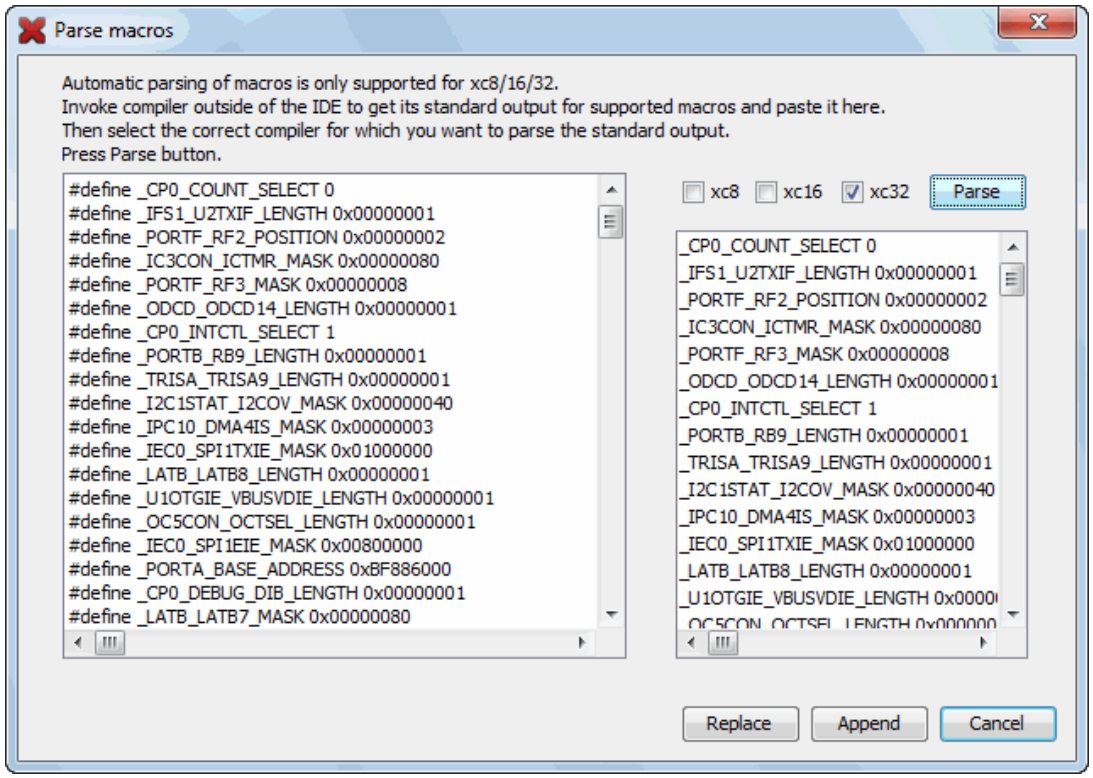
1. Right click on the project name and select "Properties."
2. Under "Categories," select "Makefile."
3. Under "Set User Makefile properties," go to "User Macros" and click on the **Parse** button.

To parse the macros:

1. Copy and paste the contents of `macros.txt` into the dialog left panel.
2. Check the checkbox for your MPLAB XC C compiler, in this case "xc32."
3. Click the **Parse** button.

Click a button to **Replace** the current contents of the User Macros text box, **Append** to existing text in the text box, or **Cancel** to leave the existing text unchanged.

Figure 6-18. Parse Macros Dialog



6.7 Use Linked Resources for Source File Folders

A macro can be used to point to a source base (or library version) and changed to move to another source base. This is particularly useful in MPLAB Harmony applications.

Open a project in MPLAB X IDE and select *File>Project Properties*. Click on the "General" category and click "Browse" next to "Macro." Find and Open the folder containing the source base. Click **OK**. In the project, open file(s) from this source base as "Relative to Macro."

To change to another source base, browse to the folder that contains the new source base under "Macro" and select it. Any files from the old source base that match the new source base will now refer to the new source base.

6.8 Work with Third-Party Hardware Tools

When working with third-party hardware tools, these basic installation requirements apply:

- Acquire and install any related USB drivers for the hardware. USB drivers are usually available from the third-party site, along with installation instructions.
- Install the MPLAB X IDE plugin for this tool. For details, see 5.26 Add Plugin Tools.

When installation is complete, you should be able to see and select the hardware tool in the Project Properties window (see 4.4 View or Make Changes to Project Properties).

6.9 Use Code Coverage

Code coverage provides visibility as to what portions of the code are being executed. To view code coverage output, an MPLAB XC C compiler that supports code coverage must be used with MPLAB X IDE. Support for code coverage begins with these versions:

- MPLAB X IDE v5.25
- MPLAB XC8 v2.10
- MPLAB XC16 v1.40
- MPLAB XC32 v2.30

In MPLAB X IDE, you can enable code coverage in the Project Properties window under a compiler category. Once enabled, debug your test code to execute it completely and then halt/pause. Select *Window>Debugging>Code Coverage* to see the percentage of covered code executed/total code covered in the **Code Coverage** tab window.

For more detailed coverage information, you can purchase an MPLAB Code Coverage add-on license (SW006026-COV). This will activate the full code coverage features of your compiler to include:

- Editor text highlighted by colors representing the coverage
- a detailed report written to the Code Coverage tab window

For more information, see:

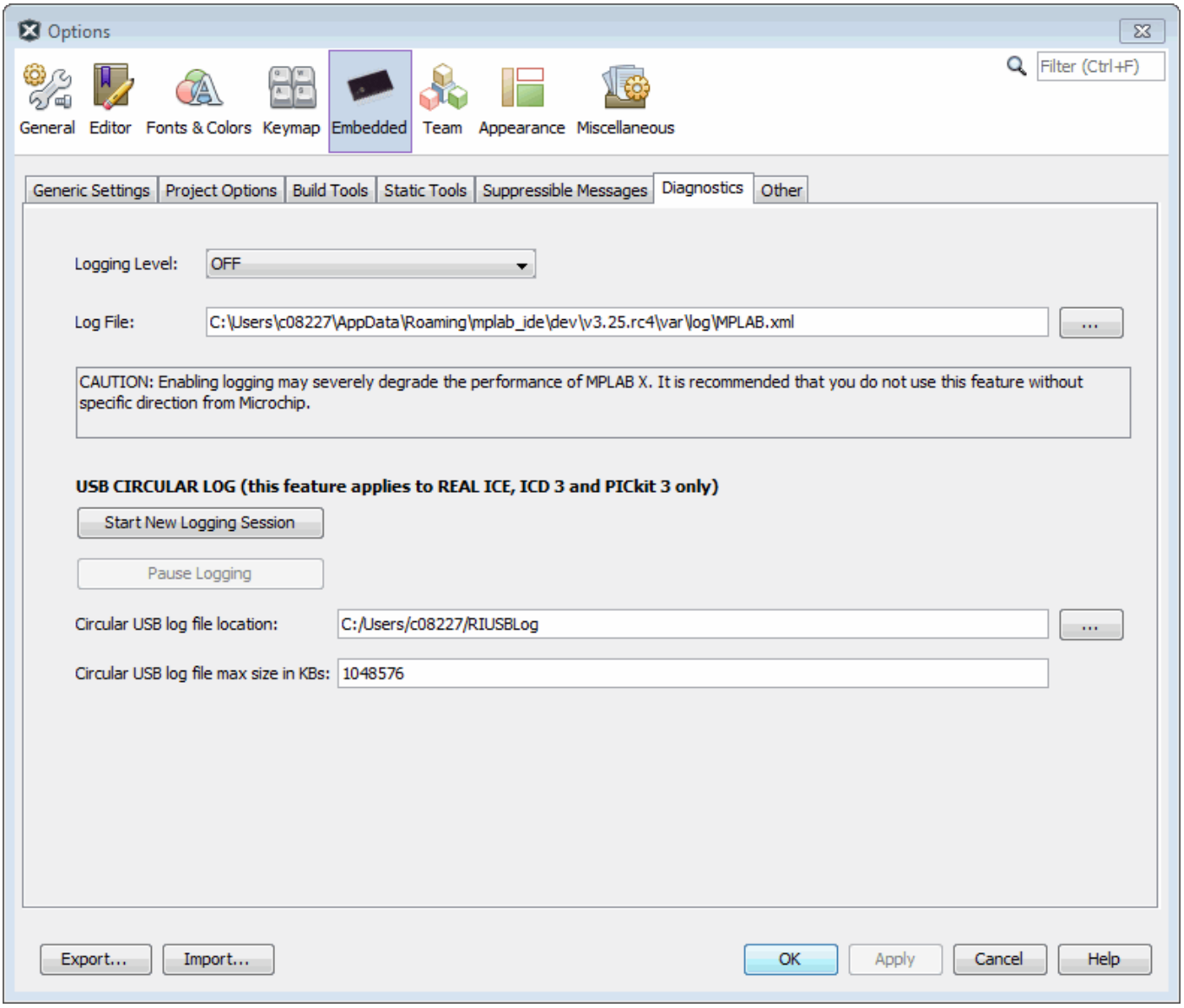
http://www.microchip.com/mplab/codecoverage

6.10 Log Data

Log files are useful for capturing your program execution and debugging problems.

When you have an MPLAB X IDE error or issue and need to contact technical support, you may be asked to capture data in two log files: MPLAB X IDE log file and the NetBeans platform log file. Other supporting information will be requested as well.

Figure 6-19. Data Logging



6.10.1 MPLAB® X IDE Log File

The MPLAB X IDE log file generates data based on the Java Logger class.

To set up a log file:

1. Select *Tools>Options* (*MPLAB X IDE>Preferences* for macOS), **Embedded** button, **Diagnostics** tab.
2. Select a logging level from the drop-down box.
   Note: The higher the logging level, the more data is collected, but the slower your application will run.
3. Select a location (path) for the log file.

To log data:

1. Set up the log file using logging level "Finest."
2. Take note of the log file name and location.
3. Repeat the steps to cause the error or issue.
4. Find the log file(s) and send to technical support along with other requested information.

6.10.2 NetBeans Platform Log File

The NetBeans log file generates information on the executing NetBeans Platform.

To log data:

1. Open the log file the Output window by selecting *View>IDE log*.
2. Right click in the window and select "Clear."
3. Repeat the steps to cause the error or issue.
4. Right click in the window and select "Save As" to save the text to a file.
5. Send the file to technical support with other requested information.

6.10.3 Additional Supporting Information

Although logs are useful, other information will be required and requested by technical support to help them resolve your issue. This information includes:

- The steps performed that correspond to the log file.
- Details about the project tools, such as which debug and language tools were used and which versions of these tools were selected.
- Other MPLAB X IDE project data as necessary.

6.10.4 Logging Considerations

When logging:

- Keep the log files as short and focused as possible. That is, instead of running code and logging for a long time till an error occurs, attempt to capture a more focused run just before the error.
- For hardware tools, a circular log is useful in determining USB communication issues.

6.11 Package an MPLAB X IDE Project

Right click on your project and select "Package" to package your project into a zip file (`.zip`) file. While MPLAB X IDE has the ability to zip files up, it cannot unzip them. So a separate program will be required to unzip the project.

To package the project, this feature examines the project file to determine the location of the project files to include. Only files contained in the project folder and those using relative, not absolute, paths will be included.

Items included in the package are the source files, `makefile` and `nbproject` directory.

Header files are not included unless they have been added to the project. Therefore, the unzipped project may not build if any user-generated header files are required.

6.12 Hardware Tool Connections and Debugging

The connection between your hardware debug tool and target can determine what debug features are available, in addition to the device-related debug features.

Hardware Tool to Target Connections - PIC MCUs

Table 6-4. Hardware Tool to Target Connections - PIC MCUs

Connection	Hardware Support(1)	Debug Support (2,3,4)	Trace Support(1)	Support Speed
Standard (ICSP) Communications	**RI**, **ICD4**, **ICD3**: Modular cable (6-pin RJ-11); **Snap**, **PK4**, **PK3**: Ribbon cable (6-pin SIL)	Basic and Advanced	**RI**: Native Trace	15 MIPS or less PIC32: Device Dependent

.....continued

Connection	Hardware Support[1]	Debug Support [2,3,4]	Trace Support[1]	Support Speed
High-Speed (LVDS) Communications	**RI**: Performance Pak (AC244002)	Basic and Advanced	**RI**: Native Trace, **RI**: SPI Trace	Greater than 15 MIPS PIC32: Device Dependent
	RI: Performance Pak + Isolator Unit (AC244005)	Basic and Advanced	None	Greater than 15 MIPS PIC32: Device Dependent
JTAG	**RI**: JTAG Adapter (AC244007), **SJL**	Basic	None	PIC32: Device Dependent
Logic Port	**RI**: Logic Port Probes (4 pins)	N/A	**RI**: I/O Port Trace	Device Dependent
	RI:Trace Interface Kit (AC244006)	N/A	**RI**: Instruction Trace (PIC32 MCUs, some Emulation Headers)	Device Dependent
Standard (ICSP) Communications + Logic Port	**RI**: Power Monitor (AC244008)	Basic	None	15 MIPS or less PIC32: Device Dependent

1. For tool abbreviations, see last table.
2. Support is device dependent, i.e., some debug features are not available on all devices.
3. *Basic Debug Support*: Run, Halt, Reset, Step Into, Step Over, Software/Hardware Breakpoints.
4. *Advanced Debug Support*: Data Capture, Runtime Watch, etc.

Table 6-5. Hardware Tool to Target Connections - AVR MCUs

Connection	Hardware Support[1,2]	Device Support[3]	Debug Support[4,5,6]	Trace Support	Support Speed
JTAG	AI, PK4	All AVR MCUs	Basic	No	32kHz to 7.5MHz
aWire	AI	AVR 32-bit MCUs	Basic	No	7.5kbit/s to 7Mbit/s
PDI 2-wire	AI, PK4	AVR XMEGA® MCUs	Basic	No	32kHz to 7.5MHz
debugWIRE	AI, PK4	AVR 8-bit MCUs	Basic	No	4kbit/s to 0.5Mbit/s
UPDI	AI, PK4	AVR 8-bit MCUs	Basics	No	Up to 750kbit/s
SPI	AI, PK4	AVR 8-bit MCUs	Programming only	N/A	8kHz to 5MHz
TPI	AI, PK4	tinyAVR 8-bit MCUs	Programming only	N/A	Device dependend

.....continued

Connection	Hardware Support(1,2)	Device Support(3)	Debug Support(4,5,6)	Trace Support	Support Speed

1. For tool abbreviations, see last table.
2. ICD4 and PK4 require the AC102015 Debugger Adapter Board.
3. Device support is on-going and may not be currently available.
4. Support is device dependent, i.e., some debug features are not available on all devices.
5. *Basic Debug Support*: Run, Halt, Software/Hardware Breakpoints.
6. *Programming Only*: No debugging with this connection.

Table 6-6. Hardware Tool to Target Connections - SAM MCUs

Connection	Hardware Support(1,2)	Device Support(3)	Debug Support(4,5,6)	Trace Support	Support Speed
JTAG	ICD4	SAM MCUs	Basic	No	32kHz to 7.5MHz
SWD	ICD4	SAM MCUs	Basic and Advanced	ITM - 3MB/s	32kHz to 10MHz

1. For tool abbreviations, see last table.
2. ICD4 and PK4 require the AC102015 Debugger Adapter Board.
3. Device support is on-going and may not be currently available.
4. Support is device dependent, i.e., some debug features are not available on all devices.
5. Basic Debug Support: Run, Halt, Software/Hardware Breakpoints.
6. Advanced Debug Support: Additional features over Basic; device dependent.

Table 6-7. Tool Abbreviations

Abbreviation	Tool Name
Snap	MPLAB Snap in-circuit debugger/production programmer
PK4	MPLAB PICkit 4 in-circuit debugger/production programmer
PK3	PICkit 3 in-circuit debugger/development programmer
ICD4	MPLAB ICD 4 in-circuit debugger/production programmer
ICD3	MPLAB ICD 3 in-circuit debugger/production programmer
RI	MPLAB REAL ICE in-circuit emulator/production programmer
AI	Atmel-ICE in-circuit debugger/programmer
SJL	SEGGER J-Link debug probes (Third Party)

For more on device-dependent debug features, see:

`<MPLAB X IDE install>docs/FeatureSupport/HWToolDebugFeatures.html`

6.13 Use IDE Scripting

MPLAB X IDE provides scripting capabilities to control the IDE behavior programmatically. Allow developed for use with Microchip MPU projects, scripts may be used with MCU projects as well.

About the Scripts

The scripts are written in Jython. The scripts have access to an API that allows you to:

- Control the debugging session state (run, halt, etc.).

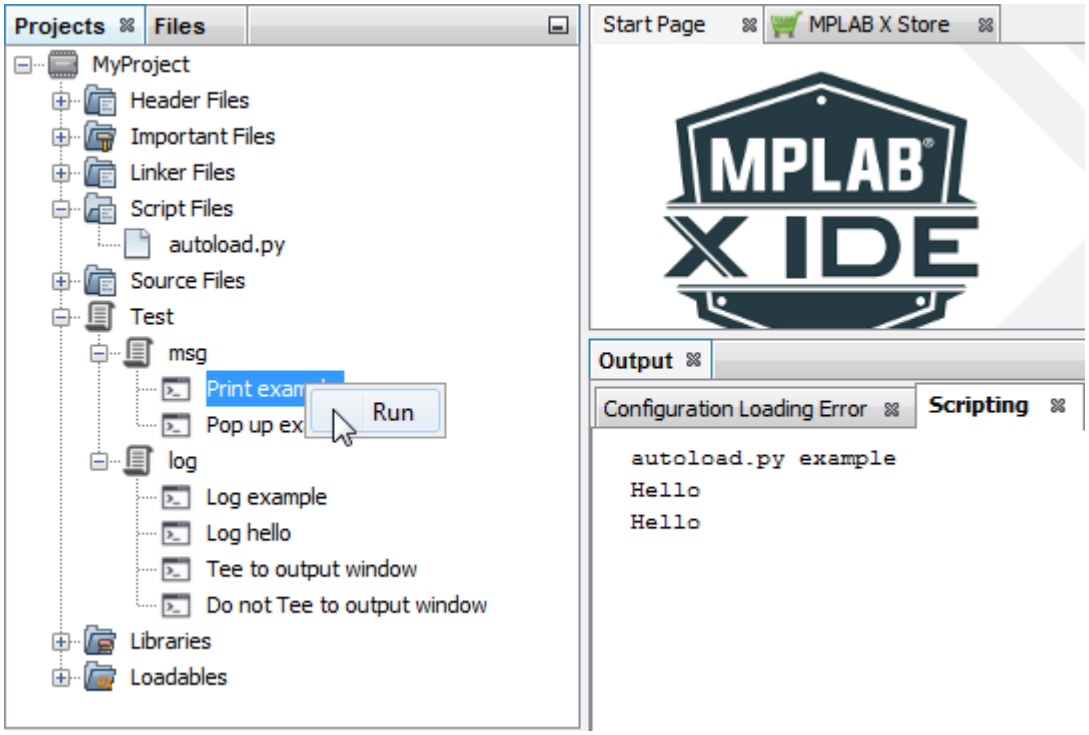
- Access memory in the device while debugging.
- Set breakpoints with callbacks to functions that determine if a halt should happen or not. This allows implementation of conditional breakpoints.
- Place actions in the Projects window which, when clicked, will run methods in a script.
- Provide hooks for debug events: a function will be called when a given event happens.

Using the Scripts

To gain access to the scripting features, follow the steps below.

1. Close the project that will use the scripts.
2. Create a file named `autoload.py`. A self-documenting `autoload.py` that explains the APIs is available under `<MPLAB X IDE Install folder>\docs\example_code`.
3. Copy the file into your project directory. Edit the file here as necessary.
4. Re-open the project and you will see a "Script Files" folder in the project tree. The file `autoload.py` will be under this folder.
5. Run the scripts under "Tests" as needed.

Figure 6-20. Unedited autoload.py in Project



6.14 Checksums

A checksum is a hexadecimal number derived from the application code and other settings of an MCU or DSC. Refer to your device programming specification for details on how the checksum is generated.

A checksum is created when a project is built. You can view the generated checksum in the Dashboard window: 5.19 View the Dashboard Display

Checksum: 0x339C

Checksums are used to detect errors between builds. For example, consider a project built on one computer with one checksum. If that project is copied to another computer and built there, an identical checksum means the project was successfully copied while a different checksum means the copying failed.

Care should be taken when developing project code if the checksum is used for error detection. Do not use the compiler macros `__TIME__` or `__DATE__`. Also, do no use the macros `__FILE__` or `assert()` if file paths will change.

For some device, the checksum may be used as the user ID: 4.10.3 Insert unprotected checksum in user ID memory.

6.15 Configurations

The term "configuration" is used in many different contexts. The following table presents the terms, and definitions of them.

Table 6-8. Configurations Context

Term	Definition
Configuration Bits Configuration Fuses Configuration Words Configuration Bytes Configuration Registers	All of these terms refer to the physical section of device memory used to configure the device. See your device data sheet for available configuration settings.
Configuration Bits Window	The MPLAB X IDE window which displays the configuration settings available for the project device. For more on this window, see 4.19 Set Configuration Values in the Configuration Bits Window.
Project Configuration	Build settings for the project. When a project is created, the configuration is "[default]." This name may be changed and multiple configurations may be created for the project. For more information, see 6.4 Work with Multiple Configurations. This data is stored in the file `configurations.xml`.
Project Configuration Type	Settings for a specific type of project. This applies only to Standalone projects configured as "application" and Library projects configured as "library." For details, see 4.10.1 Change Project Configuration Type.
User Configuration Data	MPLAB X IDE configuration settings created by the user. See 8.7 Viewing User Configuration Data.

7. Editor

MPLAB X IDE is built upon the Netbeans platform, which provides a built-in editor to create new code or change existing code.

For information on the NetBeans Source Editor, see:

https://docs.oracle.com/cd/E50453_01/doc.80/e50452/working_nbeans.htm#A1151635

C compiler information regarding the editor is available under the NetBeans help topic:

https://netbeans.org/kb/docs/cnd/navigating-editing.html

7.1 Editor Usage

To begin using the Editor, create or open an existing file using *File>New File* or *File>Open File*, respectively. Additional controls and features are described in the following sections.

Desktop Controls

The following desktop items are associated with the Editor:

- File menu – opens a file in an Editor window (see 10.1.1 File Menu).
- Edit menu – accesses edit commands (see 10.1.2 Edit Menu).
- Editor toolbar – also accesses edit commands – located at the top of each file's Editor window (see 10.2.10 Editor Toolbar).
- Window right-click (context) menu – for additional commands.

Display Controls

Basic display controls are shown in the table below. Additional formatting control may be found under 7.2 Editor Options.

Action	Description
Maximize the Source Editor.	Do one of the following: • Double click a file's tab in the Source Editor. • Make sure that the Source Editor window has focus and then press Shift-Escape. • Choose *Window>Configure Window>Maximize*.
Revert a maximized Source Editor to its previous size.	Do one of the following: • Double click a file's tab in the Source Editor. • Press <Shift> + <Escape>. • Choose *Window>Configure Window>Restore*.
Display line numbers.	Choose *View>Show Line Numbers*.
View two files simultaneously.	1. Open two or more files. 2. Click the tab of one of the files and drag it to the side of the window where you want the file to be placed. When a red preview box appears to show you where the window will be placed, release the mouse button to drop the window. The window can be split horizontally or vertically, depending on where you drag the tab.
Split the view of a single file.	1. Right click the document's tab in the Source Editor and choose "Clone Document." 2. Click the tab of the cloned document and drag it to the part of the window where you want the copy to be placed.

.....continued

Action	Description
Move between open files	1. Press and hold <Ctrl>. 2. Press <Tab>. A selection box will appear. 3. Select the file you want to move to.
Format code automatically.	Right click in the Source Editor and choose "Format." If any text is selected, only that text will be formatted. If no text is selected, then the whole file is formatted.

Basic Editor Features

The following table summarizes some of the more frequently-used features of the editor.

Editor Feature	Reference
Unicode is supported.	By default, newly-created projects in the IDE use ISO-8859-1 character encoding. To change this: Right click the project name in the Projects window and choose Properties. In the left column under "Categories," select "General." On the bottom of the page, find "Encoding" and change.
Code is colored based on syntax.	*Tools>Options* (*MPLAB X IDE>Preferences* for macOS), **Fonts and Colors** button, **Syntax** tab. Based on Encoding set during Project creation.
Errors are flagged as code is typed.	See the table of contents for the Netbeans Help topic: https://netbeans.org/kb/docs/java/editor-codereference.html#coloring
Colored markers provide quick access to multiple symbols, errors, etc.	*Tools>Options* (*MPLAB X IDE>Preferences* for macOS), **Fonts and Colors** button, **Annotations** tab
Smart code completion makes suggestions and provides hints.	*Tools>Options* (*MPLAB X IDE>Preferences* for macOS), **Editor** button, **Code Completion** tab
Mouseover symbols in debug mode to see values.	On halt when debugging, mouse over symbols to see values. **Note:** You cannot view the value of macros in this way. Use the Macro Expansion Window by right clicking (or Cmd+Alt+Click for macOS) on the macro in the editor during debug.
Assembly and C code may be collapsed and expanded	See 7.5 Code Folding.
Right clicking on a function (`delay(x)`) finds usages. This can limit find within a function (e.g., a local `i` variable).	See the Netbeans Help topic: http://wiki.netbeans.org/Find_Usages_in_Compiled_Dependencies
Right clicking on a function (`delay(x)`) shows a call graph The call graph has buttons on the side to switch order, etc.	See "View The Call Graph."
Create comments with keywords (example: `//TODO`) in the source code and action items will be scanned and added automatically to the Action Items window.	See the Netbeans Help topic: https://netbeans.org/features/ide/index.html regarding the Options Window, Team: Action Items.

.....continued	
Editor Feature	**Reference**
File history is available to view recent changes and revert, even without a version control system.	5.22 Control Source Code using Local History
Navigation is simplified with items such as "Go to file," "Go to type," "Go to symbol," "Go to header," and "Go to declaration."	See "Navigate Menu."
Refactor options (rename functions and variables, find all functions, etc.) for improving code structure.	See 7.6 C Code Refactoring.

Editor Troubleshooting

For errors in code highlighted in the editor, see your compiler documentation. For information on error highlighting, see the NetBeans help topic:

https://netbeans.org/kb/docs/java/editor-codereference.html#coloring

Specific errors may be found under 9.5 Errors.

7.2 Editor Options

To set editor options:

1. Select *Tools>Options* (*MPLAB X IDE>Preferences* for macOS) to open the Options dialog.
2. Click on the **Editor** button. Then click on a tab to set up editor features.

Each tab and its options are listed below.

Table 7-1. General Tab

Item	Description
Code Folding	Check to enable Code Folding and select which types of code to fold. For more on Code Folding, see 7.5 Code Folding. See also the NetBeans help topic: https://netbeans.org/kb/docs/java/editor-codereference.html
Camel Case Behavior	Check to enable camel case navigation.

Table 7-2. Formatting Tab

Item	Description
Language	Select the programming language to which the formatting will apply.
Category	Select a category, currently only "Tabs and Indents."
Expand Tabs to Spaces	Check to make tabs into spaces.
Number of Spaces per Indent	Enter the equivalent number of spaces per indent.
Tab Size	Enter the size of a tab.
Right Margin	Enter the size of the right margin.

Table 7-3. Code Completion Tab

Item	Description
Language	Select the programming language to which code completion will apply.
Completion options	Check to enable code completion options. For more on code completion, see the NetBeans help topic: https://netbeans.org/kb/docs/java/editor-codereference.html

Table 7-4. Code Templates Tab

Item	Description
Language	Select the programming language to which the template will apply.
Templates	Enter template information for the language specified above. ***Abbreviation***: Enter an abbreviation to type into the editor. ***Expanded Text***: After typing the abbreviation and the "Expand template on" character(s), expand the abbreviation to this text in the editor. ***Description***: Add an optional description of the template item. For more on code templates, see the NetBeans help topic: https://netbeans.org/kb/docs/java/editor-codereference.html
Expand template on	Select character(s) that will be typed into the editor to expand the abbreviated text to the expanded text.

Table 7-5. Hints Tab

Item	Description
Language	Select the programming language to which the hints will apply.
Hint Window	In the Hint window, select an occurrence and specify the "hint" you wish to receive in the event of the occurrence: Error, Warning or Warning on current line. For more on using hints, see the NetBeans help topic: https://netbeans.org/kb/docs/java/editor-codereference.html

Table 7-6. Macros Tab

Item	Description
Macros	Add, delete and define editor macros. ***To create a new macro***, click the **New** button, enter the name for the new macro, then select the new macro in the list and enter the code, in quotes, in the Macro Code editor. ***To set a keyboard shortcut for a macro***, select a macro from the list, click the **Set Shortcut** button and enter the shortcut in the dialog box. Use this shortcut to run your macro. ***To remove a macro***, select a macro from the list and click the **Remove** button. ***To modify the macro code***, select a macro from the list and edit the code in the Macro Code editor.
Macro Code	Click on the macro name above and then type in your macro code in quotes. For a list of macro keywords, see: http://wiki.netbeans.org/FaqEditorMacros **Note:** Generally, it is easier to add a new macro by recording it than by adding one manually in the Macro Code editor. Use *Edit>Start/Stop Macro Recording*.

7.3 Hyperlinks in Code

Hyperlinks in code help you to navigate to information relevant to the link content.

Hyperlinks in C Code

Hyperlink navigation lets you jump from the invocation of a function, variable, or constant to its declaration.

To use a hyperlink, do one of the following:

- Mouse over a function, variable, or constant while pressing <Ctrl> (Windows and Linux) or <Command> (Mac). A hyperlink appears, along with a tooltip with information about the element. Click the hyperlink and the editor jumps to the declaration. Press <Alt> + <Left Arrow> to jump back to the invocation.
- Mouse over an identifier and press <Ctrl> + <B> or <Command> + <B>. The editor jumps to the declaration. Press <Alt> + <Left Arrow> to jump back to the invocation.

Press <Alt> + <Left Arrow> and <Alt> + <Right Arrow> to move backward and forward through the history of the cursor position.

Hyperlinks in ASM Code

To navigate to header files included in assembly source files, press <Ctrl> (Windows and Linux) or <Command> (Mac) while placing the mouse cursor over the file name referenced by a `#include` statement. Then click the mouse select button to open the include file in its own file tab in the editor.

7.4 Editor Red Bangs

The Source Editor highlights syntax and semantic errors, such as missing semicolons and unresolved identifiers, in your source code. Errors are shown in red underline and a red bang "glyph"  is displayed in the left margin of the editor window next to each line that contains an error. An error message is displayed when you mouse over the glyph.

Note: Not all errors in the editor are actual errors, but may be unrecognized symbols. Try building your code to determine if it will build anyway. For more details, see the section "Errors Caused by Unrecognized Symbols."

Error Conditions and Additional Marking

Errors may occur when you do one of the following:

- Type in new code
- Edit existing code
- Add an existing file to your project
- Move a file to a different location in your project
- Change the properties of the project or one of its source files

If there are any errors in a file, red marks are displayed in the error stripe on the right side of the editor window. The error stripe represents the whole file, not just the lines currently displayed. Double click a mark in the error stripe to jump to the line the mark represents.

Errors Caused by Unrecognized Symbols

A symbol that is unrecognized by a compiler preparser or parser could be marked as an error but may not actually be an error (the code will build). Therefore, if you suspect this type of error, try to build the project anyway.

For examples, see the following issued under 9.5 Errors:

- For 8-bit code, #asm and #endasm cause red bangs in the Editor window.
- For 16-bit code, MPLAB XC16 packed attribute statement causes red bangs in the Editor window.

7.5 Code Folding

Code folding allows you to hide some sections of code so you can focus on others. Sections of C or assembly code may be collapsed (hidden) or expanded (displayed), depending on your selected options.

Code folding is enabled by default. For most C or assembly code, sections of code that may be folded are signified by "-" and "+" icons to the left of the code.

Enable and Set Up Code Folding

Use the following steps to enable and set up code folding:

1. Select *Tools>Options* (*MPLAB X IDE>Preferences* for macOS) to open the Options dialog.
2. Click on the **Editor** button.
3. Click on the **General** tab and check "Use code folding."
4. Check other options to specify sections to fold. If you do not see the type of code you want to fold listed there, you can do a custom fold.

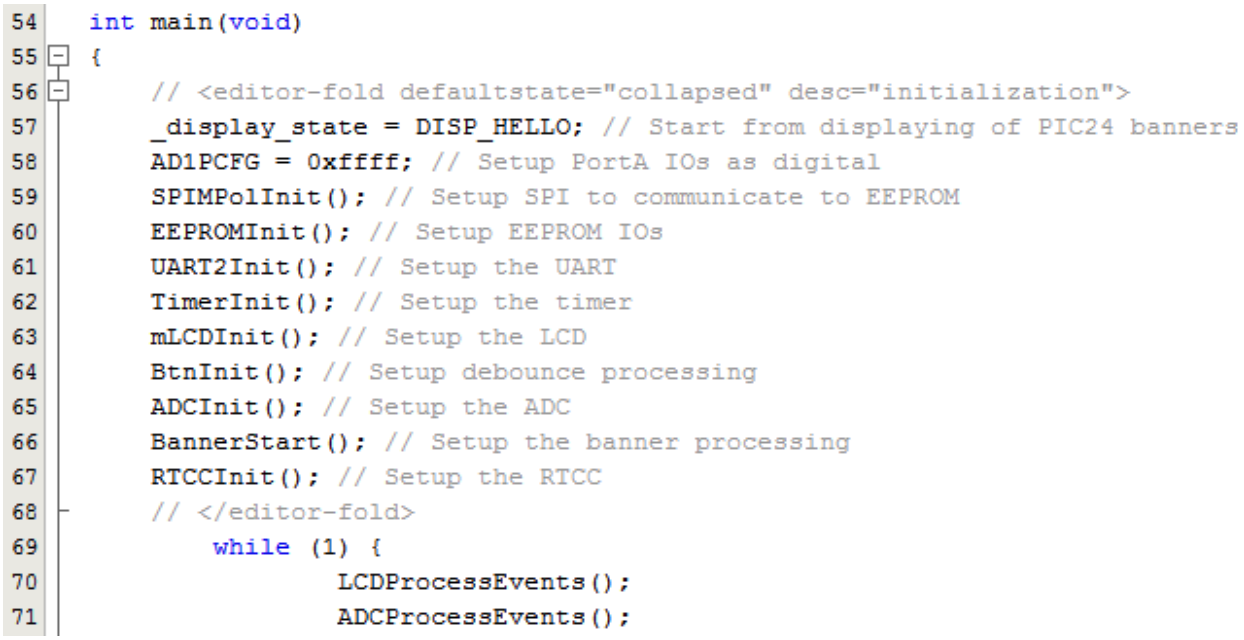
Expand/Collapse Code Folding

Within the editor, click the "-" icon next to a method and it folds. Click the "+" icon next to a folded method and it expands. Folded code is denoted by an ellipsis box. Hold the cursor over the ellipsis box and the IDE displays the collapsed method.

Figure 7-1. Expanded Code

```
54   int main(void)
55   {
56       // <editor-fold defaultstate="collapsed" desc="initialization">
57       _display_state = DISP_HELLO; // Start from displaying of PIC24 banners
58       AD1PCFG = 0xffff; // Setup PortA IOs as digital
59       SPIMPolInit(); // Setup SPI to communicate to EEPROM
60       EEPROMInit(); // Setup EEPROM IOs
61       UART2Init(); // Setup the UART
62       TimerInit(); // Setup the timer
63       mLCDInit(); // Setup the LCD
64       BtnInit(); // Setup debounce processing
65       ADCInit(); // Setup the ADC
66       BannerStart(); // Setup the banner processing
67       RTCCInit(); // Setup the RTCC
68       // </editor-fold>
69           while (1) {
70                   LCDProcessEvents();
71                   ADCProcessEvents();
```

Figure 7-2. Collapsed Code



Code Folding – Inline Assembly and MPLAB C18 C

Code folding does not work for inline assembly code in MPLAB C18 when the `_asm` and `_endasm` directives are used. The work-around is to put the code block near the end of the code/file.

Custom Code Folding – C

To custom fold C code, do one of the following:

- Type the following comments around your code:

```
// <editor-fold defaultstate="collapsed" desc="user-description">
C code block to fold
// </editor-fold>
```

OR

- Type `fcom` and press the <Tab> key to automatically enter the comment text above.

After the comment has been entered, customize it for your code:

`defaultstate`	Enter either `collapsed` or `expanded`,
`desc`	Enter a description of the code. Once collapsed, this description can still be seen for reference.

An example of custom code folding is shown in "Expand/Collapse Code Folding" above.

For additional information see the NetBeans help topic:

https://ui.netbeans.org/docs/ui/code_folding/cf_uispec.html

Custom Code Folding – ASM

To custom fold assembly code, do the following:

- For MPASM assembler or MPLAB XC16 assembler code, type the following comments around your code:

```
; <editor-fold defaultstate="collapsed" desc="user-description">
ASM code block to fold
; </editor-fold>
```

- For MPLAB XC32 assembler code, type the following comments around your code:

```
// <editor-fold defaultstate="collapsed" desc="user-description">
ASM code block to fold
// </editor-fold>
```

OR

- For MPASM assembler or MPLAB XC16 assembler code, type `fcom`; and press the <Tab> key to automatically enter the relevant comment text above.
- For MPLAB XC32 assembler code, type `fcom//` and press the <Tab> key to automatically enter the relevant comment text above.

Assembly code will fold in the same way that C code folds. For an example of code folding, see the C code example above.

Expand/Collapse Custom Code Folding

To expand/collapse code blocks, go to the View menu, or right-click in the Editor window, and select one of the following:

1. *Code Folds>Collapse Fold* to hide the block of code.
2. *Code Folds>Expand Fold* to display the block of code.
3. *Code Folds>Collapse All* to hide all folding blocks of code.
4. *Code Folds>Expand All* to display all folding blocks of code.

An example of custom code folding is shown in "Expand/Collapse Code Folding" above.

7.6 C Code Refactoring

Refactoring is the use of small transformations to restructure code without changing any program behavior. Just as you factor an expression to make it easier to understand or modify, you refactor code to make it easier to read, simpler to understand, and faster to update. And just as a refactored expression must produce the same result, the refactored program must be functionally equivalent with the original source.

Some common motivations for refactoring code include:

- Making the code easier to change or easier to add a new feature
- Reducing complexity for better understanding
- Removing unnecessary repetition
- Enabling use of the code for other needs or more general needs
- Improving the performance of your code

The IDE's refactoring features simplify code restructuring by evaluating the changes that you want to make, showing you the parts of your application that are affected, and making all necessary changes to your code. For example, if you use the Rename operation to change a class name, the IDE finds every usage of that name in your code and offers to change each occurrence of that name for you.

Refactor Menu

When you use the IDE's refactoring operations, you can change the structure of your code and have the rest of your code updated to reflect the changes you have made.

Refactoring operations are available on the Refactor menu (see 10.1.6 Refactor Menu).

Undoing/Redoing Refactoring Changes

You can undo (or redo) any changes that you made using the commands in the Refactor menu by using the buttons on the Undo/Redo toolbar menu.

	**Undo** - The IDE rolls back all the changes in all the files that were affected by the refactoring.

	**Redo** - The IDE repeats the refactoring changes in all the files that were affected.

If any of the affected files have been modified since the refactoring took place, the refactoring Undo/Redo is not available.

Finding Function Usages

You can use the Find Usages command to determine each place a function is used in your project's source code.

To find where a function is used in your project:

1. In the Navigator window or the Source Editor window, right click the function name and choose Find Usages - <Alt>+<F7> (or <Ctrl>+<F7> for macOS).
2. The Find Usages command displays the code lines that call the function. In the Find Usages dialog box, click Find.
   The Usages window displays the file name and the line of code for each usage found in that file.

To jump to a specific occurrence of the function, do one of the following actions:

- In the Usages window, use the up and down arrow buttons in the left pane to navigate from one occurrence of the function to the next.
- Double click a line of code to open the file and position the cursor on that particular line of code.

Additional IDE Find Mechanisms

The other IDE tools that enable you to search for all the places where specific text is used in a project include:

- ***Finding and Replacing Text***. Searches for all the places where specific text is used in a source file that is open in the Editor. Choose *Edit>Find* to open the Find dialog box, or choose *Edit>Replace* to open the Replace dialog box. These commands finds all matching strings, regardless of whether the string is a C code element.
- ***Find in Projects***. As with the Find command, the Find in Projects command searches for matching strings, regardless of whether the string is a function name. Choose *Edit>Find in Projects* to open the "Find in Projects" dialog box and then type the string of text that you are looking for.

To find where a function is declared in a source file, you can double click the function in the Navigator window. If the function is declared in a different source file, right click the function and choose *Navigate>Go To Declaration* from the contextual menu.

Renaming a Function or Parameter

To rename a function or parameter and update references to it throughout your project:

1. In the Source Editor, right click the function or parameter and choose *Refactor>Rename* from the context menu.
2. In the Rename dialog box, type the New Name.
3. Optionally, click **Preview**. In the Refactoring window, at the bottom of the Source Editor, review the lines of code that will be changed and clear the check box of any code that you do not want changed.
4. Click **Do Refactoring** to apply the selected changes.

For quick in-place renaming, place the cursor in the item that you want to rename and press Ctrl-R. Then type the new name. To finish renaming, press **Escape**.

Moving, Copying and Safely Deleting C Code

These functions are specific to C++ code. See 10.1.6 Refactor Menu.

7.7 C/C++ Code Error Directive

When using a `#error` directive in code, it is best to enclose the statement in parentheses, as in:

```
(#error This line shown only when there is an error.)
```

Otherwise, the syntax parser honors the `#error` directive in the same manner as a C compiler and stops in its tracks. Therefore the text immediately following the directive will be grayed in the Editor to signify this.

8. Project Files and Folders

MPLAB X IDE has several windows, usually in the File Pane, for working with project files and folders, namely the Projects, Files, Favorites and Classes windows.

It is recommended to look at the other project file and folder related topics in this chapter:

- Path, File and Folder Name Restrictions
- Path, File and Folder Project Recommendations
- Viewing User Configuration Data
- Importing an MPLAB IDE v8 Project – Relative Paths
- Moving, Copying or Renaming a Project
- Deleting a Project

8.1 Projects Window View

The Projects window is the main entry point to your projects. It shows a logical view of important project contents. For more on this window, see 12.15 Projects Window.

You will, at very least, need to add source files to your project. MPLAB X IDE will find many default files for you (header files, linker scripts). You may add library and precompiled object files, as well as edited header and linker script files. Files that will not be included in the build may be placed under "Important Files."

Figure 8-1. Projects Window – Simple C Code Project

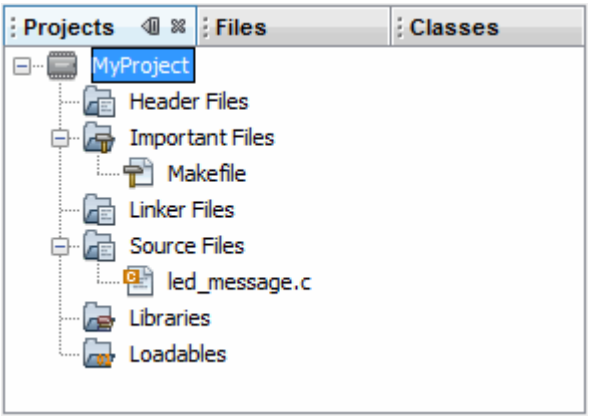


Table 8-1. Projects Window Definitions

Virtual Folder	Files Contained
Header Files	Header files (`.h` or `.inc`)
Important Files	Important files, such as makefiles . Other documents can be placed here, such as data sheet PDFs.
Linker Files	Linker files (`.ld`, `.gld` or `.lkr`)
Source Files	Source files (`.c`, `.cpp` or `.asm`)
Libraries	Library files (`.a` or `.lib`)
Loadables	Precompiled object files (`.o`)

See also the NetBeans help topic. Select *Help>Help Contents* and in the Search tab enter "Projects Window C or C++." Find the first occurrence of "Projects Window" in the list.

8.2 Files Window View

The Files window allows you to view all of your project contents in a directory-based view, even files and folders not shown in the Projects window.

Figure 8-2. Files Window – Simple C Code Project

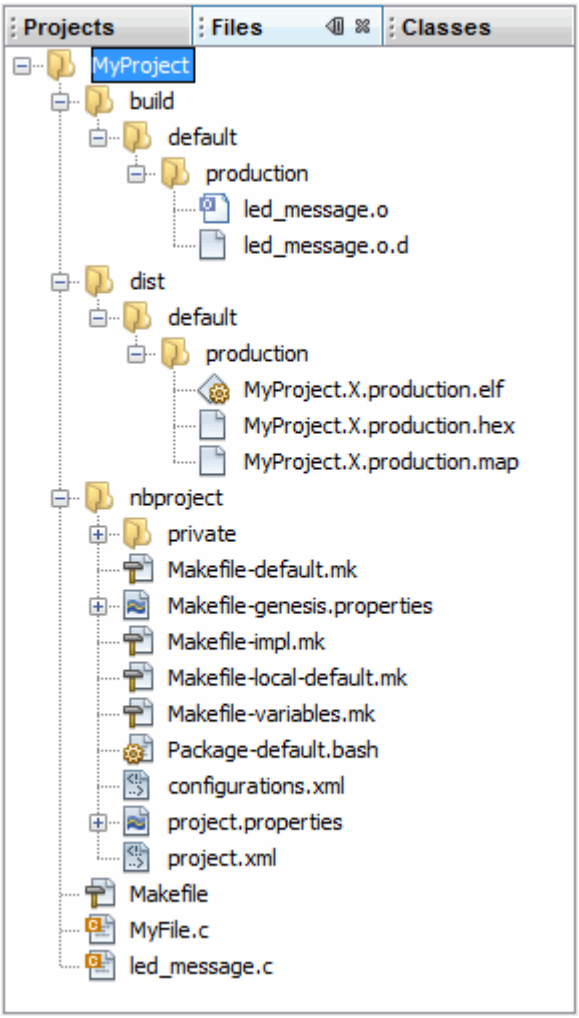


Table 8-2. Files Window Definitions

Folder	Description
MyProject	The project folder, which contains the Makefile file and C code or assembly application files. Makefile is the master makefile for the project. This file is generated at project creation time and it is not touched after that (it will not be regenerated). You can make changes to this file if you are familiar with GNU Make. But MPLAB X IDE provides ways to add a pre-step and post-step (Project Properties) which can be used instead of modifying the Makefile itself.
build(1)	The intermediate files folder. Files are contained in subfolders depending on project configuration, usage and location. The build files are: • Run files (`.o`) • Dependency files (`.o.d`) • HI-TECH® intermediate files (`.p1`)

.....continued

Folder	Description
dist(1)	The output files folder. Files are contained in subfolders depending on project configuration, usage and location. The distribution files are: • Executable files (`.hex`) • ELF or COF object files (`.elf` or `.cof`) • library files (`.a` or `.lib`)
nbproject	The makefile and metadata folder. Contains these files: • The private folder contains the user and computer specific settings of the project • Project makefile • Configuration-specific makefiles • `project.xml` IDE-generated metadata file • `configurations.xml` metadata file
default, MyConfig(2)	The project configurations folders. If no configurations are created by the developer, all code is in default.
production, debug(2)	The production and debug versions folders.
_ext(2)	The external project files folder. If a file is referenced outside the project folder, it is listed here.

1. You do not need to check these folders into source control. A build will create them. See also 5.23 Control Source Code using a Revision Control System.
2. These items are not shown in the figure above.

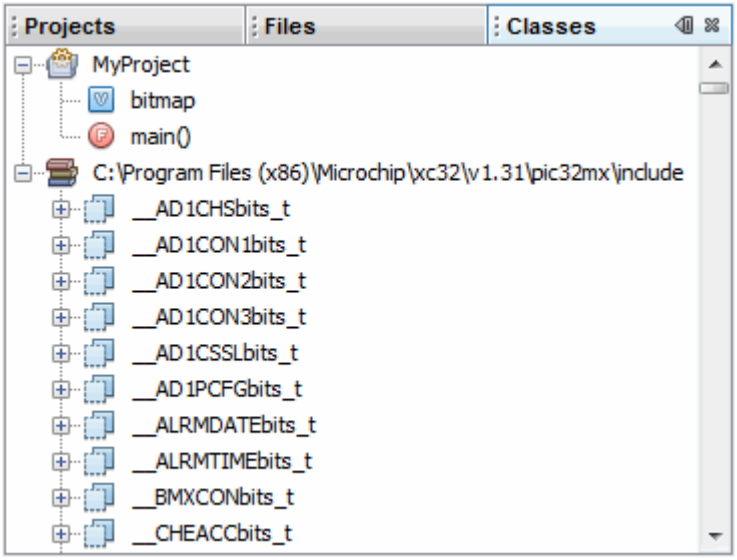
See also the NetBeans help topic. Select *Help>Help Contents* and in the Search tab enter "Files Window C and C++". Find the first occurrence of "Files Window" in the list.

8.3 Classes Window View

For C or C++ code, you can see all of the classes (functions), and the members and fields for each class, in the Classes window (*Window>Classes*).

See also the NetBeans help topic. Select *Help>Help Contents* and in the Search tab enter "Using the Classes Window."

Figure 8-3. Classes Window – Simple C Code Project



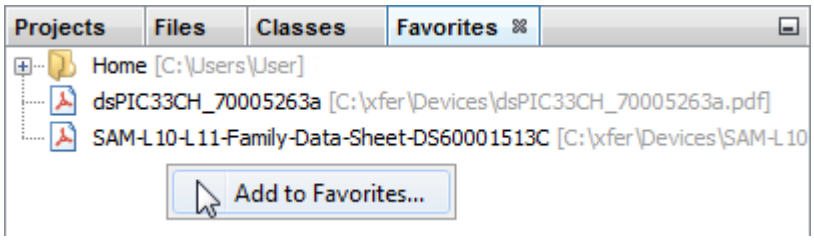
8.4 Favorites Window View

The Favorites window (*Window>Favorites*) enables you to access any file or folder on your computer or network, whether or not it is in a project.

When you first open the Favorites window, it only contains your computer's home directory.

- To add a file or folder, you can right click in the Favorites window and choose "Add to Favorites."
- To add a file, you can right click on the file name in the Projects window and select *Tools>Add to Favorites*.

Figure 8-4. Favorites Window with Examples



See also the NetBeans help topic. Select *Help>Help Contents* and in the Search tab enter "Favorites Window."

8.5 Path, File and Folder Name Restrictions

You should only use the following ASCII characters for path, file and folder names:

- A through Z
- a through z
- 0 through 9
- underscore: _
- dash: -
- period: .

Windows Operation System

Windows operating systems have a maximum path length of 260 characters. An explanation from Microsoft is found here:

https://msdn.microsoft.com/en-us/library/windows/desktop/aa365247(v=vs.85).aspx

Although Windows will attempt to stop you from creating path lengths that are too long, it is possible to do this (from certain applications, cutting-and-pasting, etc.) If a project source file uses such a path, the project will not build correctly.

Additionally, there is a command-line limitation for Windows OS of 8191 characters. For details, see:

Section "Command line too long (Windows OS)"

Cross-Platform Operating Systems

If you plan on using MPLAB X IDE on different platforms (Windows, Mac, or Linux operating systems), be aware of these issues:

- Use the forward slash "/" in relative paths. The backslash "\" works only on Windows OS platforms. Example: `#include headers/myheader.h`.
- Linux OS is case-sensitive, so `generictypedefs.h` is not the same as `GenericTypeDefs.h`.

8.6 Path, File and Folder Project Recommendations

When adding existing files or folders to a project, consider the name restrictions discussed in 8.5 Path, File and Folder Name Restrictions. Specifically, take care with long paths. Although there is a known limit for Windows OS (path length of 260 characters, command line of 8191 characters), a very long path on other systems may cause issues. If you have problems building, consider shortening long paths or moving files or folders into the project folder.

To ensure the most portable project, use relative paths when importing files or folders.

Related Links

4.9.3 Add Existing Files to a Project

6.11 Package an MPLAB X IDE Project

8.5 Path, File and Folder Name Restrictions

8.7 Viewing User Configuration Data

The MPLAB X IDE user directory (userdir) stores user configuration data such as window layouts, editor settings, menu and toolbar customizations, and various module settings such as the list of known compilers.

When you install plugins (*Tools>Plugins*), the plugin information (code) is stored in the userdir, not in the MPLAB X installation directory. However, you may change this location under *Tools>Plugins>Settings>Advanced* by selecting "Force install into shared directories" from the "Plugin Install Location" drop down menu.

To find out where your user directory is located, select *Help>About* and find the path next to **Userdir**. This information is created and managed by MPLAB X IDE.

8.8 Importing an MPLAB IDE v8 Project – Relative Paths

For an MPLAB IDE v8 project located at:

```
C:\MyProjects\mplab8project
```

On importing to MPLAB X IDE you will get:

```
C:\MyProjects\mplab8project\mplab8project.X
```

The MPLAB X IDE imported project is placed by default under the MPLAB IDE v8 project to preserve maintainability of both projects. However, you can place the MPLAB X IDE project folder anywhere. Also, the name of the project will be set as `mplab8project.X` but you can rename it if you wish. There is a convention to finish the name of an MPLAB X IDE project with `.X` but this is just a convention.

The new project will not copy the source files into its folder, but instead will reference the location of the files in the v8 folder. To create an independent MPLAB X IDE project, create a new project and copy the MPLAB IDE v8 source files to it.

For details, see 5.3 Import an Existing MPLAB IDE v8 Project.

8.9 Moving, Copying or Renaming a Project

After you have created a project, you may realize you need to make changes. Using the commands on the context (right click) project menu, you can move, copy or rename your project from within MPLAB X IDE. There is also an option to delete the project. For details, see "Project Menu" under 12.15 Projects Window.

You can also use an external tool; the project file structure does not require you to use MPLAB X IDE.

8.10 Deleting a Project

To delete a project in MPLAB X IDE:

1. In the Projects window, right click on the project name and select "Delete."
2. In the "Delete Project" dialog, you can delete:
   2.1. The project file from the computer - some source files will remain.
   2.2. The project file with all of the source files (via check box).

The project is deleted so that MPLAB X IDE will no longer see it.

Note: MPLAB X IDE does not release all resources assigned to a project when it closes. MPLAB X IDE runs on Java so releasing resources does not occur immediately. Under Java it is best to let the JRE decide when to call the garbage collection (GC) within the software code. However, you can force GC by enabling the memory toolbar and clicking on it.

9. Troubleshooting

This section is designed to help you troubleshoot any problems, errors or issues you encounter while using MPLAB X IDE. If none of this information helps you, see "Support" for ways to contact Microchip Technology.

9.1 USB Driver Installation Issues

To install the correct USB drivers, see 2.3 Install the USB Device Drivers (for Hardware Tools).

To troubleshoot errors, see 2.3.3.5 Tool Communication Issues.

9.2 Operating System Issues

Different operating systems have different issues concerning path, file and folder naming. For more information, see 8.5 Path, File and Folder Name Restrictions.

9.3 NetBeans Platform Issues

The NetBeans platform may have issues concerning the platform release used for MPLAB X IDE. For more help, visit these NetBeans web sites:

http://www.netbeans.org

http://netbeans.org/community/releases/index.html

The following issues have a known impact in MPLAB X IDE.

- Choosing "Inherited" as foreground color results in text for category being invisible in editor
- No visual indicators for debug
- Unable to resolve identifier
- SFR in Watch Window deselected when quickly stepping through code
- 9.3.5 Sometimes menus disappear when navigating submenus

9.3.1 Choosing "Inherited" as foreground color results in text for category being invisible in editor

When the Foreground color for Field is "Inherited" the text in popup completion window (tooltip) is invisible. The popup completion window shows unselected entries as white text on a white background. Scrolling down the list shows one entry at a time with white text on a blue background. Otherwise the feature seems to work normally.

This issue is recorded as a Netbeans Bugzilla issue for Fonts & Colors:

https://netbeans.org/bugzilla/show_bug.cgi?id=249766

Options->Fonts & Colors. Choosing "Inherited" option as foreground color for any [Syntax] category causes the text to be invisible in the editor. Looking at the preferences xml file "org-netbeans-modules-editor-settings-CustomFontsColors-tokenColorings.xml" it is clear that there is no linkage to a valid color for the foreground attribute of the affected category like "Field" or "Identifier."

Work-around:

1. Select *Tools>Options*, **Fonts & Colors** button, **Syntax** tab.
2. For "Language," choose "C."
3. Under "Category," click "Field."
4. To the left, see that "Foreground" is selected as "Inherited." Select a color from the list.
5. Click **OK**.

9.3.2 No visual indicators for debug

Although debugging appears to occur, there is no Pause arrow or line highlight, and breakpoints appear to be broken (broken icon in gutter). This can be caused by incorrect naming of file paths. See 8.5 Path, File and Folder Name Restrictions.

9.3.3 Unable to resolve identifier

Live parser shows unresolved identifier for structure nested in structure without name. This results from NetBeans native CND not supporting anonymous field declarations. The workaround is to specify an identifier for the field. The down side is that you will have to specify that field name everywhere you use it in code.

Example:

```
struct dados {
    unsigned char signature1;
    unsigned char signature2;
};
typedef union {
    unsigned char data [2];
    struct dados;         // unresolved ident.
} EEPROM_DATA;

void main(void) {
    EEPROM_DATA edata;
    edata.data [0] = 0xAA;    // this is ok
    edata.signature2 = 0xDD;  // unresolved ident.
}
```

See also:

https://netbeans.org/bugzilla/show_bug.cgi?id=256329

9.3.4 SFR in Watch Window deselected when quickly stepping through code

When you step, the cursor updates in the source so the focus is removed from the Watches window. Netbeans then re-draws the whole Watches window, which removes any highlighting.

9.3.5 Sometimes menus disappear when navigating submenus

For Windows OS, sometimes when an MPLAB X IDE menu is selected and then a submenu of a menu item is selected, the menu will disappear. You may need to restart MPLAB X IDE to fix the problem.

9.4 MPLAB X IDE Issues

The following list are issues with MPLAB X IDE. If you do not see your issue here, look at documentation related to your project's debug and compiler tool.

- Importing an MPLAB IDE v8 Project – Settings
- Importing an MPLAB IDE v8 Project – Modified Linker Scripts
- MPLAB C18 C Compiler and Structures
- Project is Empty/Project Files Grayed Out
- Code in Editor is gray after #error statement
- Internet Connection is not stable enough to download latest toolchain

9.4.1 Importing an MPLAB IDE v8 Project – Settings

Settings that were saved in a workspace in MPLAB IDE v8 (such as tool settings) will not be transferred to the new MPLAB X IDE project. Refer to the MPLAB IDE v8 help for what is stored in a workspace (*MPLAB IDE Reference>Operational Reference>Saved Information*).

9.4.2 Importing an MPLAB IDE v8 Project – Modified Linker Scripts

If you have modified your MPLAB IDE v8 project linker script so that it includes an object file, the linker will be unable to find the file when it is imported into MPLAB X IDE, because the build paths for MPLAB IDE v8 and MPLAB X IDE are different.

So you may see an error like:

```
<install path>ld.exe: cannot find file.o
```

since in MPLAB IDE v8 all build-related files are in one directory, whereas in MPLAB X IDE build files are in different subdirectories.

You can edit your linker script to work with MPLAB X IDE by using wild cards. For example, change:

```
/* Global-namespace object initialization - MPLAB v8*/
.init   :
{
KEEP (crti.o(.init))
:
} >kseg0_program_mem
```

to:

```
/* Global-namespace object initialization - MPLAB X*/
.init   :
{
KEEP (*crti.o(.init))
:
} >kseg0_program_mem
```

Alternatively, you can use an address attribute that allows you to place functions in C code.

```
int __attribute__((address(0x9D001000))) myfunction (void) {}
```

This allows you to place a function at an address without modifying the default linker script.

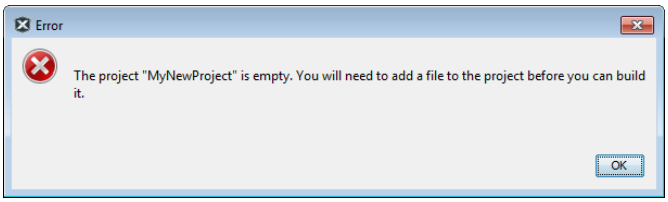
9.4.3 MPLAB C18 C Compiler and Structures

The pointer value of a structure in RAM memory will not be displayed correctly in the Watches window. This is because the compiler does not store complete debug information. The work-around is to use the MPLAB XC8 C compiler instead.

9.4.4 Project is Empty/Project Files Grayed Out

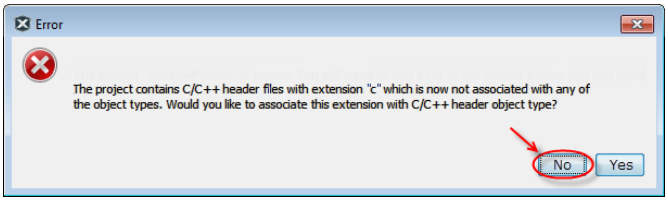
If you have loaded a corrupted project and answered a dialog question incorrectly, the Projects window could be grayed out and MPLAB X IDE will say the project cannot be built due to having no source files.

Figure 9-1. Project Empty Error



The problem occurs when the corrupted project is being loaded into MPLAB X IDE. A dialog will appear asking if C source files should be associated with header files. You can correct the issue at this point by clicking "No" (the default).

Figure 9-2. New Header Extension Error



However, if you say "Yes," the source files are now seen as header files and the project looks empty. To fix the project, you can use either of the following methods:

- Exit from MPLAB X IDE.

- Go to the MPLAB X IDE User Directory (specified in the *Help>About* dialog) and delete the directory for your version of MPLAB X IDE. For example, delete the folder `v3.05` from:
  `C:\Users\UserName\AppData\Roaming\mplab_ide\dev\v3.05`
- Launch MPLAB X IDE again.

OR

- Go to *Tools>Options*, Embedded button, **Other** tab.
- Remove 'c' from the header files association.
- Add 'c' to the source files association.
- Click **OK**.

9.4.5 Code in Editor is gray after #error statement

See 7.7 C/C++ Code Error Directive.

9.4.6 Internet Connection is not stable enough to download latest toolchain

The feature covered in the link below is not available because your internet connection is determined to not be stable.

4.1.8.2 Download Latest MPLAB XC Compilers

9.5 Errors

Errors can take many forms in the IDE, most commonly shown by icons in windows or messages in the Output window. Hovering over icons will pop up text that may explain the issue. For text messages, please refer to online help to search for the error.

Some common errors and resolutions for the IDE or Editor are listed in the sections below:

- Command line too long (Windows OS)
- Couldn't reserve space for Cygwin™ heap (Windows 7 or 8)
- Destination Path Too Long/The filename or extension is too long (Windows OS)
- The project ProjectName is empty
- For 8-bit code, #asm and #endasm cause red bangs in the Editor window
- For 16-bit code, MPLAB XC16 packed attribute statement causes red bangs in the Editor window
- Hexmate Conflict Report Address Error
- Programming Errors
- Programmer to Go Not Supported
- Mouseover shows "Not Recognized" in Editor
- Error Code in -10000 Range

9.5.1 Command line too long (Windows OS)

When a project's makefile is run, each command that is executed is passed to the local native shell. For Windows operating systems, there is a limit of 8191 characters. Therefore, for a project with hundreds of C files that link, the limit could be surpassed and this error will be displayed.

Linux OS and macOS can usually take very long command lines. If you want to find out the length limit for your system, you can type on the command line `getconf ARG_MAX`.

Response File Workaround

If you are using the MPLAB XC8 or MPLAB XC32 C compiler v1.01 and above, or MPLAB XC16 C compiler v1.22 and above, you may be able to use a response file when calling the linker to work around this issue.

- For MPLAB XC8, in the Project Properties* window, click in "Categories" on "XC8 linker." Under "Option categories" select "Additional Options." Select the check box next to "Use response file to link."
- For MPLAB XC16, in the Project Properties* window, click in "Categories" on "xc16-ld." Under "Option categories" select "General." Check the check box next to "Use response file to link."
- For MPLAB XC32, in the Project Properties* window, click in "Categories" on "xc32-ld." Under "Option categories" select "General." Check the check box next to "Use response file to link."

* To open the Project Properties window, see 4.3 Open Project Properties.

Library Workaround

For all other compilers, you can create an MPLAB X IDE library project and move some source files from your main project to the library project. Then add the library project to your main project.

To create a library project select *File>New Project*, click the "Microchip Embedded" category, and then select "Library Project" as the project.

To add the library to your main project, open the Project Properties window i.e., *File>Project Properties*), click in "Categories" under "Libraries," and then click the **Add Library Project** button.

Archive Checkbox Workaround

For MPLAB XC16 and MPLAB XC32, there is now a check box you can use when archiving long sets of files into libraries that might push the link line over the character limit. The "Break into multiple lines" option can be used to tell the compiler to break up the archive line into smaller lines to avoid this limitation.

Find this option in:

Project Properties window, "xc16-ar" or "xc32-ar" category, "General" option category, "Break into multiple lines" check box.

9.5.2 Couldn't reserve space for MinGW heap

MPLAB X IDE uses the MinGW in its make process. In Windows OS, you may have virtual memory allocation issues.

Windows 7 or 8

To change the Virtual Memory settings, click *Start>Control Panel*. Then click System and then *Security>System>Advanced Systems Setting or System Protection*. On the **Advanced** tab, click on the **Performance Settings** button. In this dialog click the **Advanced** tab and then click on the **Change** button. Enter a custom size value as follows:

- Initial Size (MB) = Currently Allocated (shown at the bottom)
- Maximum Size (MB) = Recommended (shown at the bottom)

Click **Set**, click **OK**, and reboot your personal computer.

Windows 10

See:

https://www.geeksinphoenix.com/blog/post/2016/05/10/how-to-manage-windows-10-virtual-memory.aspx

9.5.3 Destination Path Too Long/The filename or extension is too long (Windows OS)

Windows OS has a maximum path length of 260 characters. For details, see 8.5 Path, File and Folder Name Restrictions.

9.5.4 The project ProjectName is empty

See 9.4.4 Project is Empty/Project Files Grayed Out.

9.5.5 For 8-bit code, #asm and #endasm cause red bangs in the Editor window

The `#asm` and `#endasm` statements are valid constructs for declaring in-line assembly code within an MPLAB XC8 C program. But in the IDE Editor, the C preparser does not see these as valid directives and will display a red bang next to these statements. However, this code will still work.

If you want to eliminate the red bangs in the Editor, you can use the `asm()` statement. This is the preferred method of using in-line assembly for all MPLAB XC C compilers.

Example:

```
// Inline Code that causes error
    #asm
        movlw 0x20;
        nop;
```

```
        nop;
    #endasm
// Workaround for inline assembly - that will not cause error
// (Multi line asm code)
    asm("\
    movlw 0x20;\
    nop;\
    nop;");
// Workaround for inline assembly - that will not cause Error

// (Single line asm code)
    asm("movlw 0x20; nop; nop");
```

9.5.6 For 16-bit code, MPLAB XC16 packed attribute statement causes red bangs in the Editor window

The following statement is marked as an error by the compiler parser but will build:

```
unsigned long long Bits8 : 8 __attribute__ ((packed));
```

The parser rules for qualifiers allow for any order, but the expectation is that the qualifiers are all specified before the identifier. Therefore, use one of the statements below to not produce an error:

```
unsigned long long attribute ((packed)) Bits8 : 8;
```

OR

```
attribute ((packed)) unsigned long long Bits8 : 8;
```

9.5.7 Hexmate Conflict Report Address Error

When Hexmate generates a conflict report, the address of the conflict in the hex file is sometimes not the same as the address shown in a memory window (where the data resides in memory.) For example:

Table 9-1. Hex Address vs. Memory Address

Device Family	Address in Hex File	Address in Memory
Baseline	0x100	0x80*
Midrange	0x100	0x80*
PIC18 MCUs	0x100	0x100
16-bit Devices	0x100	0x80*
32-bit Devices	0x100	0x100
* For some devices, the address shown in the conflict report is twice as large as the address of the data in memory.		

9.5.8 Programming Errors

If you code protect a device and then attempt to program it, you will receive a programming failure error for an address region.

The "Erase Device Memory Main Project"  function can help to determine if this is the programming issue. This function is available as an optional icon that you can add to a toolbar (*View>Toolbars>Customize*). Alternately, use MPLAB IPE.

1. Ensure code protection is off in your code.
2. Erase the device the device memory, which will erase the configuration bit settings.
3. Reprogramming the device

If programming succeeds, then code protect was the issue.

9.5.9 Programmer to Go Not Supported

A device that has too many configuration (config) words currently cannot support Programmer to Go (PTG). PTG is expecting config words to be sent all at once, but MPLAB X IDE sends each config word one at a time for devices with a large amount of config words.

9.5.10 Mouseover shows "Not Recognized" in Editor

When using mouseovers to reveal values in the Editor, ensure:

1. You are in debug mode (*Debug>Debug Project*). You cannot see symbols values unless you are debugging.
2. You are not mousing over a macro. You cannot see the value of a macro this way. Use the Macro Expansion Window by right clicking on the macro and selecting *Navigate>View Macro Expansion*.

9.5.11 Error Codes in -10000 Range

Error codes in the -10000 range represents Microsoft error codes. To determine the Microsoft error code, make the number positive (-10121 becomes 10121) and subtract 10000 (10121 - 10000 = 121). Microsoft error codes can be found at:

https://msdn.microsoft.com/en-us/library/windows/desktop/ms681382(v=vs.85).aspx

9.6 Forums

If you do not see your issue here, check out our forums at:

http://www.microchip.com/forums/f238.aspx

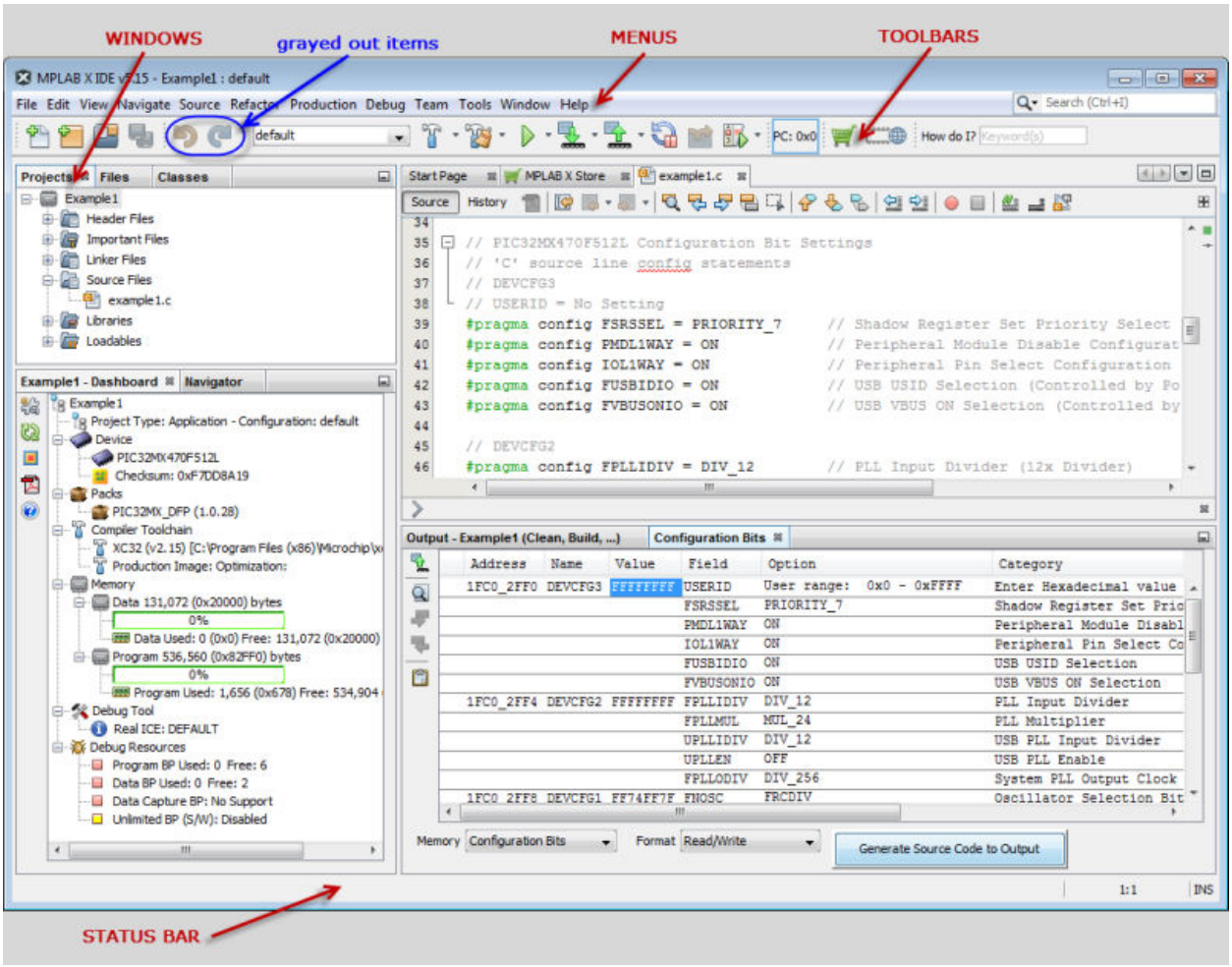
Here you will find discussions and recently posted solutions to problems.

10. Desktop Reference

The MPLAB® X IDE desktop is a resizable window that operates independent of the rest of the desktop items. The desktop consists of menus, toolbars, a status bar and tabbed windows. Menus, toolbars and the status bar are discussed here. Windows and dialogs are discussed in their own section.

Note: When the NetBeans help topic refers to a workspace, it is talking about a desktop. It is not referring to the MPLAB IDE v8 (and earlier) workspace.

Figure 10-1. MPLAB® X IDE Desktop



10.1 Menus

Many MPLAB® X IDE functions are accessible as menu items through the menu bar located across the top of the desktop. Menu items followed by ellipses (...) will open a dialog. For more on dialogs, see 12. MPLAB X IDE Windows and Dialogs.

Shortcut keys for menu items are listed next to the menu item.

Example: The shortcut keys for "New File" are Control-N (CTRL+N). More shortcut key information is available from *Help>Keyboard Shortcuts Card*.

Menu items may be grayed out for various reasons. See 10.4 Grayed Out or Missing Items and Buttons.

Additional context menus are available by right clicking in a window. For more on these menus, see 12.15 Projects Window.

Available menus are listed below.

Note: See the Start Page, My MPLAB IDE>Extend MPLAB>Selecting Simple or Full-Featured Menus to see all available menus and menu items. Not all of these items will be applicable to embedded development.

10.1.1 File Menu

This section lists the menu items in the File menu.

For keyboard shortcuts of some of these menu items, see *Help>Keyboard Shortcuts Card*.

Table 10-1. File Menu Options

Command	Action
New Project	Creates a new project with the New Project wizard.
New File	Creates a new file with the New File wizard.
Open Project	Opens an existing project.
Open Recent Project	Displays a list of all recently-opened projects for selection.
Import	Imports one of the following: • Hex/ELF... (Prebuilt) File – built with another tool. • MPLAB IDE v8 Project – launch Import Legacy Project wizard. • Other Embedded – import from another embedded platform. • From ZIP – import from a zipped project, i.e., unzip and import.
Close Project	Closes the current project.
Close Other Projects	Closes all projects except for the main project.
Close All Projects	Closes all open projects.
Open File	Opens an existing file.
Open Recent File	Displays a list of all recently-opened files for selection.
Project Group	Associates the current project with a group.
Project Properties	Opens the Project Properties window.
Save	Saves the current file.
Save As	Saves the current file under a new path and/or name.
Save All	Saves all open files. If the "Compile on Save" feature is selected, this will also compile/build project files.
Page Setup	Sets up the page for printing.
Print	Shows print preview of current file and allows printing.
Print to HTML	Prints the current file to a new file in HTML format.
Exit	Quits MPLAB® X IDE.

10.1.2 Edit Menu

This section lists the menu items in the Edit menu.

For keyboard shortcuts of some of these menu items, see *Help>Keyboard Shortcuts Card*.

Table 10-2. Edit Menu Options

Command	Action
Undo	Reverses (one at a time) a series of editor actions, except Save.
Redo	Reverses (one at a time) a series of Undo commands.
Cut	Deletes the current selection and places it on the clipboard.
Copy	Copies the current selection to the clipboard.
Paste	Pastes the contents of the clipboard into the insertion point.
Paste Formatted	Pastes the formatted contents of the clipboard into the insertion point.
Paste from History	Pastes from Copy history.
Delete	Deletes the current selection.
Select All	Selects everything in the current document or window.
Select Identifier	Selects the word nearest the cursor.
Find Selection	Finds instances of the current selection.
Find Next	Finds next instance of found text.
Find Previous	Finds previous instance of found text.
Find	Finds a text string.
Replace	Finds a string of text and replaces it with the string specified.
Find Usages	Finds usages and subtypes of selected code.
Find in Projects	Finds specified text, object names, object types within projects.
Replace in Projects	Replaces text, object names, object types within projects.
Start Macro Recording	Starts recording keystrokes.
Stop Macro Recording	Stops recording keystrokes . To manage macros, select *Tools>Options*, **Editor** button, **Macros** tab.

10.1.3 View Menu

This section lists the menu items in the View menu.

For keyboard shortcuts of some of these menu items, see *Help>Keyboard Shortcuts Card*.

Table 10-3. View Menu Options

Command	Action
Editors	Selects an available editor to view your files. • History – Show the history of edits to this file. • Source – Show the source code in this file.
Split	Splits the current editor window and its contents into two windows, either vertically or horizontally oriented.
Code Folds>Collapse Fold	If the insertion point is in a foldable section of text, collapses those lines into one line.
Code Folds>Expand Fold	If the currently selected line in the Source Editor represents several folded lines, expands the fold to show all of the lines.

.....continued

Command	Action
Code Folds>Collapse All	Collapses all foldable sections of text in the Source Editor.
Code Folds>Expand All	Expands all foldable sections of text in the Source Editor.
Code Folds>Collapse Fold Tree	Collapses all folds in a nested group of folds.
Code Folds>Expand Fold Tree	Expand all folds in a nested group of folds.
Web Browser	Opens the default web browser to the NetBeans home page.
IDE Log	Opens the log file in a tab of the Output window.
Toolbars>File, etc.	Shows the associated toolbar.
Toolbars>Small Toolbar Icons	Uses small icons on the toolbars.
Toolbars>Reset Toolbars	Resets to the default toolbar setup.
Toolbars>Customize	Customizes the items on an existing toolbar and allows creation of a new one.
Show Editor Toolbar	Shows the editor toolbar on the File tab.
Show Line Numbers	Shows line numbers in left margin.
Show Non-printable Characters	Shows characters even if they are not printable (non-ASCI).
Show Breadcrumbs	Shows breadcrumbs, or the path to the file from the project file, on the bottom of the editor window.
Show Indent Guide Lines	Shows faint lines to represent indents.
Show Diff Sidebar	Shows the DIFF sidebar.
Show Versioning Labels	Shows version labels.
Synchronize Editor with Views	Synchronizes the editor with open views.
Show Only Editor	Shows editor window in full IDE window.
Full Screen	Expands window to full length and breadth of screen.

10.1.4 Navigate Menu

This section lists the menu items in the Navigate menu.

For keyboard shortcuts of some of these menu items, see *Help>Keyboard Shortcuts Card*.

Table 10-4. Navigate Menu Options

Command	Action
Go to File	Finds and opens a specific file.
Go to Type	Finds and opens a specific class or interface.
Go to Symbol	Finds the symbol name as specified.
Go to Previous Window	Gives focus to the previously selected window.
Go to Source	Displays the source file containing the definition of the selected class.
Go to Declaration	Jumps to the declaration of the item under the cursor.
Go to Super Implementation	Jumps to the super implementation of the item under the cursor.
Last Edit Location	Scrolls the editor to the last place where editing occurred.
Back	Navigates back.

.....continued	
Command	**Action**
Forward	Navigates forward.
Go to Line	Jumps to the specified line.
Toggle Bookmark	Sets a bookmark on a line of code.
Bookmark History Popup Next	Goes to the next bookmark in the Bookmark window (*Window>IDE Tools>Bookmarks*).
Bookmark History Popup Previous	Goes to the previous bookmark in the Bookmark window (*Window>IDE Tools>Bookmarks*).
Next Error	Scrolls the Source Editor to the line that contains the next build error.
Previous Error	Scrolls the Source Editor to the line that contains the previous build error.
Select in Projects	Opens the Projects window and selects current document within.
Select in Files	Opens the Files window and selects current document within.
Select in Classes	Opens the Classes window and selects current document within.
Select in Favorites	Opens the Favorites window and selects current document within.

10.1.5 Source Menu

This section lists the menu items in the Source menu.

For keyboard shortcuts of some of these menu items, see *Help>Keyboard Shortcuts Card*.

Table 10-5. Source Menu Options

Command	Action
Format	Formats the selected code, or the entire file if nothing is selected.
Remove Trailing Spaces	Removes spaces at the end of the line.
Shift Left	Moves the selected line, or lines one tab to the left.
Shift Right	Moves the selected line, or lines one tab to the right.
Move Up	Moves the selected line, or lines one line up.
Move Down	Moves the selected line, or lines one line down.
Move Code Element Up	Moves the selected code element up one line.
Move Code Element Down	Moves the selected code element down one line.
Duplicate Up	Copies the selected line or lines one line up.
Duplicate Down	Copies the selected line or lines one line down.
Toggle Comment	Toggles the commenting out of the current line or selected lines.
Complete Code	Shows the code completion box.
Insert Code	Pops up a context aware menu that you can use to generate common structures such as constructors, getters, and setters.
Fix Code	Displays editor hints. The IDE informs you when a hint is available when the light bulb is displayed.
Show Method Parameters	Selects the next parameter. You must have a parameter selected (highlighted) for this shortcut to work.

.....continued

Command	Action
Show Documentation	Shows documentation for item under the cursor.
Insert Next Matching Word	Generates the next word used elsewhere in your code, as you type its beginning characters.
Insert Previous Matching Word	Generates the previous word used elsewhere in your code, as you type its beginning characters.
Inspect	Runs the specified inspection on the selected file, package, or project.
Scan for External Changes	Scans file for changes made outside of MPLAB® X IDE.

10.1.6 Refactor Menu

This section lists the menu items in the Refactor menu. The items you see are dependent on the type of object (variable, function, etc.) you are refactoring. For more information, see 7.6 C Code Refactoring.

For keyboard shortcuts of some of these menu items, see *Help>Keyboard Shortcuts Card*.

Table 10-6. Refactor Menu Options

Command	Action
Rename	Enables you to change the name of a variable or function to something more meaningful. In addition, it updates all source code in your project to reference the element by its new name.
Move	Moves a class to another package or into another class In addition, all source code in your project is updated to reference the class in its new location.
Copy	Copies a class to the same or a different package.
Safely Delete	Checks for references to a code element and then automatically deletes that element if no other code references it.
Change Function Parameter	Changes the amount and name of parameters for the selected function.
Encapsulate Fields	**C++ Only:** Click in a C++ source file to see this option. Automatically generates a getter method and a setter method for a field and optionally updates all referencing code to access the field using the getter and setter methods. See also: http://wiki.netbeans.org/Refactoring

10.1.7 Production Menu

This section lists the menu items in the Production menu (previously Run menu).

Note: The Run action has been removed from this menu, but it still available as an icon on the toolbar: 

For keyboard shortcuts of some of these menu items, see *Help>Keyboard Shortcuts Card*.

Table 10-7. Production Menu Options

Command	Action
Build Project	Builds all the files in a project.
Clean and Build Project	Removes (cleans) previously generated project files and then rebuilds the files in a project.

.....continued	
Command	**Action**
Batch Build Project	Builds multiple configurations of a project (only embedded available).
Set Project Configuration	Selects project configuration – Default.
Set Main Project	Sets the main project by selecting from a list of open projects.
Configuration Bits	Opens the Configuration Bits window.
Check File	Checks a file against a standard (XML related).
Validate File	Validates a file against a standard (XML related).
Repeat Build/Run	Runs again after halt.
Stop Build/Run	Ends run.

10.1.8 Debug Menu

Below are the menu items in the Debug menu.

Table 10-8. Debug Menu Options

Command	Action
Debug Project	Debugs the main or selected project.
Discrete Debugger Operation	Performs the following debug operations one step at a time (discretely): • Build for Debugging (build with debug executive). • Program Device for Debugging (program with debug build). • Launch Debugger. • Live Connect Debug. This is useful for changing the Memory window setting during debug and using starter kits.
Finish Debugger Session	Ends the debugging session.
Pause	Pauses debugging – use "Continue" to resume.
Continue	Resumes debugging after "Pause" until the next breakpoint or the end of the program is reached.
Step Over	Executes one source line of a program . If the line is a function call, executes the entire function and then stops.
Step Into	Executes one source line of a program. If the line is a function call, it executes the program up to the function's first statement and then stops.
Step Out	Executes one source line of a program. Finishes execution of the current function and stops on the source line immediately following the call to that function.
Step Instruction	Executes one machine instruction If the instruction is a function call, it executes the function and returns control to the caller.
Run to Cursor	Runs the current project to the cursor's location in the file and stop program execution.
Reset	Resets the device.
Set PC at cursor	Sets the program counter (PC) value to the line address of the cursor.

.....continued

Command	Action
Focus Cursor at PC	Moves the cursor to the current PC address and centers this address in the window.
Stack>Make Callee Current	Makes the method being called the current call . Only available when a call is selected in the Call Stack window.
Stack>Make Caller Current	Makes the calling method the current call. Only available when a call is selected in the Call Stack window.
Stack>Pop Topmost Call	Pops the call on top of the stack.
Stack>Pop To Current Stack Frame	Pops the current stack from to the top of the stack.
Stack>Pop Last Debugger Call	Pops the last call from the debug tool to the top of stack.
Toggle Line Breakpoint	Adds a line breakpoint or removes the breakpoint at the cursor location in the program.
New Breakpoint	Sets a new breakpoint at the specified line, exception, or method.
New Watch	Adds the specified symbol to watch.
New Run Time Watch	Adds the specified symbol to watch that will change value as the program runs/ executes.
Disconnect from Debug Tool	Disconnects communications between MPLAB® X IDE and the debug tool To reconnect, either Run or Debug.
Run Debugger/ Programmer Self Test	Performs a debug tool self-test For tools that support a self test, follow the tool documentation to set up the hardware and then run this test to confirm proper operation.
Hardware Tool Emergency Boot Firmware Recovery	Run utility to restore hardware tool boot firmware to its factory state. See MPLAB ICD 4 or PICkit 4 Help for details.

10.1.9 Team Menu

This section lists the menu items in the Team menu.

For keyboard shortcuts of some of these menu items, see *Help>Keyboard Shortcuts Card*.

Table 10-9. Team Menu Options

Command	Action
Shelve Changes	Temporarily sets aside some not yet committed changes in a working directory as a patch file.
Git, Mercurial, Subversion	Displays submenus that are specific to each version management system . Please see product documentation for more on the submenu options.
History	Shows the history of a file or reverts file to history version.
Find Task	Finds an issue in a version control system.
Report Task	Reports an issue to a version control system.
Create Build Job	Creates a build using a version control system.

10.1.10 Tools Menu

This section lists the menu items in the Tools menu.

For keyboard shortcuts of some of these menu items, see *Help>Keyboard Shortcuts Card*.

Table 10-10. Tools Menu Options

Command	Action
Embedded	Visible if a plugin has been added – select the plugin from the submenu.
Licenses	Manage compiler licenses. For details, see the Documentation tab of: https://www.microchip.com/mplab/compilers • Roam Network Licenses. • Activate Workstation License. • License Status. • Change Licensing Type.
Packs	Open MPLAB Pack Manager to select a versioned device pack for your project device.
Templates	Opens the Template Manager.
DTDs and XML Schemas	Opens the DTDs and XML Schemas Manager.
Plugins	Opens the Plugins Manager . For details, see the NetBeans help topic: http://wiki.netbeans.org/InstallingAPlugin
Plugins Download	Opens a selection dialog, where you can: • Visit "Embedded Code Source" Website - Download a plugin to install later using the "Downloaded" tab of the Plugin Manager. • Go to MPLAB X Plugin Manager - Install a plugin immediately from the "Available Plugins" list in the Plugin Manager.
Options	Opens the Options dialog. For macOS: Use MPLAB X IDE>Preferences. See 12.16 Tools Options Window, Embedded.

10.1.11 Window Menu

This section lists the menu items in the Window menu.

For keyboard shortcuts of some of these menu items, see *Help>Keyboard Shortcuts Card*.

Table 10-11. Window Menu Options

Command	Action
Projects	Opens the Projects window. See 8.1 Projects Window View.
Files	Opens the Files window. See 8.2 Files Window View.
Classes	Opens the Classes window. See 8.3 Classes Window View.
Favorites	Opens the Favorites window. See 8.4 Favorites Window View.
Services	Opens the Services window. See NetBeans help for more on this window.
Dashboard	Opens the Dashboard window. See 5.19 View the Dashboard Display.

.....continued

Command	Action
Navigator	Opens the Navigator window. See NetBeans help for more on this window.
Acton Items	Opens the Action Items window. See 12.1 Action Items Window.
Tasks	Opens the Task List window.
Output	Opens or moves to front the Output window. See 12.13 Output Window.
Editor	Opens an empty editor window. See 7.1 Editor Usage.
Debugging>Output	Opens debugging output windows.
Debugging>Variables	Opens the Local Variables debugger window. See 4.17 Watch Local Variable Values Change.
Debugging>Watches	Opens the Watches debugger window. See 4.16 Watch Symbol Values Change.
Debugging>Call Stack	Opens the Call Stack debugger window. See 5.17 View the Call Stack.
Debugging>Breakpoints	Opens the Breakpoints window. See 4.14 Control Program Execution with Breakpoints.
Debugging>Sessions	Opens the Sessions window.
Debugging>Sources	Opens the Sources window.
Debugging>Disassembly	Opens the Disassembly window. See 12.7 Disassembly Window.
Debugging>PIC AppIO	Opens the Application In/Out window . Applies to the MPLAB® REAL ICE™ in-circuit emulator.
Debugging>Trace	Opens the Trace window . Applies to the Simulator or MPLAB REAL ICE in-circuit emulator.
Debugging>Stopwatch	Opens the Stopwatch window. See 5.15 Use the Stopwatch.
Debugging>PC Profiling	Open the PC Profiling window. Applies to the MPLAB® REAL ICE™ in-circuit emulator.
Debugging>Triggers	Opens the Trigger In window. Applies to the MPLAB® REAL ICE™ in-circuit emulator.
Debugging>Debugger Console	Opens a console window for entering commands on the command line.
Web>Web Browser	Opens your default web browser.
Web>CSS	Opens the CSS Styles window. Enables you to edit the declarations of rules for HTML elements and selectors in a CSS file.
IDE Tools>Bookmarks	Opens the Bookmarks window. Displays a list of the bookmarks in your files in your open projects.

.....continued

Command	Action
IDE Tools>Notifications	Opens the Notifications window. Displays a list of the IDE errors and warnings that occurred in the current IDE session.
IDE Tools>Terminal	Opens a Terminal window for a local or remote host.
IDE Tools>Processes	Opens the Processes window. You can attach a debug tool to a running process.
PIC Memory Views>Memory	Opens specified Memory window. Memories shown depend on the project device. See either: 12.9 Memory Windows - 8- and 16-Bit Devices 12.10 Memory Windows - 32-Bit Devices
Simulator>Stimulus	Opens the Simulator Stimulus window. See MPLAB X Simulator documentation for details.
Simulator>Analyzer	Opens the Simulator Analyzer window. See MPLAB X Simulator documentation for details.
Simulator>IOPin	Opens the Simulator IO Pin window. See MPLAB X Simulator documentation for details.
Simulator>Register Trace	Opens the Simulator Register Trace window. See MPLAB X Simulator documentation for details.
Configure Window> Maximize	Maximizes the active window to the full IDE window size.
Configure Window>Float	Opens a floating IDE window with a tab.
Configure Window> Float Group	Opens a floating IDE window group with tabs.
Configure Window>Minimize	Minimizes the active window to standard size.
Configure Window> Minimize Group	Minimizes the window group to standard size.
Configure Window>Dock	Restores a floating tab to the main window.
Configure Window> Dock Group	Restores a floating tab group to the main window.
Configure Window> Clone Document	Opens a new tab for the same document.
Configure Window> Split Window	Splits the active document either vertically or horizontally.
Configure Window< New Document Tab Group	Splits the editor window into two groups of tabs and separates the selected tab into the new group.
Configure Window>Collapse Document Tab Group	Combines separate group of tabs into one group.
Reset Window	Resets windows to their default settings.
Close Window	Closes the current tab in the current window. If the window has no tabs, the whole window is closed.
Close All Documents	Closes all open documents in the Source Editor.
Close Other Documents	Closes all open documents except the active one.

.....continued	
Command	**Action**
Document Groups	Create named document group(s).
Documents	Opens the Documents dialog box, in which you can save and close groups of open documents.

10.1.12 Help Menu

This section lists the menu items in the Help menu.

For keyboard shortcuts of some of these menu items, see *Help>Keyboard Shortcuts Card*.

Table 10-12. Help Menu Options

Command	Action
Help Contents	Displays a JavaHelp viewer with all installed help files.
Tool Help Contents	Displays a single JavaHelp viewer with a single tool help file.
Release Notes	Open a list of links to available Release Notes for MPLAB supported tools. Other MPLAB X IDE support documentation links are also listed.
Online Docs and Support	Opens the NetBeans support web page.
Keyboard Shortcuts Card	Displays the keyboard shortcuts documen.t
MPLAB X Store	Opens the MPLAB X Store tab.
Start Page	Opens or moves the Start Page tab in front of any other open tabs.
About	Displays a window about MPLAB® X IDE.

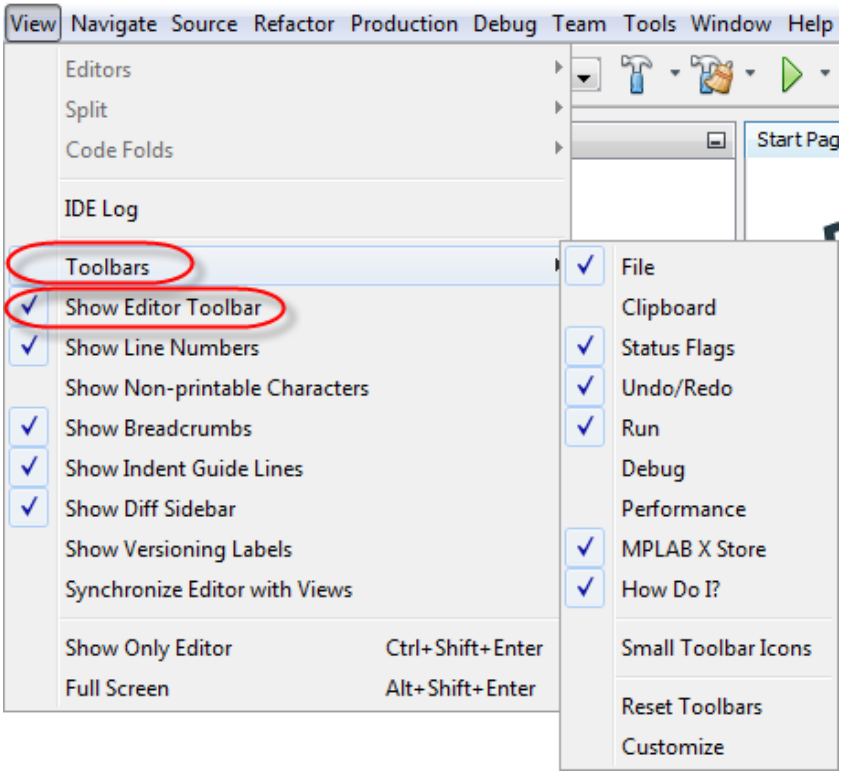
10.2 Toolbars

MPLAB X IDE displays different toolbars depending on which features or tools you are using. The icons in these toolbars provide shortcuts to routine tasks.

The following is useful when working with toolbars:

- Toolbar Features
- Customize Toolbars
- Grayed Out or Missing Items and Buttons

Figure 10-2. View Toolbars



10.2.1 File Toolbar

The File Toolbar currently contains button icons for the following functions. These functions are also on the File menu.

Icon	Icon Text	Function
	New File	Creates a new file with the New File wizard.
	New Project	Creates a new project with the New Project wizard.
	Open Project	Opens an existing project.
	Save All Files	Saves all open files.

Related Links

10.2.11 Toolbar Features
10.2.12 Customize Toolbars

10.2.2 Clipboard Toolbar

The Clipboard Toolbar currently contains button icons for the following functions. These functions are also on the Edit menu.

Icon	Icon Text	Function
	Cut	Deletes the current selection and places it on the clipboard.
	Copy	Copies the current selection to the clipboard.
	Paste	Pastes the contents of the clipboard into the insertion point.

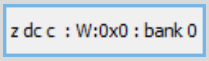
Related Links

10.2.11 Toolbar Features

10.2.12 Customize Toolbars

10.2.3 Status Flags Toolbar

The Status Flags Toolbar contains the following functions:

Image	Image Text	Description
	Program Counter	Program Counter (PC) current value.
	Status Flags	Device status flags. See your device data sheet for information on status bits and/or registers.

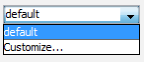
10.2.4 Undo/Redo Toolbar

The Undo/Redo Toolbar currently contains button icons for the following functions. These functions are also on the Edit menu.

Icon	Icon Name	Function
	Undo	Reverses (one at a time) a series of editor actions, except Save.
	Redo	Reverses (one at a time) a series of Undo commands.

10.2.5 Run Toolbar

The Run Toolbar currently contains button icons for the following functions. These functions are also on the Run, Debug and project context menus.

Icon	Icon Text	Function
	Set Project Configuration	Selects project configuration. Choose "default" or "Customize." See Project Properties window for details.
	Build Project	Builds all the project files.
	>Build for Debugging	Builds all the project files for debug, which includes the debug executive. Step one of Discrete Debugger Operation.
	Clean and Build Project	Deletes files from previous builds and then builds the all project files.
	>Clean and Build for Debugging	Deletes files from previous builds and then Builds all the project files for debug, which includes the debug executive. Step one of Discrete Debugger Operation.

.....continued		
Icon	**Icon Text**	**Function**
	Run Project	Builds, programs the target and Runs the selected project.
	Make and Program Device Project	Builds, programs the target and holds in Reset the selected project.
	>Program Device for Debugging	Programs the target with the debug build. Step two of Discrete Debugger Operation.
	>Program Device for Production	Programs the target with the production build.
	>Erase Device Memory	Erase device program memory.
	>Programmer To Go PICkit3	For the PICkit 3 as the project debug tool, program the target using Programmer To Go (PTG).
	Read Device Memory	Reads target device memory and loads into MPLAB X IDE.
	>Discrete Multi-Partition Read Operation	For multi-partition (dual partition) devices, read specified flash memory partition.
	>Read Device Memory to a File	Reads target device memory and saves to a file.
	>Read EE/Flash Data Memory to a File	Reads target device data memory and saves to a file.
	Hold in Reset/ Release from Reset	Alternately holds in or releases from Reset the selected project.
	Debug Project	Builds, programs the target and begins debugging the selected project.
	>Launch Debugger	Launch debugger and run target code. Step three of Discrete Debugger Operation.
	>Live Connect Debug	For MEC devices, connect to a running target and inspect the memory contents. Allows some debugging on a production device.
> = Available from icon down arrow		

Related Links

10.2.11 Toolbar Features

10.2.12 Customize Toolbars

10.2.6 Debug Toolbar

The Debug Toolbar currently contains button icons for the following functions. These functions are also on the Debug menu.

Icon	Icon Text	Function
	Finish Debugger Session	Ends the debugging session.
	Pause	Pauses debugging. Use "Continue" to resume.
	Reset	Runs the current project to the cursor's location in the file and stop program execution.

.....continued

Icon	Icon Text	Function
	Continue	Resumes debugging until the next breakpoint or the end of the program is reached.
	Step Over	Executes one source line of a program. If the line is a function call, executes the entire function then stops.
	Step Into	Executes one source line of a program. If the line is a function call, executes the program up to the function's first statement and stops.
	Step Out	Executes one source line of a program. If the line is a function call, executes the functions and returns control to the caller.
	Run to Cursor	Runs the current project to the cursor's location in the file and stop program execution.
	Set PC at cursor	Sets the program counter (PC) value to the line address of the cursor.
	Focus Cursor at PC	Moves the cursor to the current PC address and centers this address in the window.

Related Links

10.2.11 Toolbar Features

10.2.2 Clipboard Toolbar

10.2.7 Performance Toolbar

The Performance Toolbar contains:

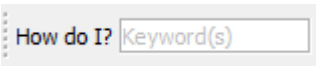
Item	Item Name	Function
	Memory Usage Display	Click to force garbage collection. This may free up memory.
	Profile the Application	Click to begin profiling the application. Click again to stop profiling and save to a file (`.npss`). See MPLAB® REAL ICE™ in-circuit emulator documentation for more on PC sampling and profiling.

10.2.8 MPLAB X Store Toolbar

The MPLAB X Store Toolbar currently contains button icons for the following functions. The MPLAB X Store function is also on the Help menu.

Icon	Icon Text	Function
	MPLAB X Store	Click to open the **MPLAB X Store** tab in the IDE. Find out about products, services, and order devices.
	Programming Center	Click to order information on the Microchip Programming Center in a browser tab. Let Microchip do your device programming in a fast, cost-effective, secure and proven method.

10.2.9 How Do I Toolbar



This feature requires an Internet connection.

Search for information on the Developer Help. Enter your question in the text box. See also:

https://microchipdeveloper.com/

10.1.12 Help Menu

10.2.10 Editor Toolbar

The Editor Toolbar currently contains button icons for the following functions. These functions are also on the Edit and Source menus. This toolbar appears at the top of the tab containing the current file source code.

Figure 10-3. Editor Toolbar

Icon	Icon Text	Function
Source	Source	Views source code.
History	History	Views the history of source code edits.
	Online Data Sheet	Opens the project device data sheet to actively lookup Peripherals and SFR's selected (highlighted) in source code. Currently this functionality is for AVR devices only.
	Last Edited	Moves to the line that contains the last edit made.
	Back	Navigates back.
	Forward	Navigates forward.
	Find Selection	Finds the first occurrence of the selected text.
	Find Previous Occurrence	Finds the previous occurrence of the selected text.
	Find Next Occurrence	Finds the next occurrence of the selected text.
	Toggle Highlight Search	Turns on/off text selected for search.
	Toggle Rectangular Selection	Turns on/off text selected lines for search.
	Previous Bookmark	Cycles backwards through the bookmarks.
	Next Bookmark	Cycles forward through the bookmarks.
	Toggle Bookmark	Sets a bookmark on a line of code.
	Shift Left	Moves the selected line or lines one tab to the left.
	Shift Right	Moves the selected line or lines one tab to the right.
	Start Macro Recording	Start recording keystrokes.

.....continued		
Icon	**Icon Text**	**Function**
	Stop Macro Recording	Stop recording keystrokes.
	Comment	Makes selected line into a comment by adding "//".
	Uncomment	Makes selected comment into a line by removing "//".
	Go to Header/Source	Moves between the header and related source code.

Related Links

10.2.11 Toolbar Features

10.2.12 Customize Toolbars

10.2.11 Toolbar Features

Toolbars have the following features:

- **Hover** the cursor over an icon to pop up the icon function.
- **Click and drag** the toolbar to another location in the toolbar area.
- **Right click** in the toolbar area to show/hide a toolbar or change the contents of some toolbars.
- Select **View>Toolbars** to show/hide a toolbar, change the contents of some toolbars or create a custom toolbar.

10.2.12 Customize Toolbars

You can customize MPLAB X IDE toolbars using the Customize Toolbars window. Select *View>Toolbars>Customize* to open the window.

Available icons include Clean Only, Run, Set PC to Cursor, etc.

Add a function to a toolbar

- Drag an icon from the Customize Toolbars window to a toolbar.

Remove a function from a toolbar

- Drag an icon from a toolbar to the Customize Toolbars window.

Add your own toolbar

- Click "New Toolbar" and name the new toolbar.

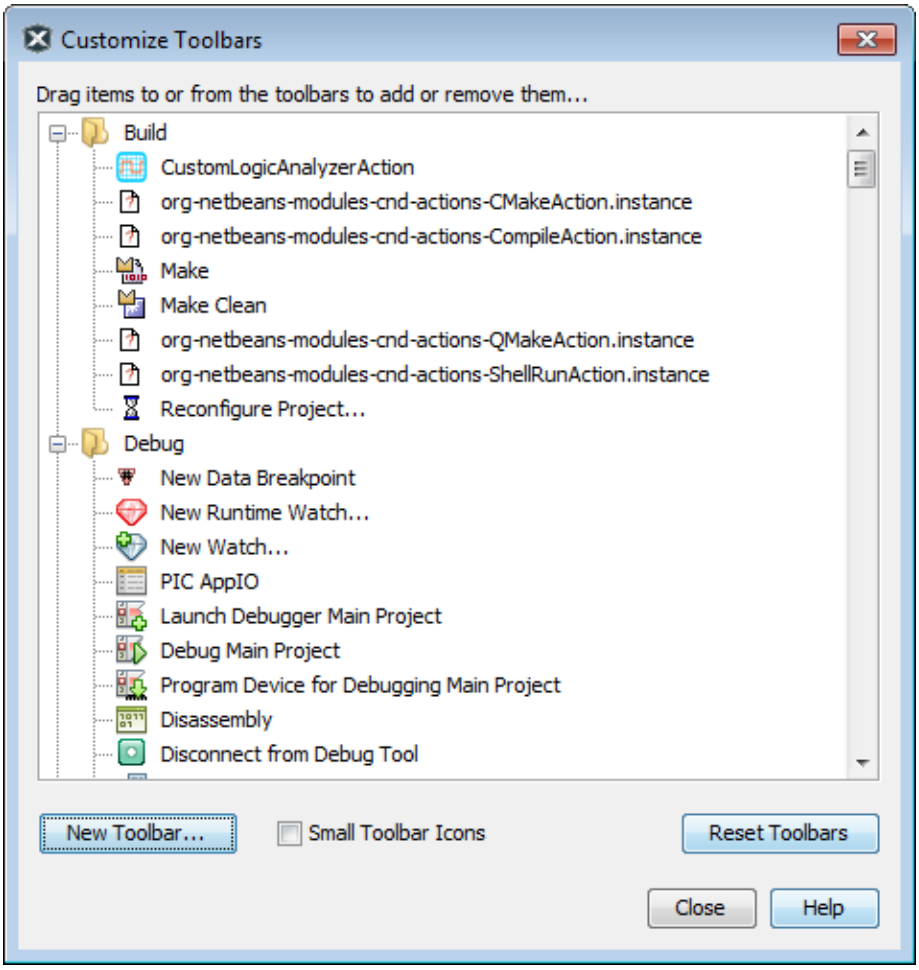
Change toolbar icon size

- Click the check box "Small Toolbar Icons" to make the icons smaller.
- Uncheck to make the icons larger.

Revert to default toolbar

- Click "Reset Toolbars."

Figure 10-4. Customize Toolbars Window



10.3 Status Bar

The status bar will provide up-to-date information on the status of your MPLAB IDE session. Editor information is also provided.

Figure 10-5. Status Bar During Debug

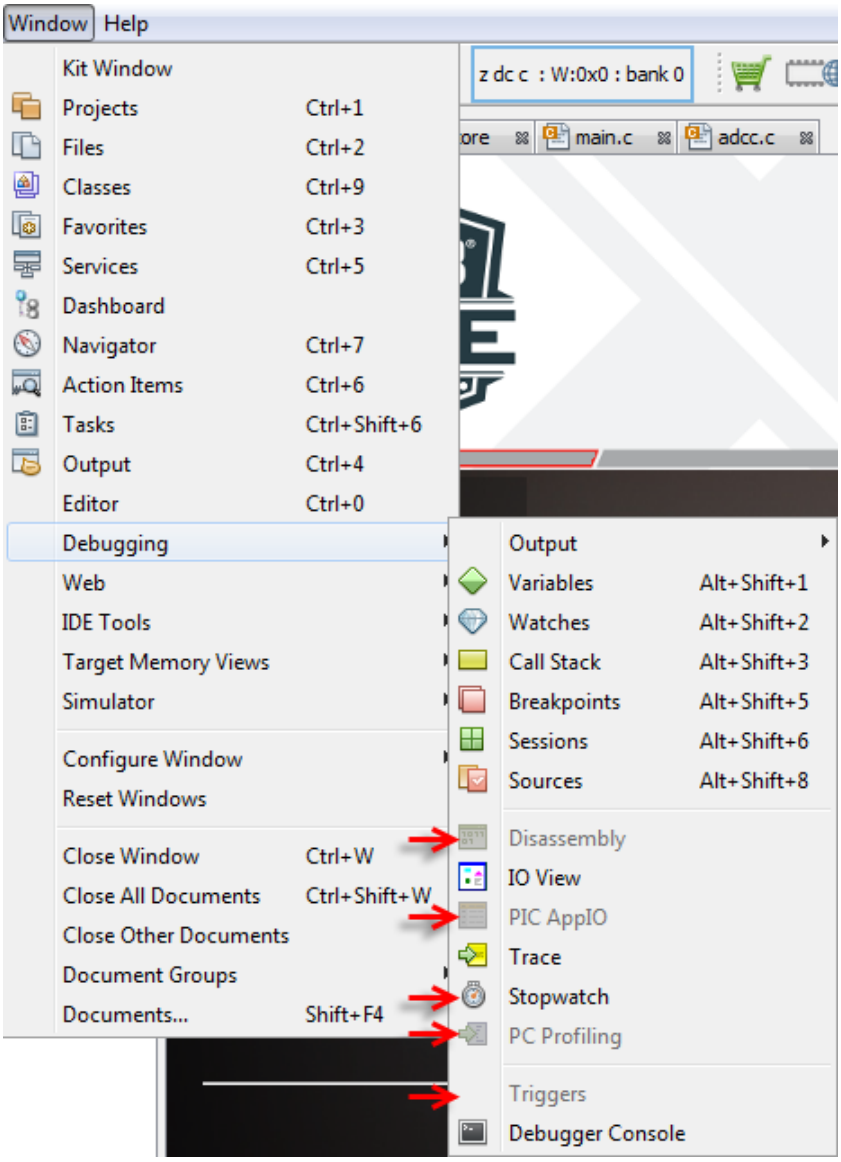
10.4 Grayed Out or Missing Items and Buttons

There are several reasons why a menu item, toolbar button or status bar item may be grayed out (unavailable) or missing:

- The item/button is related to a device feature that the selected device does not have, e.g., the PIC16F877A does not support external memory.
- The item/button is related to a tool feature that the selected tool does not have, e.g., "Step Out" is not available on MPLAB ICD 3.
- The item/button is project-related and no project has been selected, e.g., project build will not be available (No Active Project).
- The item/button is not supported for the selected device or tool.

- The item/button is performing its function and so cannot be selected again, e.g., "Run Project" is grayed out when the program is running.
- The item/button is mutually exclusive to another item, e.g., "Pause" is available when the program is running while "Continue" is grayed out and "Continue" is available when the program is halted while "Pause" is grayed out.

Figure 10-6. Grayed Out Menu Items



11. MPLAB X IDE Windows Behavior

MPLAB X IDE windows each have a specific function to help developers create and debug application code. However, all or some of the windows have features developers need to understand to fully make use each window.

11.1 MPLAB X IDE Windows Management

Information on managing IDE windows may be found in the NetBeans help topic:

Managing IDE Windows

Additional window information is provided below.

11.1.1 Window Data Updates

Open windows are updated on a program halt (except for Flash memory windows, which must be read). A program halt includes a halt after a run and stepping. Halt updates can have the following effects:

- Speed – Updating takes time. To decrease update time, close any unused windows.
- Data overwrites – The value of a file register displayed in an open window is read on halt. See your device data sheet for the register operation on read.

11.1.2 Window Data Changes

MPLAB X IDE window data may be edited as described below. If you cannot edit the data, then that information is not available for you to change.

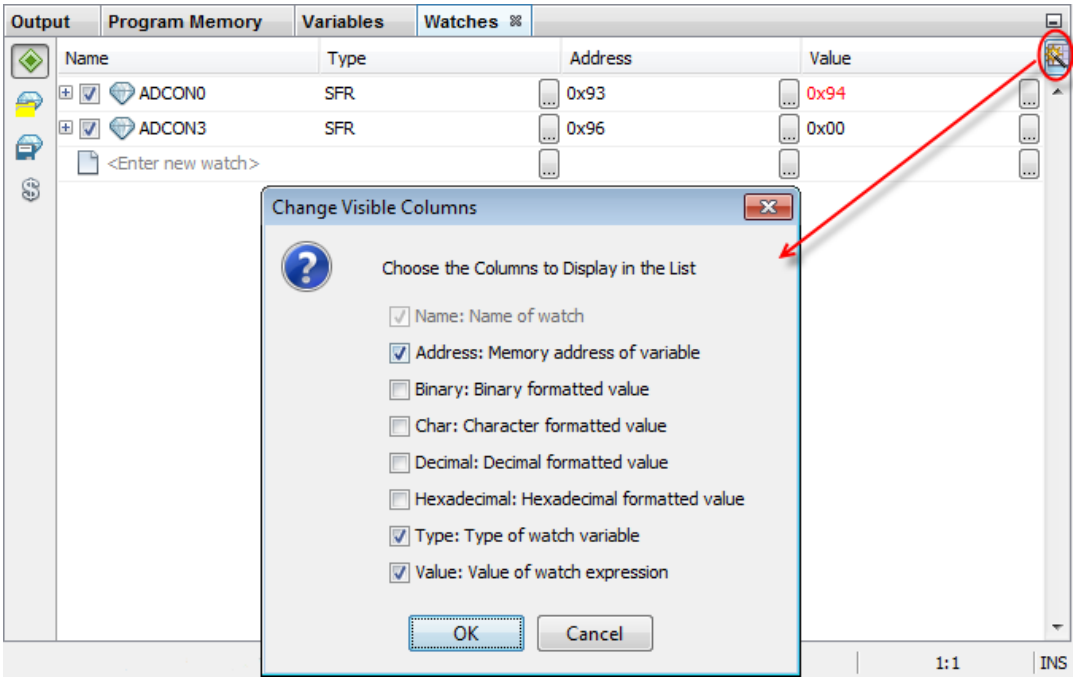
- Data may be edited "in place." Either double click to select an item and then type in a new value, or click on the ellipsis (...) next to an item and type the new value in the window that pops up.
- Data may be chosen from a drop-down list when only certain choices are possible.

11.1.3 Window Column Display

For some windows, you can change the column display. Click on the icon in the window top right corner.

	Change column display.
	Change visible columns.

Figure 11-1. Change Visible Columns Example



11.1.4 Window Focus

To ensure that you have a window in focus, click not only on the window frame, but also on a button, a table cell, or a drop-down combo box.

11.2 Banked Data Memory and Values Displayed in Windows

This feature applies only to 8-bit PIC MCU devices that use banked Data memory (RAM).

In MPLAB X IDE v5.20 and earlier, a register address always includes the bank offset in the value displayed. However, this is not always correct as the actual value can only be accurately calculated when the instruction is at the program counter (PC).

In MPLAB X IDE v5.25 and later, a register address only includes the bank offset when the PC is on the line with the register address. When the PC is not on that line, the value defaults to the one shown in code (no bank offset or bank 0).

Example 11-1. Behavior Change from v5.20 to v5.25

Consider the example code snipet in the figure below. Display behavior for disassembly code in the Program Memory window (DisAssy colomn) or Disassembly window is shown in the table.

Figure 11-2. Example Code Snipet

```
14      asm("MOVLB 0x3A"); // change to bank 58
15      asm("CLRF 0x74"); // clear common RAM register
```

Table 11-1. Behavior Change from v5.20 to v5.25

MPLAB X IDE	Disassembly of Line 15 - Not at PC	Disassembly of Line 15 - At PC
v5.20 and earlier	CLRF 0x1D74	CLRF 0x1D74
v5.25 and later	CLRF 0x74	CLRF 0x1D74

12. MPLAB X IDE Windows and Dialogs

The MPLAB X IDE desktop has four sections, or panes, which contain tabbed windows. Windows may not be visible until a feature has been selected. Windows may have associated dialogs for entering specific information.

As an example, when you first open MPLAB X IDE, only the **Start Page** and **MPLAB X Store** tabbed windows are open. After you open a project, the basic windows open – namely Projects and Files in the top left pane, Dashboard and Navigator in the bottom left pane, and Output in the bottom right pane. The **Start Page** and **MPLAB X Store** tabbed windows move into the top right pane.

MPLAB X IDE windows are a combination of basic NetBeans Platform windows and MPLAB X IDE specific windows. Dialogs open when selected from menu items. As with windows, MPLAB X IDE dialogs are a combination of basic NetBeans Platform dialogs and MPLAB X IDE specific dialogs. For information on NetBeans windows and dialogs, see 13.1 NetBeans Specific Windows and Window Menus.

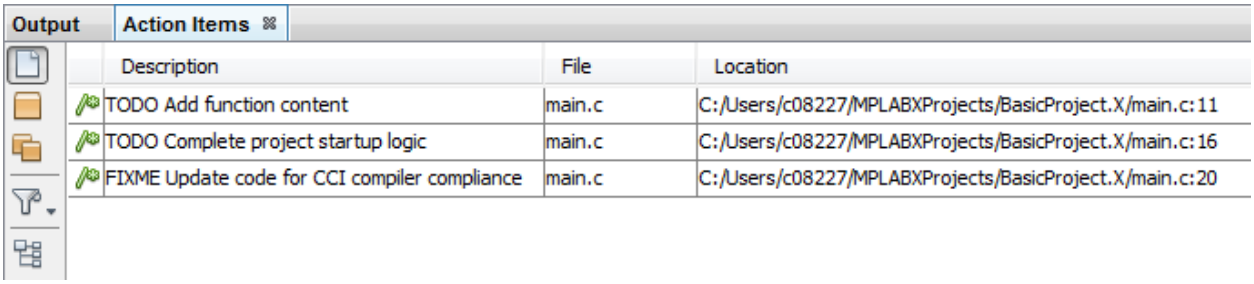
12.1 Action Items Window

Open the Action Item window from *Window>Action Items*.

The Action Items window displays a unified list of things that you need to resolve in your various files and projects, including compiler warnings and errors. Place comments in your code to specify action items. For example, in the figure below the following lines of code were used:

```
//TODO Add function content
       :
//TODO Complete project startup logic
       :
//FIXME Update code for CCI compiler compliance
```

Figure 12-1. Action Items Window Example



You can use filters to determine the entries that are displayed in the list and you can sort the list by clicking a column heading. The Action Items window icons enables you to modify and work with the displayed entries.

For more information, see: http://microchipdeveloper.com/mplabx:tasks-list

12.1.1 Action Items Window Display

This display format shows data in the following columns:

- Description – A description of the action required.
- File – The name of the file action.
- Location – The file location.

The Action Items window icons are displayed on the left side of the window.

Table 12-1. Action Items Window Icons

Icon Text	Icon Text	Description
	Show action items for currently edited file only	When selected, the IDE scans the file that is currently selected in the editor and displays a list of the action items found in that file. When you open a new file in the editor, the file is scanned and only the action items in that file are displayed.
	Show action items for the selected project	When selected, the IDE scans all the files in the currently selected project and displays all the action items in the project window.
	Show action items for all opened projects	When selected, the IDE scans all the files in all open projects and displays a list of all the action items in the window.
	Click here to select a filter	Click the Filter button to modify the filters for the action items or to create a new filter. You can click the drop-down list on this button to select a filter to apply to the list of action items.
	Group action items by category	When selected, the list of action items is grouped by category.

12.1.2 Action Items Window Menu

Select and right click on a row in the window to pop up this menu.

Table 12-2. Action Items Window Context Menu

Item	Description*
Show	Show the location in code of the error/warning.
Scope	Show the scope of the item - current file, current project, all projects.
Filter	Show how the item is filtered. Also change filtering.
Refresh	Refresh the window contents.
Sort By	Specify how the window contents should be arranged.

12.2 Breakpoints Window

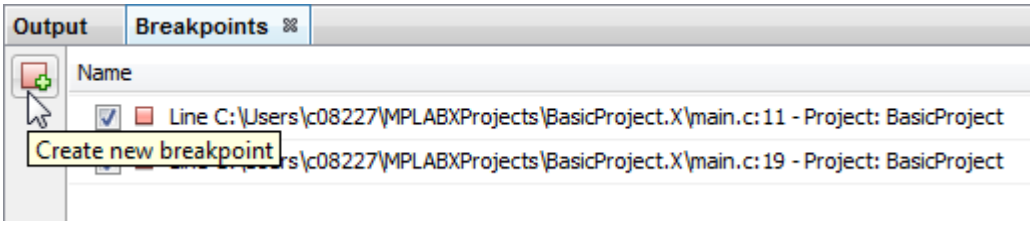
Open the window by selecting *Window>Debugging>Breakpoints*.

The Breakpoints window is used for managing breakpoints in code. For more on using the Breakpoints window, see 4.14 Control Program Execution with Breakpoints.

New Breakpoint Dialog

Click the **Create New Breakpoint** button on the Breakpoint window to open the 4.14.2 Set Breakpoints with the Breakpoint Dialog and create a new breakpoint. Types of breakpoints allowable for the project device are listed.

Figure 12-2. New Breakpoint Icon



Breakpoint Window Menus

Right click in a blank area of the window to pop the menu below.

Table 12-3. Breakpoints Window Menu - No Breakpoint Line Selected

Item	Description
New Breakpoint	Open the New Breakpoint dialog.
Enable All Disable All	Enable or disable all breakpoints.
Delete All	Delete all breakpoints.

Right click on an existing breakpoint row in the window to pop the menu below.

Table 12-4. Breakpoints Window Menu - Breakpoint Line Selected

Item	Description
Go to Source	Show the location in code of the breakpoint.
Complex Breakpoint	For a complex breakpoint, add this breakpoint to a new sequence or tuple. Also specify location in sequence or tuple.
Disable Enable	Disable or re-enable the operation of this breakpoint.
New Breakpoint	Open the New Breakpoint dialog.
Enable All Disable All	Enable or disable all breakpoints.
Delete Delete All	Delete this or all breakpoints.
Customize	Customize the breakpoint behavior.

12.3 New Breakpoint Dialog

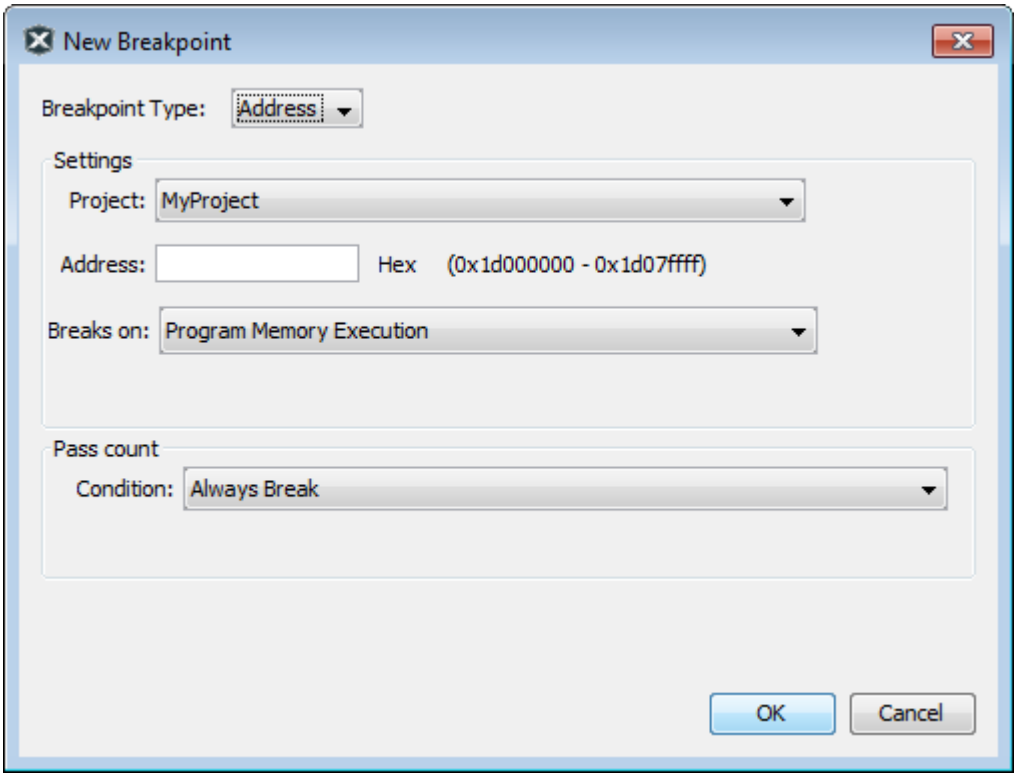
Open this dialog from the Breakpoints Window (*Window>Debugging>Breakpoints*). Either click the "New Breakpoint" icon  or select "New Breakpoint" from a context menu.

The options available for the breakpoint are determined by the type of breakpoint selected and the selected device (not all options are available for all devices).

Types of breakpointss are:

- 12.3.1 Line Breakpoints
- 12.3.2 Data Breakpoints
- 12.3.3 Address Breakpoints
- 12.3.4 Event Breakpoint
- 12.3.5 Function Breakpoints
- 12.3.6 Pass Count Operation

Figure 12-3. New Breakpoint Dialog - Address



12.3.1 Line Breakpoints

The following options are available for breakpoints specified on a line of code.

Table 12-5. Breakpoint Type: Line – Settings

Item	Description
Settings	Create a new Line break point with a left mouse click on the Editor gutter next to the file line. Or, select "Toggle Line Breakpoint" from the Editor context menu.

12.3.2 Data Breakpoints

The following options are available for data memory breakpoints.

Table 12-6. Breakpoint Type: Data – Settings

Item	Description
Project	Select an open project from the drop-down list This is the project in which code will contain the breakpoint.
Symbols	Enter the name of a global symbol, or SFR, or browse to one by clicking **Symbols**. Accessing this symbol according to "Breaks on" triggers a pause in code execution.
Enable Range Address	Check to set a range breakpoint. Uncheck to set a single breakpoint.
Address **Address (Start)**	Depending on the selection of "Enable Range Address", this item can be "Address" or "Address (Start)". Enter a hexadecimal address in data memory Accessing this address according to "Breaks on" triggers a pause in code execution.

.....continued	
Item	**Description**
Address (End)	If "Enable Range Address" is checked, this box is enabled. Enter a hexadecimal address in data memory.
Breaks on	**Read, Write, Read or Write**: Break code execution when the symbol or address stated above is read, written, or either read or written. **Read Specific Value, Write Specific Value, Read or Write Specific Value**: Break code execution when the symbol or address stated above is read and has the value specified below, written with the value specified below, or either read or written according to the value specified below. **Note:** For dsPIC DSCs, there are read and write options for the X and Y buses.
Value	For the "Breaks on" selection of Read Specific Value, Write Specific Value, or Read or Write Specific Value, enter a hexadecimal value here.
Value Comparison	*For PIC16F1xxx devices only-* Compare to "Value" as specified: **= Value**: Equal to value **!= Value**: Not equal to value **> Value**: Greater than value **< Value**: Less than value
Data Value Mask	*For PIC16F1xxx devices only-* Use mask when comparing to "Value" Enter a value in the range 0x00 to 0xhh, where: 0x00: No bits compared 0xhh: All bits compared

Table 12-7. Breakpoint Type: Data – Pass Count

Item	Description
Condition	Determine when the break specified under "Breaks on" occurs. **Always Break**: Always break when the "Breaks on" condition is met. **Break occurs Count Instructions after Event**: After an event ("Breaks on" condition) occurs, execute Count Instructions before actually breaking. **Event must occur Count times**: An event ("Breaks on" condition) must occur Count times before actually breaking.
Count	According to the Condition specified, enter either a count for the number of instructions after an event or the number of events. See 12.3.4 Event Breakpoint.
Trigger Options	*For PIC16F1xxx devices only-* Select when to trigger, either: • Do not trigger out when breakpoint is reached • trigger out when breakpoint is reached
Interrupt Context	*For PIC16F1xxx devices only-* Interrupt Context qualifier for address/data breakpoints. Select from: • Always break (break in both ISR and main code) • Break in main line (non-interrupt) context only – break in main code only • Break in interrupt context only – break in ISR code only

12.3.3 Address Breakpoints

The following options are available for program/execution memory breakpoints.

Table 12-8. Breakpoint Type: Address – Settings

Item	Description
Project	Select an open project from the drop-down list This is the project in which code will contain the breakpoint.
Enable Range Address	Check to set a range breakpoint. Uncheck to set a single breakpoint.
Address **Address (Start)**	Depending on the selection of "Enable Range Address", this item can be "Address" or "Address (Start)". Enter a hexadecimal address in data memory Accessing this address according to "Breaks on" triggers a pause in code execution.
Address (End)	If "Enable Range Address" is checked, this box is enabled. Enter a hexadecimal address in data memory.
Breaks on	**Program Memory Execution**: Break code execution when the address specified above is reached. **TBLRD Program Memory**: Break code execution when a table read to the address specified above occurs. **TBLWT Program Memory**: Break code execution when a table write to the address specified above occurs.

Table 12-9. Breakpoint Type: Data – Pass Count

Item	Description
Condition	Determine when the break specified under "Breaks on" occurs. **Always Break**: Always break when the "Breaks on" condition is met. **Break occurs Count Instructions after Event**: After an event ("Breaks on" condition) occurs, execute Count Instructions before actually breaking. **Event must occur Count times**: An event ("Breaks on" condition) must occur Count times before actually breaking.
Count	According to the Condition specified, enter either a count for the number of instructions after an event or the number of events. See 12.3.4 Event Breakpoint.
Trigger Options	*For PIC16F1xxx devices only-* Select when to trigger, either: • Do not trigger out when breakpoint is reached • trigger out when breakpoint is reached
Interrupt Context	*For PIC16F1xxx devices only-* Interrupt Context qualifier for address/data breakpoints. Select from: • Always break (break in both ISR and main code) • Break in main line (non-interrupt) context only – break in main code only • Break in interrupt context only – break in ISR code only
* See also Section "For some devices (PIC16F1xxx MCUs), enhanced event breakpoints actions are available."	

12.3.4 Event Breakpoint

The following options are available for event breakpoints.

Table 12-10. Breakpoint Type: Event

Item*	Description
Project	Select an open project from the drop down list. This is the project whose code will contain the breakpoint.
Break on clock mode switch	Break when the clock mode switches.
Break on Reset instruction	Break when a device Reset instruction occurs.
Break on Sleep	Break when Sleep is entered.
Break on stack over/underflow	Break when the stack either overflows or underflows.
Break on wake up	Break when the device wakes up from sleep.
Break when watchdog timer has expired	Break when the watchdog timer period has ended.
Break on execution out of bounds	Break when the program attempts to move out of normal program memory/space.
Break on MCLR Reset	Break on a Master Clear (MCLR) Reset.
Break on trigger in signal	Break when a Trigger In pulse is detected.
* Select event(s) from the list that will cause executing code to pause (break). Some events may not be available for your device.	

For some devices (PIC16F1xxx MCUs), enhanced event breakpoints actions are available.

Action	Description
Break	Break (halt) execution per option specified.
Trigger out	Emit a trigger out pulse per option specified.
Break and trigger out	Break (halt) execution and emit a trigger out pulse per option specified.

12.3.5 Function Breakpoints

The following options are available for function breakpoints.

Table 12-11. Breakpoint Type: Function – Settings

Item	Description
Symbol/Hex Address	Enter or select either a symbol representation or address for a function.

Table 12-12. Breakpoint Type: Function – Conditions

Item	Description
Condition	Check to enable. Then enter a break condition.
Break when hit count exceeds	Check to enable. Then enter an integer number to represent the number of times the function is to be entered, after which a break occurs.

Table 12-13. Breakpoint Type: Function – Actions

Item	Description
Suspend	On break, suspend: **No thread (continue)**: No program suspension. **Breakpoint thread**: Only suspend the function thread **All threads**: Suspend all threads (not main program)

.....continued	
Item	**Description**
Thread ID	Set ID of breakpoint thread.
Print Text	Enter text to be displayed in the Output window when a break occurs.

12.3.6 Pass Count Operation

Using a pass count allows you to delay breaking until after a specified count.

Break occurs Count Instructions after Event

Count is the number of instructions that execution passes after the breakpoint but before stopping.

For example,

- 0 specifies that execution stops immediately
- 1 specifies that execution stops after one additional instruction
- 10 specifies that execution stops after ten additional instructions

Event must occur Count times

Count is the number of times that execution passes the event until stopping.

For example,

- 0 specifies that execution stops immediately
- 1 specifies that execution passes the event one time, then stops the next time (the second time the event occurs)
- 10 specifies that execution passes the event ten times, then stops the next time (the eleventh time the event occurs)

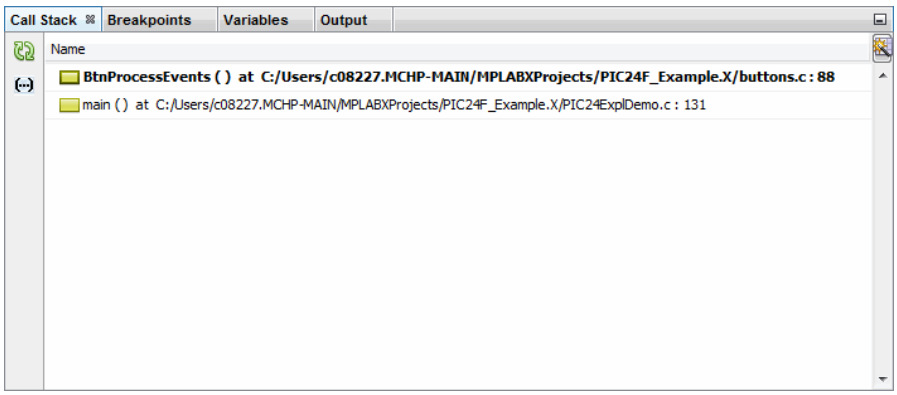
12.4 Call Stack Window

Select *Window>Debugging>Call Stack* to open the Call Stack window. You must be in debug mode to view content.

The Call Stack window displays C functions and their arguments listed in the order in which they were called in the executing program. C code that is not optimized works best.

The Call Stack is available for 16- and 32-bit devices. The window is updated on Debug Pause or Halt.

Figure 12-4. Call Stack Window



The following icons are available on the window on Pause or Halt.

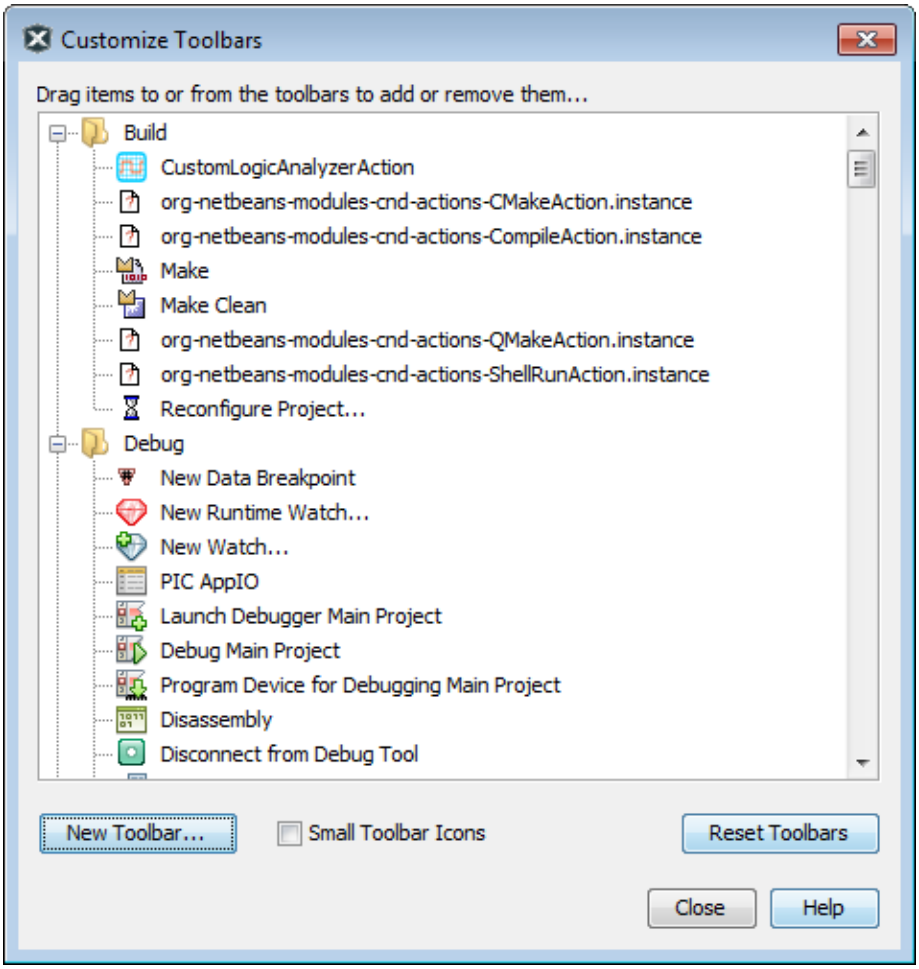
Table 12-14. Call Stack Icons

Icon	Description
	Auto or Manual Refresh of Window Contents. Depends on the value of "Disable auto refresh for call stack view during debug sessions" under *Tools>Options>Embedded>Generic Settings* – Unchecked = false (default), checked = true. See 12.16.1 Generic Settings Tab. False = Auto Refresh (Green Icon): The window contents are automatically updated on Pause or Halt. If the window is closed or not focused, selecting the window and clicking this button will display the updates without having to run and pause/halt again. True = Manual Refresh (Orange Icon): The window contents are not automatically updated on Pause or Halt. This button must be clicked to update window contents on a pause/halt.
	Select to disable/enable evaluation of function parameter variables.
	Change visible columns. For this window, show or hide location of call stack frame.

12.5 Customize Toolbars Window

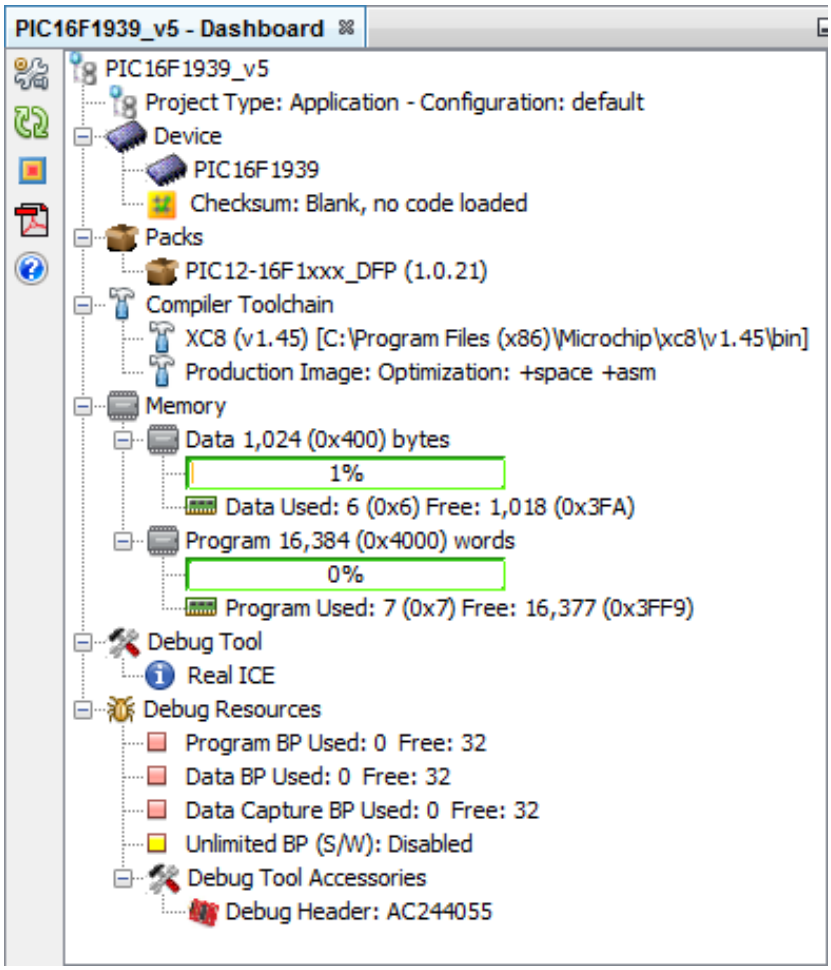
You can customize MPLAB X IDE toolbars using the Customize Toolbars window. Select *View>Toolbars>Customize* to open the window.

For customization instructions, see 10.2.12 Customize Toolbars.



12.6 Dashboard Window

The Dashboard window contains general project information, such as checksums, compiler versions, memory usage, and breakpoint resources. For more information, see: 5.19 View the Dashboard Display.



12.7 Disassembly Window

View disassembled C code in the Disassembly window. Select *Window>Debugging>Disassembly* to open this window. You must be in debug mode.

When debugging, the "Step Over" function works in the source code context. Disassembly debugging is done using the "Step In" function. If during Disassembly debugging "Step In" is not needed - for example in cases where a CALL instruction needs to be stepped over and not stepped in - this can be achieved by using the "Run to Cursor" feature. Place the cursor at the line where PC needs to be run to and hit the "Run to Cursor"/F4 button.

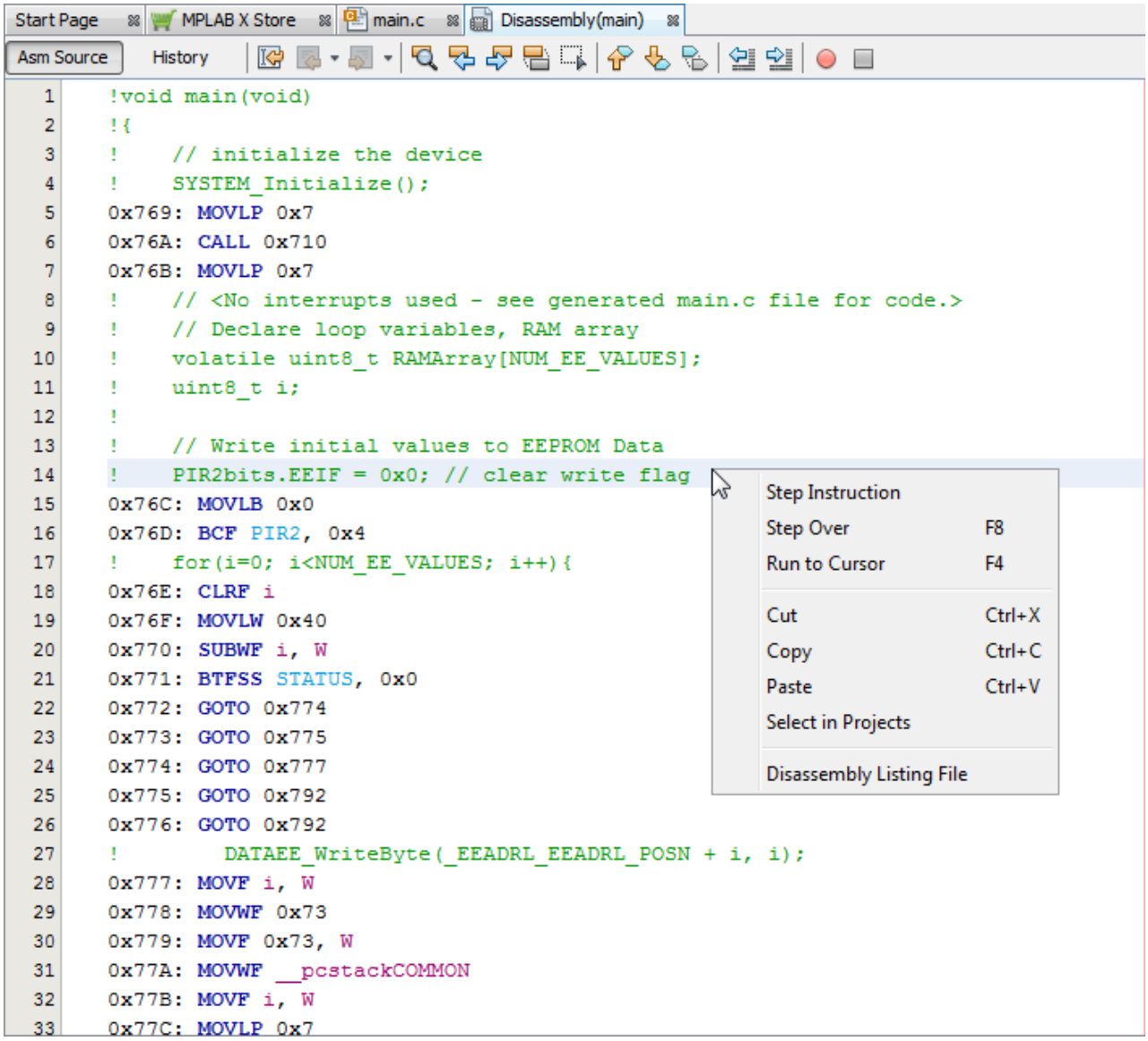
Right click in the window to see the context menu.

A quick way to view the entire disassembly file is to select "Disassembly Listing File."

If your project device uses RAM banking, see also:

11.2 Banked Data Memory and Values Displayed in Windows

Figure 12-5. Disassembly Window



12.8 I/O View Window

Use the I/O View window (*Window>Debugging>IO View*) to see an overview of registers of the target device for the current project. For an overview of operation, see:

5.20 View Registers for the Project (I/O View)

I/O View Window Sections

The default view of the window is vertically split with peripheral groups in the top section and registers in the bottom section. Each peripheral typically has defined settings and value enumerations which can be displayed by expanding a register in the peripheral view (top section). The register view (bottom section) will display all registers which belong to a selected peripheral group. If no peripheral is selected, the view is empty. Each register can also be expanded to display the predefined value groupings that belong to the register.

Figure 12-6. I/O View Window with Content

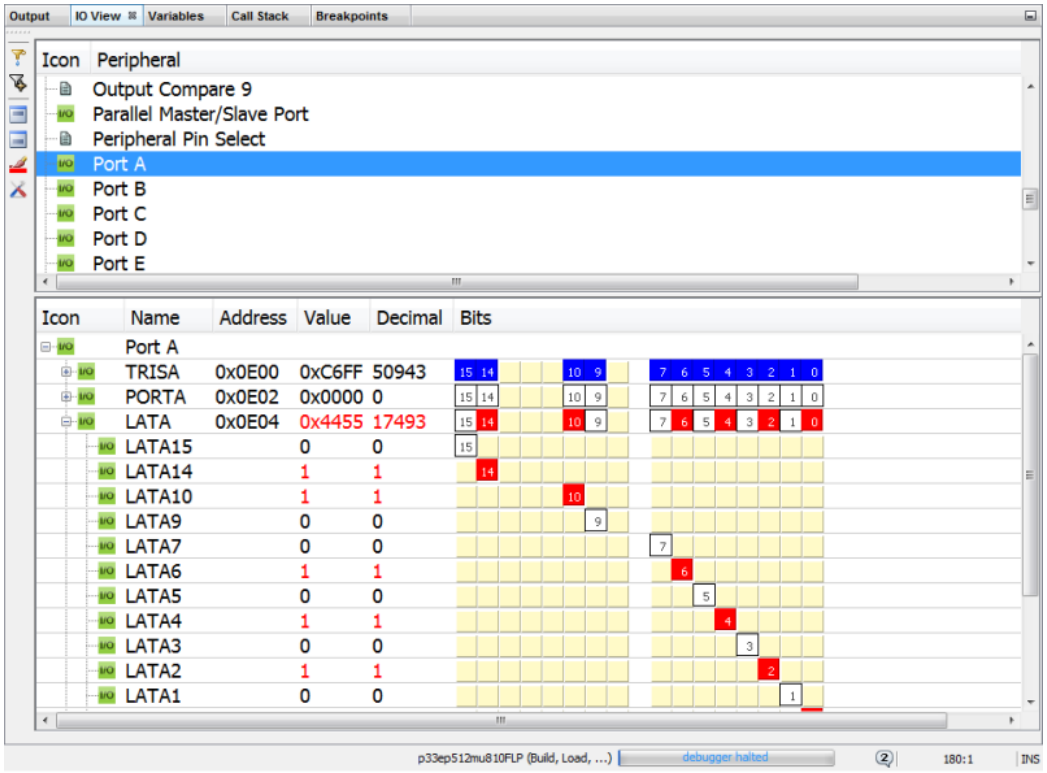


Table 12-15. I/O View Icons

Icon	Function
	Filter Groups. Filter by groups of peripherals available in view. Click icon to open the Select dialog. Select peripherals using Shift-click (range) or Ctrl-click (individually). Alternately you can specify a Register Text Filter file.
	Filter Select All Groups.
	Toggle Peripheral Pane. Click to show/hide peripheral (top) pane.
	Toggle Register Pane. Click to show/hide register (bottom) pane.
	Showing Standard Bits. Toggle for Highlight Modified Bits. Standard bits (no highlighting) are displayed for modified bits. Click on blue line icon to change to red line icon.
	Highlighting Modified Bits. Toggle for Standard Bits. Modified bits are highlighted in display. Click on red line icon to change to blue line icon.

.....continued

Icon	Function
	Toggle Highlighting of Modified Bits. If previous icon selected as "Highlighting Modified Bits" then click this icon to toggle the type of highlighting for modified bits. Click to open the IO View Settings dialog.

Table 12-16. I/O View Context Menus

Item	Function
Expand All	Expand all collapsed content in pane/section.
Expand Row	Expand collapsed content in selected row.
Collapse All	Collapse all expanded content in pane/section.
Adjust Table Columns	Register Pane Only. Adjust width of table columns.

12.9 Memory Windows - 8- and 16-Bit Devices

Memory windows (*Window>Target Memory Views*) display the many types of device memory, such as SFRs and Configuration bits. Use the "Memory" and "Format" drop-down boxes to customize your window.

For more on these controls, see 4.18 View or Change Device Memory.

For details on associated dialogs, see 12.11 Memory Windows Associated Dialogs.

Figure 12-7. Window>Target Memory Views - PIC16F1939

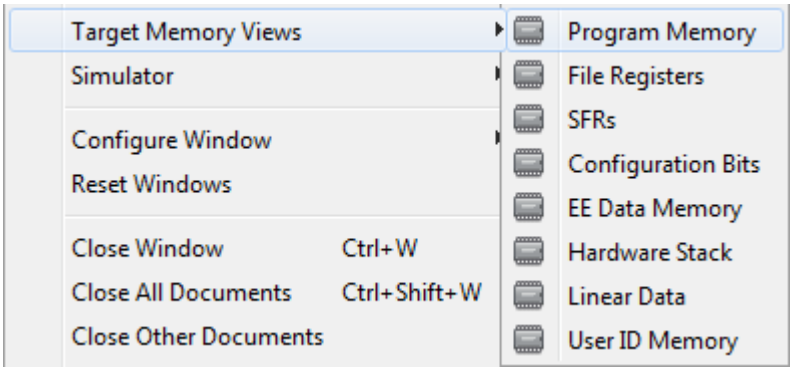
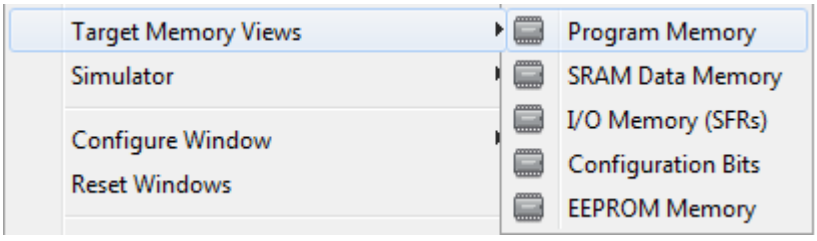


Figure 12-8. Window>Target Memory Views - ATtiny817



12.9.1 Program Memory Window

The Program Memory window displays locations in the range of program memory for the currently selected processor. If external program memory is supported by the selected device and enabled, it will also appear in the Program Memory window.

For the MPLAB X Simulator, when a program memory value changes or the processor is halted, the data in the Program Memory window is updated.

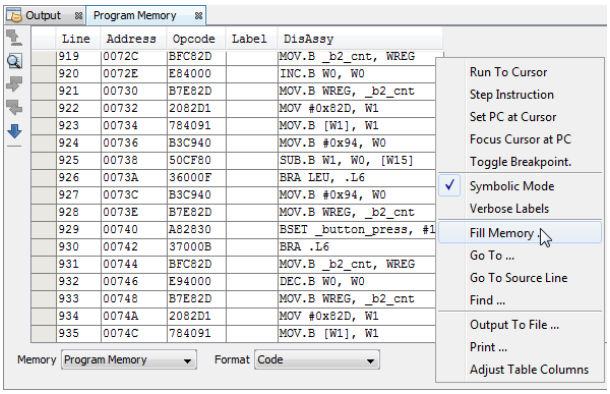
For any Microchip hardware debug tool (e.g., MPLAB ICD 4 in-circuit debugger), when a program memory value changes or the processor is halted, the data in the Program Memory window is **not** updated; you must do a Read of device memory.

When using a hardware tool for debug, some registers may show an "R" for each nibble of data to represent a reserved resource.

If your project device uses RAM banking, see also:

11.2 Banked Data Memory and Values Displayed in Windows

Figure 12-9. Program Memory Window with Content



12.9.1.1 Program Memory Window Displays

You may specify how memory is displayed in the window by selecting from the Format drop-down box on the bottom of the window.

When using a hardware tool for debug, some registers may show an "R" for each nibble of data to represent a reserved resource.

Code Format

Code format displays disassembled hex code with symbols. The window will have the following columns:

- Debug Info – Information useful for debugging
  . A pointer shows the current location of the program counter.
- Line – Reference line number.
- Address – Opcode hexadecimal address.
- Opcode – Hexadecimal opcode, shown in 2- or 3-byte blocks
  For most PIC MCUs, these blocks represent words.
  For PIC18CXXX devices, the blocks represent 2 bytes.
  For dsPIC DSC devices, the blocks represent 3 bytes.
- Label (Symbolic Only) – Opcode label in symbolic format.
- Disassembly – A disassembled version of the opcode mnemonic.

Hex Format

This format displays program memory information as hex code. The window will have the following columns:

- Address – Hexadecimal address of the opcode in the next column.
- Opcode Blocks – Hexadecimal opcode, shown in 2- or 3-byte blocks
  For most PIC MCUs these blocks represent words. For PIC18CXXX devices, the blocks represent 2 bytes. For dsPIC DSC devices, the blocks represent 3 bytes.
  The opcode block that is highlighted represents the current location of the program counter.
- ASCII – ASCII representation of the corresponding line of opcode.

PSV Mixed (dsPIC DSC/PIC24 devices only)

This format displays program memory as opcode and the PSV area (CORCON register, PSV bit set). The window will have the following columns:

- Debug Info – Information useful for debugging. A pointer shows the current location of the program counter.
- Line – Reference line number.
- Address – Opcode hexadecimal address.
- PSV Address – Data space hexadecimal address of the opcode.
- Data – Opcode formatted as data.
- Opcode – Hexadecimal opcode, shown in 3-byte blocks.
- Label – Opcode label in symbolic format.
- Disassembly – A disassembled version of the opcode mnemonic.

For more information, see the dsPIC30F Family Reference Manual (DS70046).

PSV Data (dsPIC DSC/PIC24 devices only)

This format displays program memory as file registers, for when program space is visible in data space (CORCON register, PSV bit set). The window will have the following columns:

- Address – Program space hexadecimal address of the data.
- PSV Address – Data space hexadecimal address of the data.
- Data Blocks – Hexadecimal data, shown in 3-byte blocks
  . The highlighted data block represents the current location of the program counter.
- ASCII – ASCII representation of the corresponding line of data.

For more information, see the dsPIC30F Family Reference Manual (DS70046).

12.9.1.2 Program Memory Window Icons

Icons are located on the left side of the window.

Table 12-17. Program Memory Window Icons

Icon	Icon Text	Function
	Refresh by Read Device Memory	Same function as the Debug toolbar "Read Device Memory" icon - uploads device memory to the MPLAB X IDE.
	Find	Specify a string to find in the window. Select to match whole word or case.
	Find Next	Find next instance of string from Find.
	Find Previous	Find previous instance of string from Find.
	Go To	Go to specified line number or address.

12.9.1.3 Program Memory Window Menus

Right click in the memory window data area set to CODE OR PSV MIXED FORMAT to pop up this menu. Not all items may be visible for all devices.

Table 12-18. Program Memory Window Menu - Code/PSV Mixed

Item	Description
Run to Cursor	Run the program to the current cursor location.
Step Instruction	Perform a step instruction.
Set PC at Cursor	Set the Program Counter (PC) to the cursor location.
Focus Cursor at PC	Move the cursor to the current PC address and centers this address in the window.

.....continued	
Item	**Description**
Toggle Breakpoint	Toggle (on/off) existing breakpoint.
Symbolic Mode	Display disassembled hex code with symbols.
Verbose Labels	**PSV Mixed Format Only** Show internal compiler labels.
Fill Memory	Fill memory from Start Address to End Address with the value in Data. Specify other options in the Fill Memory dialog.
Go To	Go to the address/function specified using the Go To dialog.
Go To Source Line	Go to the corresponding line in source code in the editor.
Find	Find text specified using the Find dialog.
Output To File	Write the displayed window contents to a text file using the Output to File dialog.
Print	Print the contents of this window using the Print dialog. **Note:** If you have a large memory device, the number of pages printed can become very large. A suggestion is to print the window contents to a file (Print dialog, **General** tab, "Print to File" check box) and then select which pages from the file you need to print.
Adjust Table Columns	Adjust the columns automatically.

Right click in the memory window data area set to HEX OR PSV DATA FORMAT to pop up this menu. Not all items may be visible for all devices.

Table 12-19. Program Memory Window Menu - Hex/PSV Data

Item	Description*
Hex Display Width	**Hex Format Only** Set the hexadecimal display width (Options depend on the device selected).
Fill Memory	Fill memory from Start Address to End Address with the value in Data. Specify other options in the Fill Memory dialog.
Go To	Go to the address/function specified using the Go To dialog.
Find	Find text specified using the Find dialog.
Output To File	Write the displayed window contents to a text file using the Output to File dialog.
Import Table	Import tabular data from a file into a Memory window using the Import Table dialog.
Export Table	Export tabular data from a Memory window into a file using the Export Table dialog.
Print	Print the contents of this window using the Print dialog. **Note:** If you have a large memory device, the number of pages printed can become very large. A suggestion is to print the window contents to a file (Print dialog, General tab, "Print to File" check box) and then select which pages from the file you need to print.
Adjust Table Columns	Adjust the columns automatically.

12.9.2 File Registers Window

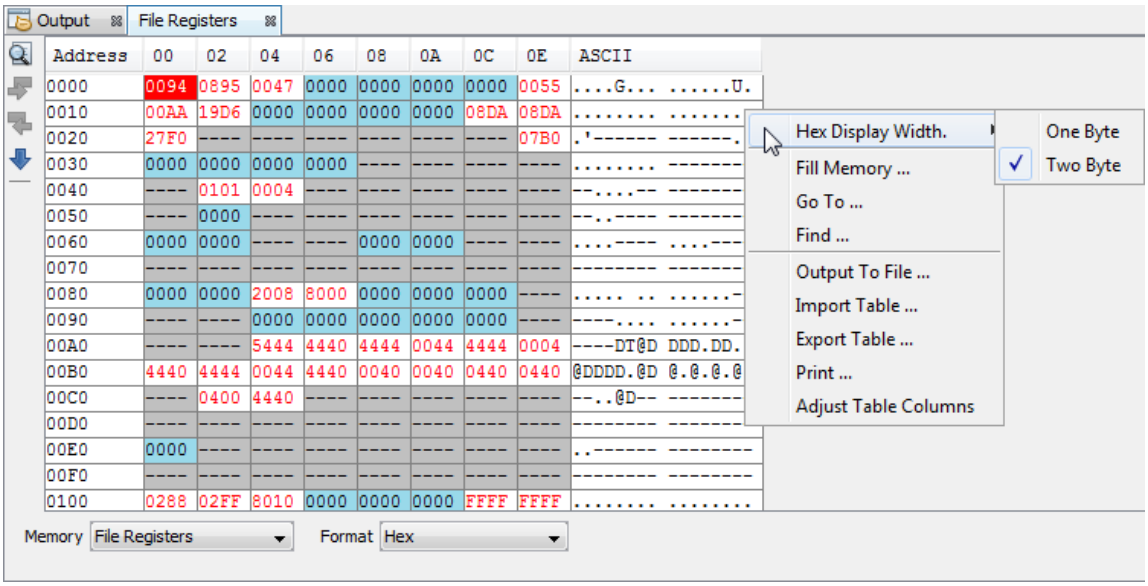
The File Registers window displays all the file registers of the selected device. When a file register value changes, or the processor is interrogated, the data in the File Registers window is updated (values updated shown in red).

Some data has colored cells:

- Blue cells represent SFRs.
- Green cells represent project variables.
- Gray cells represent reserved memory.

Note: To speed up debugging with certain hardware tools, close this window. Use the SFR or Watch window instead.

Figure 12-10. File Register Window with Content



12.9.2.1 File Registers Window Displays

You may specify how memory is displayed in the window by selecting from the Format drop-down box on the bottom of the window.

When using a hardware tool for debug, some registers may show an "R" for each nibble of data to represent a reserved resource.

Hex

This format displays file register information as hex data. The window has the following columns:

- Address – Hexadecimal address of the data in the next column.
- Data Blocks – Hexadecimal data, shown in 1- or 2-byte blocks.
- ASCII – ASCII representation of the corresponding line of data.

Symbol

This format displays each file register symbolically with corresponding data in hex, decimal, binary and character formats. The window has the following columns:

- Address – Data hexadecimal address.
- Symbol Name – Symbolic name for the data.
- Radix Information – Hex, Decimal, Binary, Char
  Radix information is displayed in these four columns. Hex is shown in 1- or 2-byte blocks.

Dual Port (dsPIC33FJ DSC/PIC24HJ MCU devices only)

This format displays file register information as hex data. The window has the following columns:

- Address – Data hexadecimal address.
- DMA Address – Offset from the silicon DMA address.
- Data Blocks – Hexadecimal data, shown in 1- or 2-byte blocks.
- ASCII – ASCII representation of the corresponding line of data.

For information on dsPIC33F DSC and PIC24H MCU devices, see the Microchip web site for device data sheets and dsPIC33F and PIC24H Reference Manual sections.

XY Data (dsPIC DSC devices only)

This format displays file register information as hex data. The window has the following columns:

- Address – X hexadecimal address of the data.
- Y Bus – Y hexadecimal address of data, if supported.
- Data Blocks – Hexadecimal data, shown in 2-byte blocks.
- ASCII – ASCII representation of the corresponding line of data.

For more information on dsPIC DSC devices, see "dsPIC30F Family Reference Manual" (DS70046).

12.9.2.2 File Registers Window Icons

Icons are located on the left side of the window.

Table 12-20. File Registers Icons

Icon	Icon Text	Function
	Find	Specify a string to find in the window. Select to match whole word or case.
	Find Next	Find next instance of string from Find.
	Find Previous	Find previous instance of string from Find.
	Go To	Go to specified line number or address.

12.9.2.3 File Registers Window Menu

Right click in the memory window data area to pop up this menu. Not all items may be visible for all devices.

Table 12-21. File Registers Window Context Menu

Item	Description
Hex Display Width	**Hex Format Only** Set the hexadecimal display width (options depend on the device selected).
Fill Memory	Fill memory from Start Address to End Address with the value in Data. Specify other options in the Fill Memory dialog.
Go To	Go to the address/function specified using the Go To dialog.
Find	Find text specified using the Find dialog.
Output To File	Write the displayed window contents to a text file using the Output to File dialog.
Import Table	**Hex, Dual Port or XY Data Formats Only** Import tabular data from a file into a Memory window using the Import Table dialog.
Export Table	**Hex, Dual Port or XY Data Formats Only** Export tabular data from a Memory window into a file using the Export Table dialog.
Print	Print the contents of this window using the Print dialog. **Note:** If you have a large memory device, the number of pages printed can become very large. A suggestion is to print the window contents to a file (Print dialog, **General** tab, "Print to File" check box) and then select which pages from the file you need to print.
Adjust Table Columns	Adjust the columns automatically.

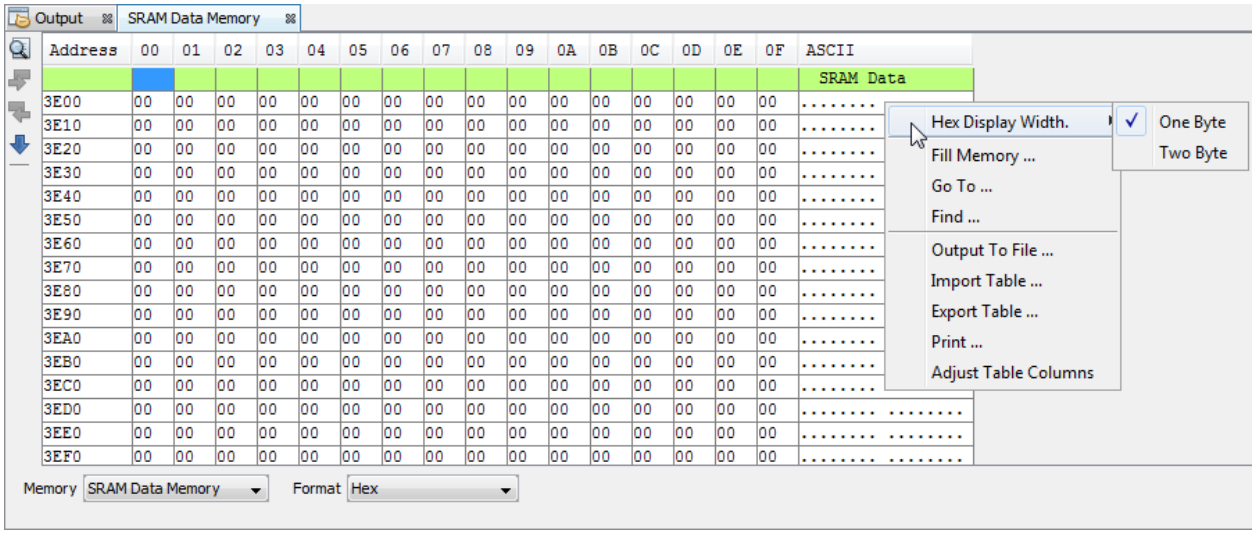
12.9.3 SRAM Data Memory Window

The SRAM Data Memory window displays all the SRAM of the selected AT device. When a register value changes, or the processor is interrogated, the data in this window is updated (values updated shown in red).

When using a hardware tool for debug, some registers may show an "R" for each nibble of data to represent a reserved resource.

Note: To speed up debugging with certain hardware tools, close this window. Use the I/O Memory (SFRs) Window or Watches Window instead.

Figure 12-11. SRAM Data Memory Window with Content



12.9.3.1 SRAM Data Memory Window Displays

You may specify how memory is displayed in the window by selecting from the Format drop-down box on the bottom of the window.

Hex

This format displays file register information as hex data. The window has the following columns:

- Address – Hexadecimal address of the data in the next column.
- Data Blocks – Hexadecimal data, shown in 1- or 2-byte blocks.
- ASCII – ASCII representation of the corresponding line of data.

Symbol

This format displays each file register symbolically with corresponding data in hex, decimal, binary and character formats. The window has the following columns:

- Address – Data hexadecimal address.
- Symbol Name – Symbolic name for the data.
- Radix Information – Hex, Decimal, Binary, Char.

Radix information is displayed in four columns. Hex is shown in 1- or 2-byte blocks.

12.9.3.2 SRAM Data Memory Window Icons

Icons are located on the left side of the window.

Table 12-22. SRAM Data Memory Window Icons

Icons	Icon Text	Function
	Find	Specify a string to find in the window. Select to match whole word or case.

.....continued

Icons	Icon Text	Function
	Find Next	Find next instance of string from Find.
	Find Previous	Find previous instance of string from Find.
	Go To	Go to specified line number or address.

12.9.3.3 SRAM Data Memory Window Menu

Right click in the memory window data area to pop up this menu. Not all items may be visible for all devices.

Table 12-23. SRAM Data Memory Window Context Menu

Item	Description
Hex Display Width	**Hex Format Only** Set the hexadecimal display width (options depend on the device selected).
Fill Memory	Fill memory from Start Address to End Address with the value in Data. Specify other options in the Fill Memory dialog.
Go To	Go to the address/function specified using the Go To dialog.
Find	Find text specified using the Find dialog.
Output To File	Write the displayed window contents to a text file using the Output to File dialog.
Import Table	**Hex Format Only** Import tabular data from a file into a Memory window using the Import Table dialog.
Export Table	**Hex Format Only** Export tabular data from a Memory window into a file using the Export Table dialog.
Print	Print the contents of this window using the Print dialog. **Note:** If you have a large memory device, the number of pages printed can become very large. A suggestion is to print the window contents to a file (Print dialog, **General** tab, "Print to File" check box) and then select which pages from the file you need to print.
Adjust Table Columns	Adjust the columns automatically.

12.9.4 SFRs Window

The Special Function Registers (SFRs) window displays the contents of the SFRs for the selected processor. The format provided by this window is more useful for viewing the SFRs than the normal file register window, since each SFR name is included and several number formats are presented. To view only a few SFRs, you might want to use a Watches window, which may help with speed issues when using hardware debug tools (i.e., faster window update rate).

Whenever a break occurs, the contents of the Special Function Registers are updated.

When using a hardware tool for debug, some registers may show an "R" for each nibble of data to represent a reserved resource.

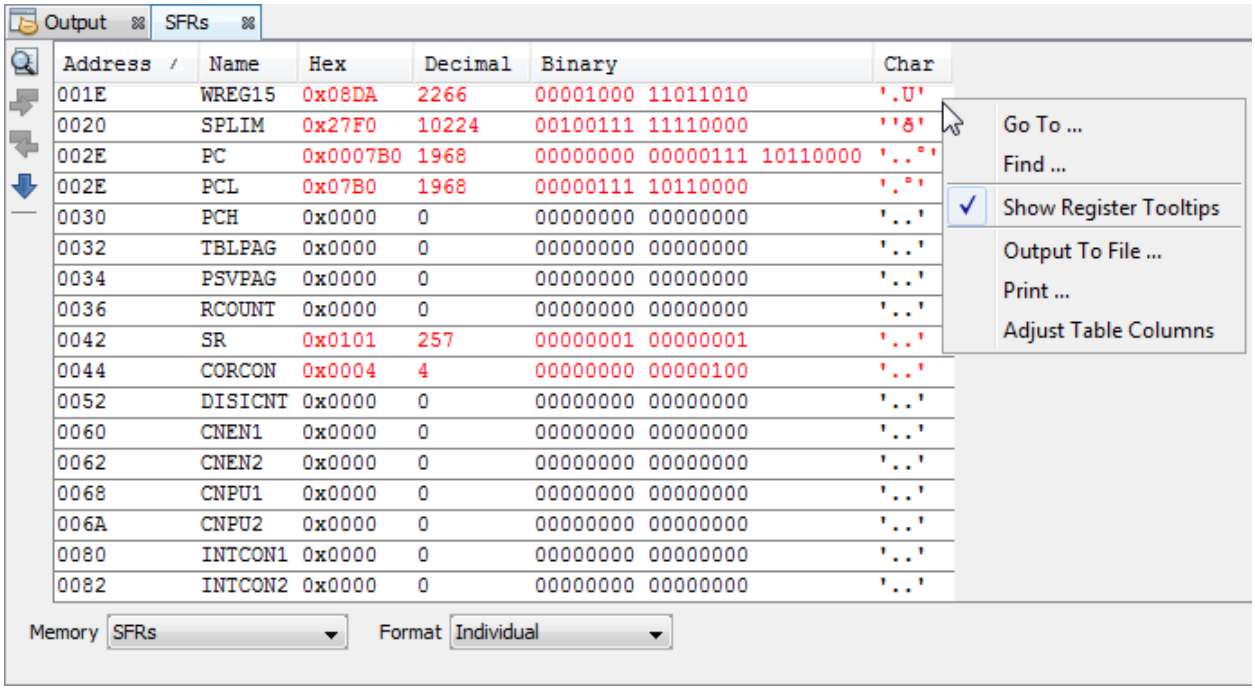
Visible Registers

If a data memory register is not physically implemented on a device, it may not appear in the SFR list. Some tools, such as simulators, may allow you to see registers that do not exist on the actual device, such as prescalers.

Single Stepping

If "Freeze Peripherals On Halt" is selected, the I/O port bits in the SFR or the Watches windows will not update when single stepping. The pin will be modified, but the read request to retrieve the new value is blocked by the freeze and cannot be updated until the next step or run command.

Figure 12-12. SFRs Window with Content



12.9.4.1 SFRs Window Displays

You may specify how memory is displayed in the window by selecting from the Format drop-down box on the bottom of the window. Choices are:

- Individual - View all SFR registers together.
- Peripherals - View SFR registers in functional groups.

Individual

In this display, SFRs are listed by address.

Data is displayed in the following columns.

- Address – SFR hexadecimal address.
- Name – Symbolic name for the SFR.
- Radix Information – Hex, Decimal, Binary
  Radix information is displayed in these four columns. Hex is shown in 1-byte blocks.

Peripheral

In this display, SFRs are grouped according to their related device peripherals.

Data is displayed in the following columns:

- Address – SFR hexadecimal address.
- Name – Name of peripheral or symbolic name for the SFR.
- Radix Information – Hex, Decimal, Binary, Char
  Radix information is displayed in these four columns. Hex is shown in 1-byte blocks.

Click the **Filter Peripheral** button to view the SFRs for only the peripherals selected.

12.9.4.2 SFRs Window Icons

Icons are located on the left side of the window.

Table 12-24. SFRs Window Icons

Icon		Function
	Find	Specify a string to find in the window. Select to match whole word or case.
	Find Next	Find next instance of string from Find.
	Find Previous	Find previous instance of string from Find.
	Go To	Go to specified line number or address.
	Filter Peripherals	**Peripheral Format Only** Filter by peripherals available in view. Click icon to open the Select dialog. Select peripherals using Shift-click (range) or Ctrl-click (individually).
	Filter Select All Peripherals.	**Peripheral Format Only** Filter Select All Peripherals.

12.9.4.3 SFRs Window Menu

Right click in the memory window data area to pop up this menu.

Table 12-25. SFRs Window Context Menu

Item	Description*
Find	Find text specified using the Find dialog.
Output To File	Write the displayed window contents to a text file using the Output to File dialog.
Print	Print the contents of this window using the Print dialog. **Note:** If you have a large memory device, the number of pages printed can become very large. A suggestion is to print the window contents to a file (Print dialog, **General** tab, "Print to File" check box) and then select which pages from the file you need to print.
Adjust Table Columns	Adjust the columns automatically.

12.9.5 I/O Memory (SFRs) Window

The I/O Memory window displays the contents of the Special Function Registers (SFRs) for the selected AT processor. The format provided by this window is more useful for viewing the SFRs than the normal SRAM Data Memory Window, since each SFR name is included and several number formats are presented. To view only a few SFRs, you might want to use a Watches Window, which may help with speed issues when using hardware debug tools (i.e., faster window update rate).

Whenever a break occurs, the contents of the Special Function Registers are updated.

When using a hardware tool for debug, some registers may show an "R" for each nibble of data to represent a reserved resource.

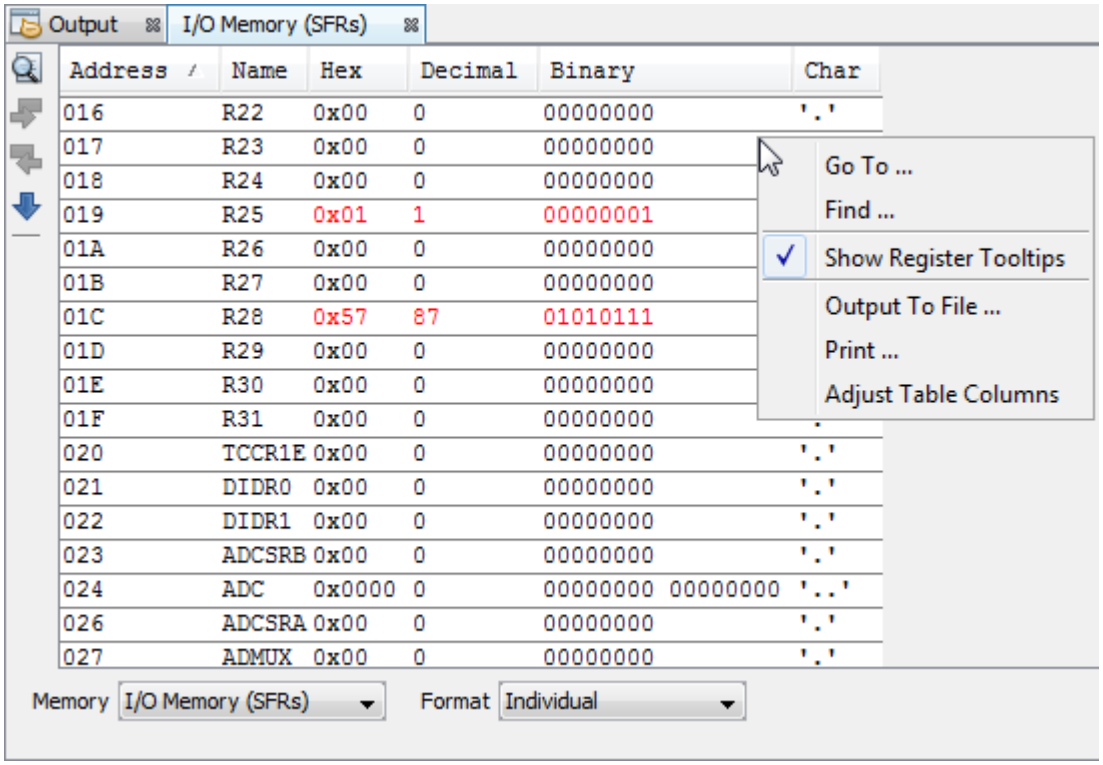
Visible Registers

If a data memory register is not physically implemented on a device, it may not appear in the SFR list. Some tools, such as simulators, may allow you to see registers that do not exist on the actual device.

Single Stepping

If "Freeze Peripherals On Halt" is selected, the I/O port bits in the SFR or the Watches windows will not update when single stepping. The pin will be modified, but the read request to retrieve the new value is blocked by the freeze and cannot be updated until the next step or run command.

Figure 12-13. I/O Memory Window with Content



12.9.5.1 I/O Memory Window Displays

You may specify how memory is displayed in the window by selecting from the Format drop-down box on the bottom of the window.

Individual

In this display, SFRs are listed by address.

Data is displayed in the following columns.

- Address – SFR hexadecimal address.
- Name – Symbolic name for the SFR.
- Radix Information – Hex, Decimal, Binary, Char.
  Radix information is displayed in these four columns. Hex is shown in 1-byte blocks.

Peripheral

In this display, SFRs are grouped according to their related device peripherals.

Data is displayed in the following columns:

- Address – SFR hexadecimal address.
- Name – Name of peripheral or symbolic name for the SFR.
- Radix Information – Hex, Decimal, Binary, Char.
  Radix information is displayed in these four columns. Hex is shown in 1-byte blocks.

Click the **Filter Peripheral** icon to view the SFRs for only the peripherals selected.

CPU Registers

In this display, CPU registers (PC and working registers Rn) are listed by address.

Data is displayed in the following columns:

- Address – SFR hexadecimal address.
- Name – Name of peripheral or symbolic name for the SFR.
- Radix Information – Hex, Decimal, Binary, Char.
  Radix information is displayed in these four columns. Hex is shown in 1-byte blocks.

12.9.5.2 I/O Memory Window Icons

Icons are located on the left side of the window.

Table 12-26. I/O Memory Window Icons

Icon		Function
	Find	Specify a string to find in the window. Select to match whole word or case.
	Find Next	Find next instance of string from Find.
	Find Previous	Find previous instance of string from Find.
	Go To	Go to specified line number or address.
	Filter Peripherals	**Peripheral Format Only** Filter by peripherals available in view. Click icon to open the Select dialog. Select peripherals using Shift-click (range) or Ctrl-click (individually).
	Filter Select All Peripherals.	**Peripheral Format Only** Filter Select All Peripherals.

12.9.5.3 I/O Memory Window Menu

Right click in the memory window data area to pop up this menu.

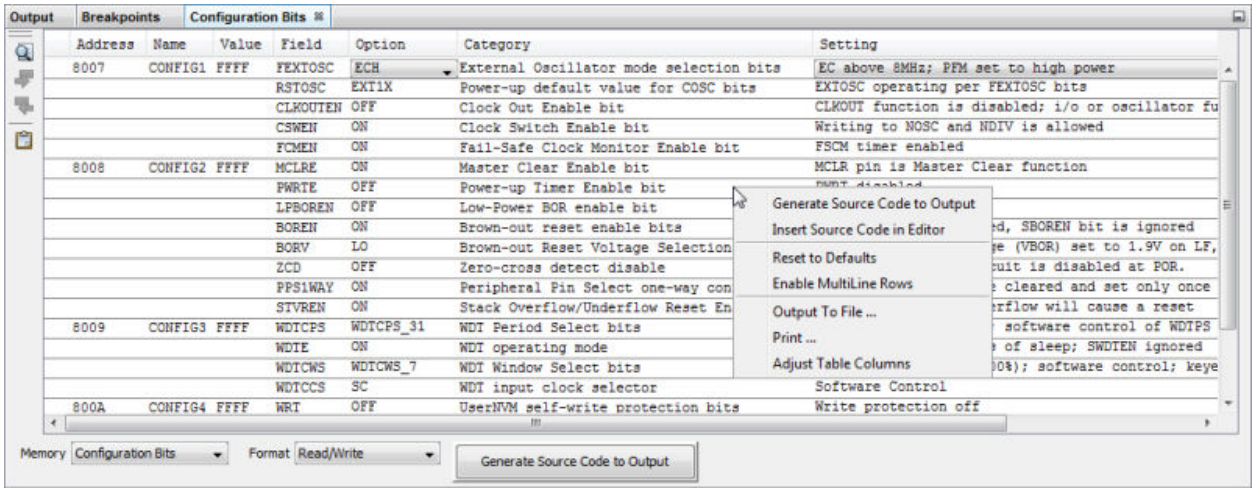
Table 12-27. I/O Memory Window Context Menu

Item	Description
Go To	Go to the address/function specified using the Go To dialog.
Find	Find text specified using the Find dialog.
Show Register Tooltips	Show or hide tooltips for a register. Hover over the register name for pop-up information.
Output To File	Write the displayed window contents to a text file using the Output to File dialog.
Print	Print the contents of this window using the Print dialog. **Note:** If you have a large memory device, the number of pages printed can become very large. A suggestion is to print the window contents to a file (Print dialog, **General** tab, "Print to File" check box) and then select which pages from the file you need to print.
Adjust Table Columns	Adjust the columns automatically.

12.9.6 Configuration Bits Window

Details about using the Configuration Bits window is discussed in 4.19 Set Configuration Values in the Configuration Bits Window.

Figure 12-14. Configuration Bits Window with Content



12.9.6.1 Configuration Bits Window Display

You may specify how memory is displayed in the window by selecting from the Format drop-down box on the bottom of the window. Available formats depend on your device.

When you are happy with the configuration bits settings in the window, you can click the **Generate Source Code to Output** button to write device-specific configuration code to the Output window. You may then cut-and-paste this code into your application in the editor window.

Table 12-28. Configuration Bits Window Display

Column Head	Definition
Address	Address of the configuration word/byte.
Name	Name of the Configuration Register.
Value	Current value of the configuration word/byte.
Field*	For configuration bits set in code, the field portion of the macro. As an example, WDTE is the field portion of the macro _WDTE_OFF.
Option*	For configuration bits set in code, the option portion of the macro. As an example, OFF is the option portion of the macro _WDTE_OFF.
Category	Name of the Configuration bit in the corresponding configuration word/byte.
Setting	Current setting of the Configuration bit. Use the drop-down list to change the setting. The Value of the configuration word/byte will change accordingly.
* Not all devices supported.	

12.9.6.2 Configuration Bits Window Icons

Icons are located on the left side of the window.

Table 12-29. Configuration Bits Window Icons

Icon	Icon Text	Function
	Refresh by Read Device Memory	Same function as the Debug toolbar "Read Device Memory" icon - uploads device memory to the MPLAB X IDE.
	Find	Specify a string to find in the window. Select to match whole word or case.

.....continued

Icon	Icon Text	Function
	Find Next	Find next instance of string from Find.
	Find Previous	Find previous instance of string from Find.
	Paste	Paste Generated Source code into Editor at cursor. If Editor not in focus, Output window shows a warning and Generated Source code placed into this window.

12.9.6.3 Configuration Bits Window Menu

Right click in the memory window data area to pop up this menu. Not all items may be visible for all MCUs.

Table 12-30. Configuration Bits Window Menu

Item	Description
Generate Source Code to Output	Set the configuration bits in the window, as desired. Then select this option to generate code in the Output window, which you can cut-and-paste into your program to set configuration bits. Same function as the button on the window.
Insert Source Code to Editor	Set the configuration bits as desired in the window. Then select this option to generate code into an Editor window that you can cut-and-paste into your program to set configuration bits.
Reset to Defaults	Reset the values of all configuration bits back to their MPLAB X IDE defaults.
Enable Multiline Rows	Allow wrapping of contents if column is shorter than length of contents (e.g., Category). Otherwise contents will be truncated, signified by ellipsis (...).
Output To File	Write the displayed window contents to a text file using the Output to File dialog.
Print	Print the contents of this window using the Print dialog. **Note:** If you have a large memory device, the number of pages printed can become very large. A suggestion is to print the window contents to a file (Print dialog, **General** tab, "Print to File" check box) and then select which pages from the file you need to print.
Adjust Table Columns	Adjust the columns automatically.

12.9.7 EE Data Memory Window

The EEPROM window displays EEPROM data for any microcontroller device that has EEPROM data memory (e.g., PIC16F1829). Data/opcode hex information of the selected device is shown.

For the MPLAB X Simulator, when an EEPROM register value changes or the processor is halted, the data in the EEPROM window is updated.

For any Microchip hardware debug tool (e.g., MPLAB PICkit 4 in-circuit debugger), when an EEPROM register value changes or the processor is halted, the data in the EEPROM window is **not** updated; you must either do a Debug Read of device memory, if the device/header supports this, or you must exit from the debug session and then read the device data.

The start of EEPROM data memory needs to be specified for use with programmers. The table below shows some generic values, but please check the programming specification for your selected device to determine the correct address.

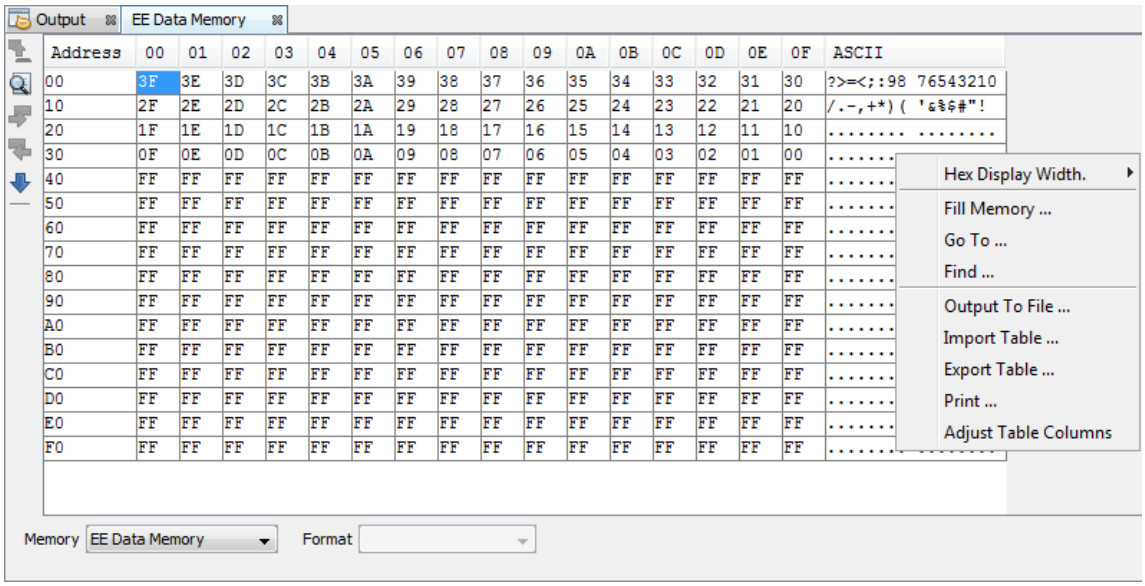
Table 12-31. Programmer – Data EEPROM Start Address

Device	Generic Start Address
Midrange MCUs	0x2100

.....continued

Device	Generic Start Address
Enhanced Midrange MCUs	0x1E000
PIC18F MCUs	0xF00000
PIC24 MCUs, dsPIC DSCs	0x7FFE00

Figure 12-15. EE Data Memory Window with Content



12.9.7.1 EE Data Memory Window Display

This display format shows data in the following columns:

- Address – Hexadecimal address of the data in the next column.
- Data Blocks – Hexadecimal data, shown in 1-, 2- or 4-byte blocks, selectable from the menu.
- ASCII – ASCII representation of the corresponding line of data.

12.9.7.2 EE Data Memory Window Icons

Icons are located on the left side of the window.

Table 12-32. EE Data Memory Window Icons

Icon	Icon Text	Function
	Refresh by Read Device Memory	Same function as the Debug toolbar "Read Device Memory" icon - uploads device memory to the MPLAB X IDE.
	Find	Specify a string to find in the window. Select to match whole word or case.
	Find Next	Find next instance of string from Find.
	Find Previous	Find previous instance of string from Find.
	Go To	Go to specified line number or address.

12.9.7.3 EE Data Memory Window Menu

Right click in the memory window data area set to pop up the menu as in shown below.

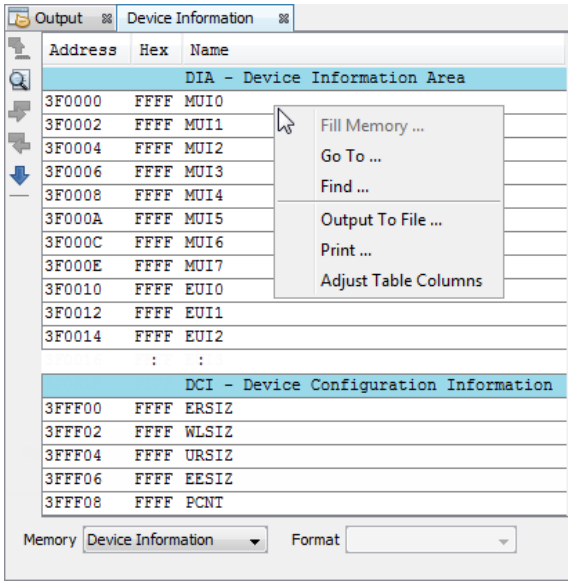
Table 12-33. EE Data Memory Window Context Menu

Item	Description*
Hex Display Width	**Hex Format Only** Set the hexadecimal display width. (Options depend on the device selected).
Fill Memory	Fill memory from Start Address to End Address with the value in Data. Specify other options in the Fill Memory dialog.
Go To	Go to the address/function specified using the Go To dialog.
Find	Find text specified using the Find dialog.
Output To File	Write the displayed window contents to a text file using the Output to File dialog.
Import Table	Import tabular data from a file into a Memory window using the Import Table dialog.
Export Table	Export tabular data from a Memory window into a file using the Export Table dialog.
Print	Print the contents of this window using the Print dialog. **Note:** If you have a large memory device, the number of pages printed can become very large. A suggestion is to print the window contents to a file (Print dialog, General tab, "Print to File" check box) and then select which pages from the file you need to print.
Adjust Table Columns	Adjust the columns automatically.

12.9.8 Device Information Window

The Device Information window displays the contents of the Device Information Area of memory for devices that support this memory area (currently PIC18F24K42 and many PIC16Fxxxxx devices). The DIA contains the calibration data for the internal temperature indicator module, stores the Microchip Unique Identifier words and the Fixed Voltage Reference voltage readings measured in mV. See your device data sheet for details.

Figure 12-16. Device Information Window



12.9.8.1 Device Information Window Display

This display format shows data in the following columns:

- Address – Hexadecimal address of the data in the next column.

- Hex – Hexadecimal data, shown in 1-, 2- or 4-byte blocks, selectable from the menu.
- Name – Description of the corresponding line of data.

12.9.8.2 Device Information Window Icons

Icons are located on the left side of the window.

Table 12-34. Device Information Window Icons

Icon	Icon Text	Function
	Refresh by Read Device Memory	Same function as the Debug toolbar "Read Device Memory" icon - uploads device memory to the MPLAB X IDE.
	Find	Specify a string to find in the window. Select to match whole word or case.
	Find Next	Find next instance of string from Find.
	Find Previous	Find previous instance of string from Find.
	Go To	Go to specified line number or address.

12.9.8.3 Device Information Window Menu

Right click in the memory window data area to pop up this menu. Not all items may be visible for all devices.

Table 12-35. Device Information Window Context Menu

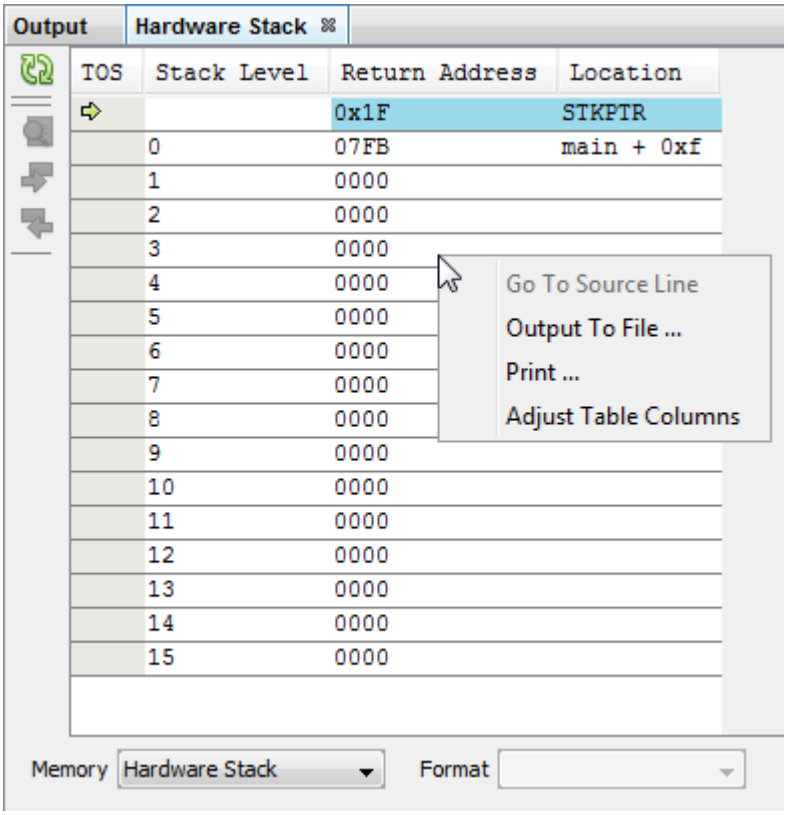
Item	Description
Fill Memory	Fill memory from Start Address to End Address with the value in Data. Specify other options in the Fill Memory dialog.
Go To	Go to the address/function specified using the Go To dialog.
Find	Find text specified using the Find dialog.
Output To File	Write the displayed window contents to a text file using the Output to File dialog.
Print	Print the contents of this window using the Print dialog. **Note:** If you have a large memory device, the number of pages printed can become very large. A suggestion is to print the window contents to a file (Print dialog, **General** tab, "Print to File" check box) and then select which pages from the file you need to print.
Adjust Table Columns	Adjust the columns automatically.

12.9.9 Hardware Stack Window

View the contents of the hardware stack on halt.

To open the hardware stack window, select *Window>PIC Memory Views>Hardware Stack*. The first stack level is denoted by "0".

Figure 12-17. Hardware Stack Window with Content



12.9.9.1 Hardware Stack Window Display

The number of levels of stack depend on the device. This display format shows data in the following columns:

- TOS – Top of Stack current location.
- Stack Level – Location of items on the stack.
- Return Address – Hexadecimal address for return from stack.
- Location – Disassembly of return address code.

When using a hardware tool for debug, some registers may show an "R" for each nibble of data to represent a reserved resource.

12.9.9.2 Hardware Stack Window Icons

The following icons are available on the window on Pause or Halt.

Table 12-36. Hardware Stack Window Icons

Icon	Icon Text	Description
	Auto Refresh Manual Refresh	Auto or Manual Refresh of Window Contents. Depends on the value of "Disable auto refresh for call stack view during debug sessions" under *Tools>Options>Embedded>Generic Settings* – Unchecked = false (default), checked = true. See 12.16.1 Generic Settings Tab. False = Auto Refresh (Green Icon): The window contents are automatically updated on Pause or Halt. If the window is closed or not focused, selecting the window and clicking this button will display the updates without having to run and pause/halt again. True = Manual Refresh (Orange Icon): The window contents are not automatically updated on Pause or Halt. This button must be clicked to update window contents on a pause/halt.

.....continued		
Icon	**Icon Text**	**Description**
	Find	Specify a string to find in the window. Select to match whole word or case.
	Find Next	Find next instance of string from Find.
	Find Previous	Find previous instance of string from Find.

12.9.9.3 Hardware Stack Window Menu

Right click in the memory window data area to pop up this menu..

Table 12-37. Hardware Stack Window Context Menu

Item	Description*
Go To Source Line	Go to the corresponding line in source code in the editor.
Output To File	Write the displayed window contents to a text file using the Output to File dialog.
Print	Print the contents of this window using the Print dialog. **Note:** If you have a large memory device, the number of pages printed can become very large. A suggestion is to print the window contents to a file (Print dialog, **General** tab, "Print to File" check box) and then select which pages from the file you need to print.
Adjust Table Columns	Adjust the columns automatically.

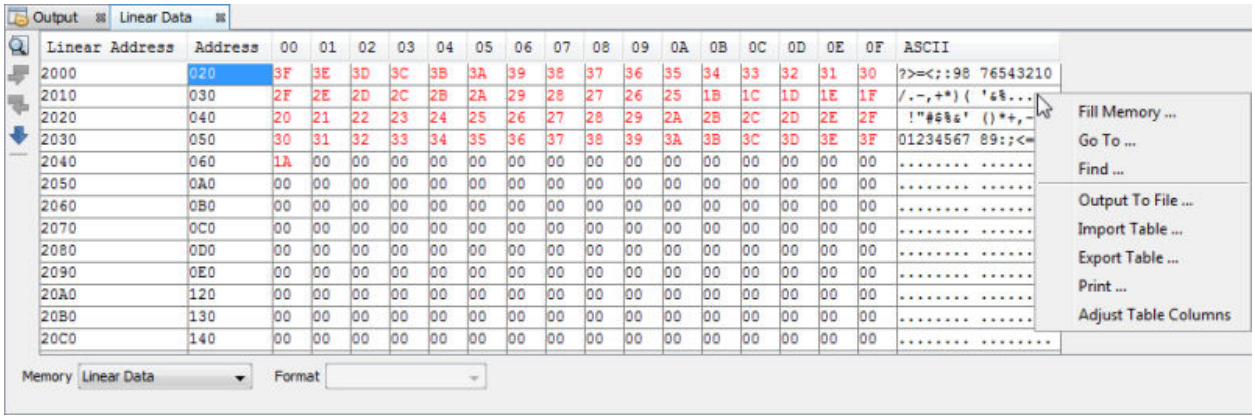
12.9.10 Linear Data Window

The Linear Data Window displays Flash data for any microcontroller device that has this type of memory. Data/opcode hex information of the selected device is shown.

For the MPLAB X Simulator, when a Linear Data register value changes or the processor is halted, the data in the Linear Data window is updated.

For any Microchip hardware debug tool (e.g., MPLAB ICD 4 in-circuit debugger), when a Linear Data register value changes or the processor is halted, the data in the Linear Data window is **not** updated; you must either do a Debug Read of device memory, if the device/header supports this, or you must exit from the debug session and then read the device data.

Figure 12-18. Linear Data Window with Content



12.9.10.1 Linear Data Window Display

This display format shows data in the following columns:

- Address – Hexadecimal address of the data in the next column.

- Data Blocks – Hexadecimal data, shown in 1-, 2- or 4-byte blocks, selectable from the menu.
- ASCII – ASCII representation of the corresponding line of data.

12.9.10.2 Linear Data Window Icons

Icons are located on the left side of the window.

Table 12-38. Linear Data Window Icons

Icon	Icon Text	Function
	Find	Specify a string to find in the window. Select to match whole word or case.
	Find Next	Find next instance of string from Find.
	Find Previous	Find previous instance of string from Find.
	Go To	Go to specified line number or address.

12.9.10.3 Linear Data Window Menu

Right click in the memory window data area set to pop up the menu as in the table below.

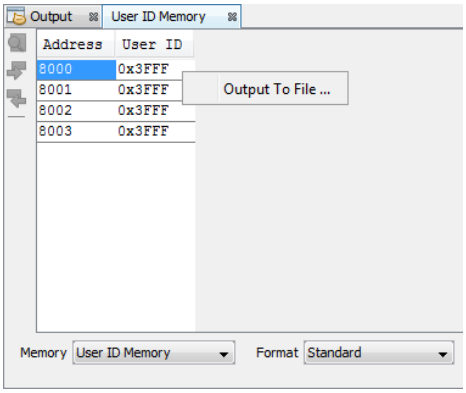
Table 12-39. Linear Data Window Context Menu

Item	Description*
Fill Memory	Fill memory from Start Address to End Address with the value in Data. Specify other options in the Fill Memory dialog.
Go To	Go to the address/function specified using the Go To dialog.
Find	Find text specified using the Find dialog.
Output To File	Write the displayed window contents to a text file using the Output to File dialog.
Import Table	Import tabular data from a file into a Memory window using the Import Table dialog.
Export Table	Export tabular data from a Memory window into a file using the Export Table dialog.
Print	Print the contents of this window using the Print dialog. **Note:** If you have a large memory device, the number of pages printed can become very large. A suggestion is to print the window contents to a file (Print dialog, **General** tab, "Print to File" check box) and then select which pages from the file you need to print.
Adjust Table Columns	Adjust the columns automatically.

12.9.11 User ID Memory Window

Some devices have memory locations where you can store checksum or other code identification (ID) numbers. These locations are readable and writable during program/verify. Depending on the device, they also may be accessible during normal execution through the TBLRD and TBLWT instructions.

Figure 12-19. User ID Memory Window with Content

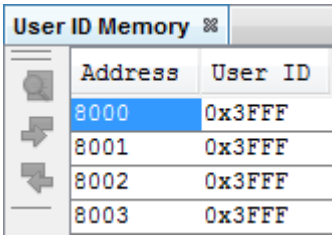


12.9.11.1 User ID Memory Window Display

Consult your device programming specification to determine what values may be entered here. For most devices, this sets the low nibble of the device ID word; the high nibble is set to '0'. The high nibble can be only be written to programmatically, such as by using Table Writes.

You may specify how memory is displayed in the window by selecting from the Format drop-down box on the bottom of the window.

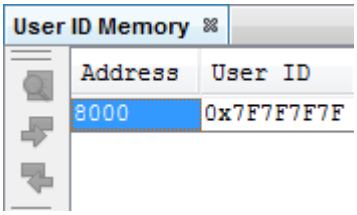
Figure 12-20. Standard Display



In the Standard display, data is shown in the following columns.

- Address – User ID hexadecimal address(es).
- User ID – Contents (in hex) of User ID memory.

Figure 12-21. Legacy Display



In the Legacy display, data is shown in the following columns.

- Address – User ID hexadecimal starting address.
- User ID – Contents (in hex) of User ID memory.

12.9.11.2 User ID Memory Window Icon

Icons are located on the left side of the window.

Table 12-40. User ID Memory Window Icons

Icon	Icon Text	Function
	Find	Specify a string to find in the window. Select to match whole word or case.

.....continued

Icon	Icon Text	Function
	Find Next	Find next instance of string from Find.
	Find Previous	Find previous instance of string from Find.

12.9.11.3 User ID Memory Window Menu

Right click in the memory window data area set to pop up the menu, as in the table below.

Table 12-41. User ID Memory Window Context Menu

Item	Description*
Output To File	Write the displayed window contents to a text file using the Output to File dialog.

12.10 Memory Windows - 32-Bit Devices

Memory windows (*Window>Target Memory Views*) display the many types of device memory, such as SFRs and Configuration bits. Use the "Memory" and "Format" drop-down boxes to customize your window.

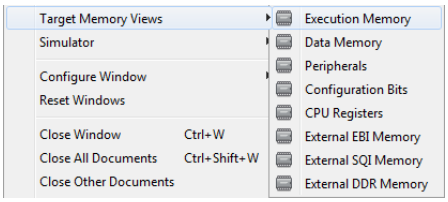
For more on these controls, see 4.18 View or Change Device Memory.

For details on associated dialogs, see 12.11 Memory Windows Associated Dialogs.

Note: The Configuration Bits window and the User ID window are similar for 8-, 16- and 32-bit devices. The only difference is that the values in the windows reflect the program memory available for the device.

Figure 12-22. Window>Target Memory Views - PIC32MX795F512L

Figure 12-23. Window>Target Memory Views - ATSAME70Q21B



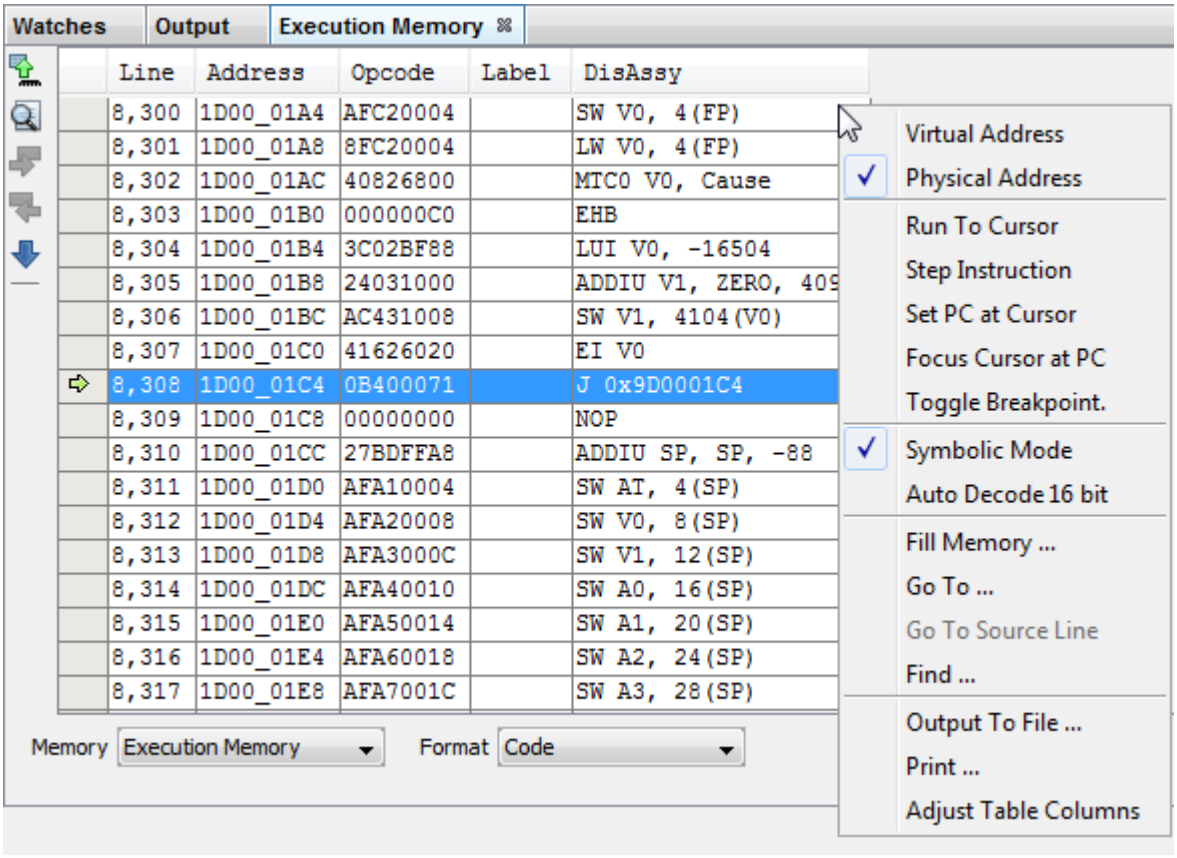
12.10.1 Execution Memory Window

The Execution Memory window displays locations in the range of program and/or data memory for the currently selected 32-bit device.

For the MPLAB X Simulator, when an execution memory value changes or the processor is halted, the data in the Execution Memory window is updated.

For any Microchip hardware debug tool (e.g., MPLAB ICD 4 in-circuit emulator), when an execution memory value changes or the processor is halted, the data in the Execution Memory window is **not** updated; you must do a Read of device memory.

Figure 12-24. Execution Memory Window with Content



12.10.1.1 Execution Memory Window Display

You may specify how memory is displayed in the window by selecting from the Format drop-down box on the bottom of the window.

When using a hardware tool for debug, some registers may show an "R" for each nibble of data to represent a reserved resource.

Code Format

This format displays execution memory information as hex code. The window will have the following columns:

- Line – Reference line number corresponding to memory address.
- Address – Physical hexadecimal address of the opcode.
- Opcode – Hexadecimal opcode, shown in 4-byte blocks. The opcode that is highlighted represents the current location of the program counter.
- Label – Opcode label in symbolic format.
- Disassembly – A disassembled version of the opcode mnemonic.

Data Format

This window will have the following columns:

- Address – Hexadecimal address of the data in the next column.
- Data Blocks – Hexadecimal data, shown in 4-byte blocks.
- ASCII – ASCII representation of the corresponding line of data.

12.10.1.2 Execution Memory Window Icons

Icons are located on the left side of the window.

Table 12-42. Execution Memory Window Icons

Icon	Icon Text	Function
	Refresh by Read Device Memory	Same function as the Debug toolbar "Read Device Memory" icon - uploads device memory to the MPLAB X IDE.
	Find	Specify a string to find in the window. Select to match whole word or case.
	Find Next	Find next instance of string from Find.
	Find Previous	Find previous instance of string from Find.
	Go To	Go to specified line number or address.

12.10.1.3 Execution Memory Window Menu

Right click in the memory window data area set to CODE FORMAT to pop up this menu. Not all items may be visible for all 32-bit MCUs.

Table 12-43. Execution Memory Window Content Menu - Code Format

Item	Description
Virtual Address (KSEG[0:1])	Select virtual addresses for display. See your device data sheet for details.
Physical Address	Select physical addresses for display.
Run to Cursor	Run the program to the current cursor location.
Step Instruction	Perform a step instruction.
Set PC at Cursor	Set the Program Counter (PC) to the cursor location.
Focus Cursor at PC	Move the cursor to the current PC address and centers this address in the window.
Toggle Breakpoint	Toggle (on/off) existing breakpoint.
Symbolic Mode	Display disassembled hex code with symbols.
Auto Decode 16 bit	Use MIP16 / microMIPS instruction decoding. See your device data sheet for details.
Fill Memory	Fill memory from Start Address to End Address with the value in Data. Specify other options in the Fill Memory dialog.
Go To	Go to the address/function specified using the Go To dialog.
Go To Source Line	Go to the corresponding line in source code in the editor.
Find	Find text specified using the Find dialog.
Output To File	Write the displayed window contents to a text file using the Output to File dialog.
Print	Print the contents of this window using the Print dialog. **Note:** If you have a large memory device, the number of pages printed can become very large. A suggestion is to print the window contents to a file (Print dialog, **General** tab, "Print to File" check box) and then select which pages from the file you need to print.
Adjust Table Columns	Adjust the columns automatically.

Right click in the memory window data area set to DATA FORMAT to pop up the menu. Not all items may be visible for all 32-bit MCUs.

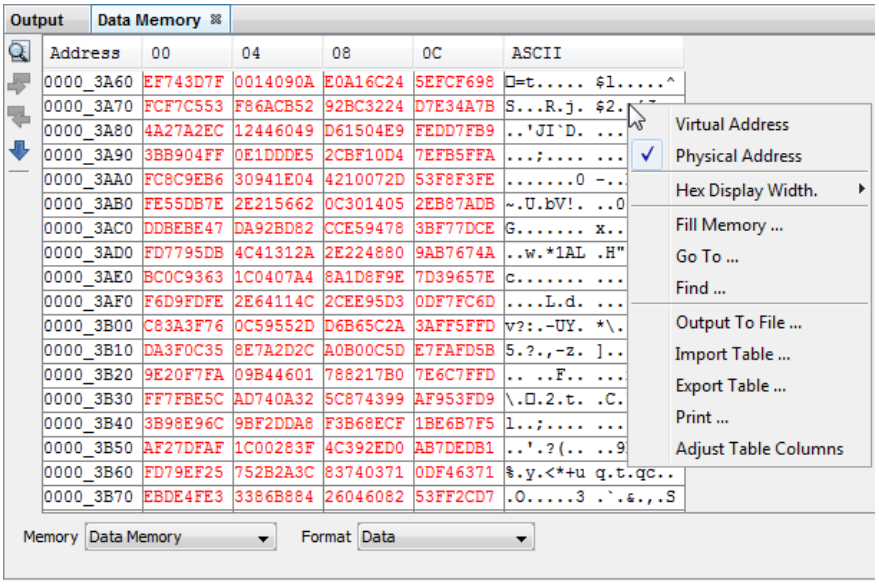
Table 12-44. Execution Memory Window Content Menu - Data Format

Item	Description
Virtual Address (KSEG[0:1])	Select virtual addresses for display. See your device data sheet for details.
Physical Address	Select physical addresses for display.
Hex Width Display	Set the hexadecimal display width. (Options depend on the device selected). Example: One byte, e.g., 00 01 02 ... 0E 0F Two bytes, e.g., 00 02 04 ... 0C 0E Four bytes, e.g., 00 04 08 0C
Fill Memory	Fill memory from Start Address to End Address with the value in Data. Specify other options in the Fill Memory dialog.
Go To	Go to the address/function specified using the Go To dialog.
Find	Find text specified using the Find dialog.
Output To File	Write the displayed window contents to a text file using the Output to File dialog.
Import Table	Import tabular data from a file into a Memory window using the Import Table dialog.
Export Table	Export tabular data from a Memory window into a file using the Export Table dialog.
Print	Print the contents of this window using the Print dialog. **Note:** If you have a large memory device, the number of pages printed can become very large. A suggestion is to print the window contents to a file (Print dialog, **General** tab, "Print to File" check box) and then select which pages from the file you need to print.
Adjust Table Columns	Adjust the columns automatically.

12.10.2 Data Memory Window

The Data Memory window displays locations in the range of data and/or program memory for the currently selected 32-bit device.

Figure 12-25. Data Memory Window with Content



12.10.2.1 Data Memory Window Display

You may specify how memory is displayed in the window by selecting from the Format drop-down box on the bottom of the window.

When using a hardware tool for debug, some registers may show an "R" for each nibble of data to represent a reserved resource.

Data Format

This window will have the following columns:

- Address – Hexadecimal address of the data in the next column.
- Data Blocks – Hexadecimal data, shown in 4-byte blocks.
- ASCII – ASCII representation of the corresponding line of data.

Code Format

This window will have the following columns:

- Line – Reference line number corresponding to memory address.
- Address – Physical hexadecimal address of the opcode.
- Opcode – Hexadecimal opcode, shown in 4-byte blocks
  The opcode that is highlighted represents the current location of the program counter.
- Label – Opcode label in symbolic format.
- Disassembly – A disassembled version of the opcode mnemonic.

12.10.2.2 Data Memory Window Icons

Icons are located on the left side of the window.

Table 12-45. Data Memory Window Icons

Icon	Icon Text	Function
	Refresh by Read Device Memory	Same function as the Debug toolbar "Read Device Memory" icon - uploads device memory to the MPLAB X IDE.
	Find	Specify a string to find in the window. Select to match whole word or case.
	Find Next	Find next instance of string from Find.

.....continued

Icon	Icon Text	Function
	Find Previous	Find previous instance of string from Find.
	Go To	Go to specified line number or address.

12.10.2.3 Data Memory Window Menu

Right click in the memory window data area to pop up this menu. Not all items may be visible for all 32-bit MCUs.

Table 12-46. Data Memory Window Content Menu

Item	Description
Virtual Address	Select virtual addresses for display. See your device data sheet for details.
Physical Address	Select physical addresses for display.
Symbolic Mode	**Code Format Only** Display disassembled hex code with symbols.
Hex Width Display	**Data Format Only** Set the hexadecimal display width. (Options depend on the device selected). Example: One byte, e.g., 00 01 02 ... 0E 0F Two bytes, e.g., 00 02 04 ... 0C 0E Four bytes, e.g., 00 04 08 0C
Fill Memory	Fill memory from Start Address to End Address with the value in Data. Specify other options in the Fill Memory dialog.
Go To	Go to the address/function specified using the Go To dialog.
Go To Source Line	**Code Format Only** Go to the corresponding line in source code in the editor.
Find	Find text specified using the Find dialog.
Output To File	Write the displayed window contents to a text file using the Output to File dialog.
Import Table	Import tabular data from a file into a Memory window using the Import Table dialog.
Export Table	Export tabular data from a Memory window into a file using the Export Table dialog.
Print	Print the contents of this window using the Print dialog. **Note:** If you have a large memory device, the number of pages printed can become very large. A suggestion is to print the window contents to a file (Print dialog, **General** tab, "Print to File" check box) and then select which pages from the file you need to print.
Adjust Table Columns	Adjust the columns automatically.

12.10.3 Peripherals Window

The Peripherals window displays the contents of the SFRs that relate to the device peripherals. To view only a few SFRs, you may prefer to use a Watch window, which may help with speed issues when using hardware debug tools (i.e., faster window update rate).

Whenever a break occurs, the contents of the SFRs are updated.

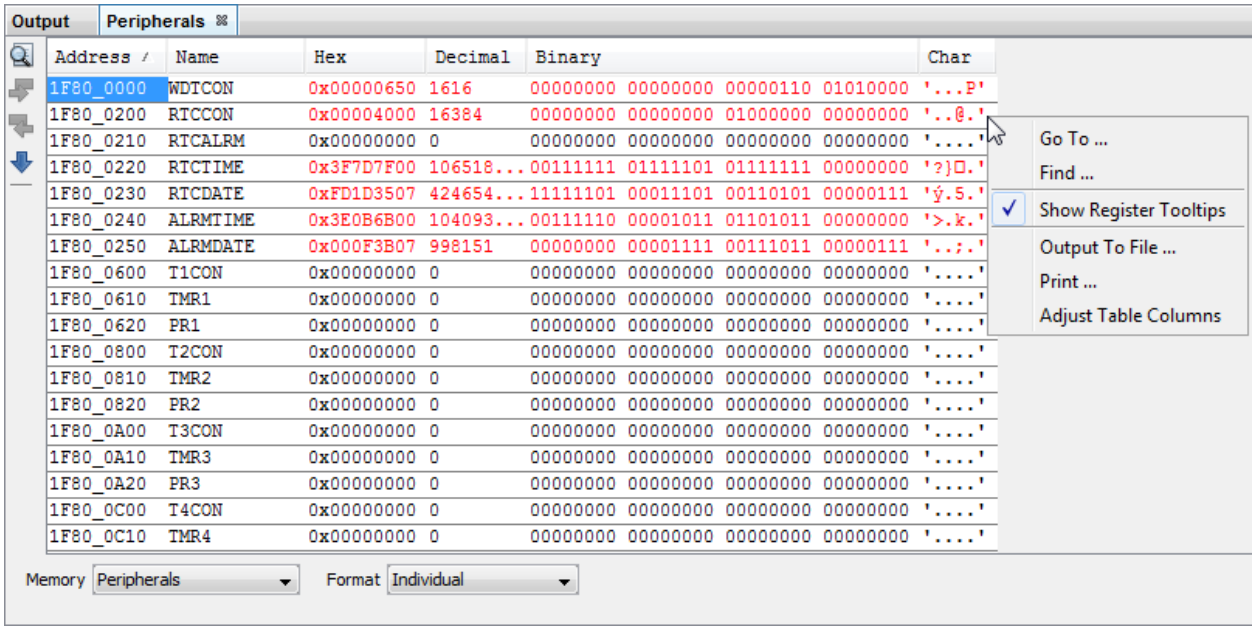
Visible Registers

If a data memory register is not physically implemented on a device, it may not appear in the SFR list. Some tools, such as simulators, may allow you to see registers that do not exist on the actual device, such as prescalers.

Single Stepping

If "Freeze Peripherals On Halt" is selected, the I/O port bits in the SFR or the Watch windows will not update when single stepping. The pin will be modified, but the read request to retrieve the new value is blocked by the freeze and cannot be updated until the next step or run command.

Figure 12-26. Peripherals Window with Content



12.10.3.1 Peripherals Window Display

You may specify how memory is displayed in the window by selecting from the Format drop-down box on the bottom of the window.

When using a hardware tool for debug, some registers may show an "R" for each nibble of data to represent a reserved resource.

- **Individual** - View all peripheral registers together.
- **Groups** - View peripheral registers in functional groups.

Data is displayed for both formats in the following columns.

- Address – SFR physical hexadecimal address.
- Virtual – SFR virtual hexadecimal address as defined by the Bus Matrix.
- Name – Symbolic name for the SFR.
- Radix Information – Hex, Decimal, Binary, Char.
  You may add radix information to the display by right clicking on the column header bar. Hex is shown in 4-byte blocks.

12.10.3.2 Peripherals Window Icons

Icons are located on the left side of the window.

Table 12-47. Data Memory Window Icons

Icon	Icon Text	Function
	Find	Specify a string to find in the window. Select to match whole word or case.

.....continued

Icon	Icon Text	Function
	Find Next	Find next instance of string from Find.
	Find Previous	Find previous instance of string from Find.
	Go To	Go to specified line number or address.

12.10.3.3 Peripherals Window Menu

Right click in the memory window data area set to pop up the menu as in the table below.

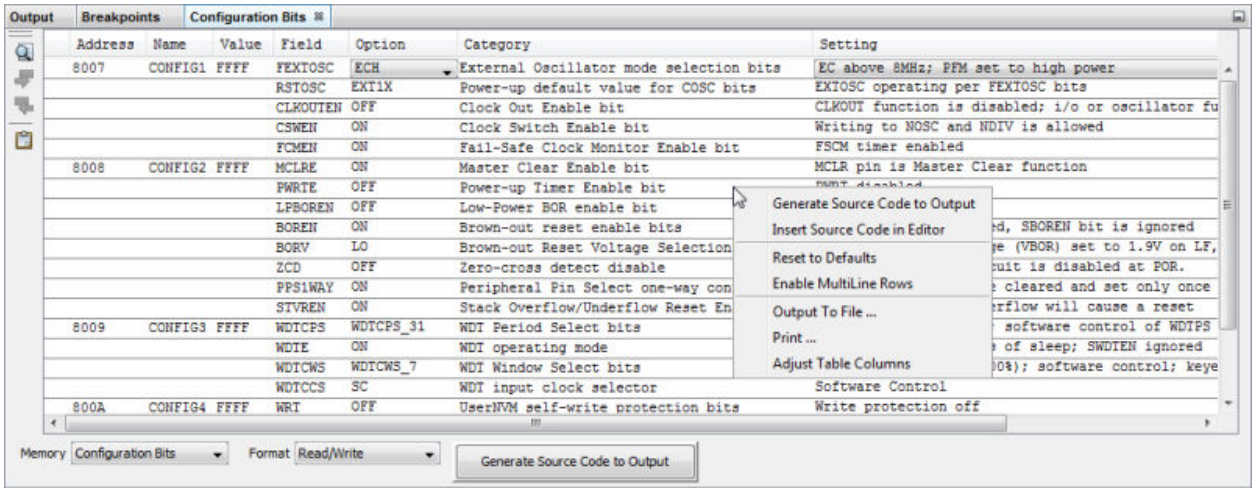
Table 12-48. Peripherals Window Context Menu

Item	Description
Find	Find text specified using the Find dialog.
Output To File	Write the displayed window contents to a text file using the Output to File dialog.
Print	Print the contents of this window using the Print dialog. **Note:** If you have a large memory device, the number of pages printed can become very large. A suggestion is to print the window contents to a file (Print dialog, **General** tab, "Print to File" check box) and then select which pages from the file you need to print.
Adjust Table Columns	Adjust the columns automatically.

12.10.4 Configuration Bits Window

Details about using the Configuration Bits window is discussed in 4.19 Set Configuration Values in the Configuration Bits Window.

Figure 12-27. Configuration Bits Window with Content



12.10.4.1 Configuration Bits Window Display

You may specify how memory is displayed in the window by selecting from the Format drop-down box on the bottom of the window. Available formats depend on your device.

When you are happy with the configuration bits settings in the window, you can click the **Generate Source Code to Output** button to write device-specific configuration code to the Output window. You may then cut-and-paste this code into your application in the editor window.

Table 12-49. Configuration Bits Window Display

Column Head	Definition
Address	Address of the configuration word/byte.
Name	Name of the Configuration Register.
Value	Current value of the configuration word/byte.
Field*	For configuration bits set in code, the field portion of the macro. As an example, WDTE is the field portion of the macro _WDTE_OFF.
Option*	For configuration bits set in code, the option portion of the macro. As an example, OFF is the option portion of the macro _WDTE_OFF.
Category	Name of the Configuration bit in the corresponding configuration word/byte.
Setting	Current setting of the Configuration bit. Use the drop-down list to change the setting. The Value of the configuration word/byte will change accordingly.
* Not all devices supported.	

12.10.4.2 Configuration Bits Window Icons

Icons are located on the left side of the window.

Table 12-50. Configuration Bits Window Icons

Icon	Icon Text	Function
	Refresh by Read Device Memory	Same function as the Debug toolbar "Read Device Memory" icon - uploads device memory to the MPLAB X IDE.
	Find	Specify a string to find in the window. Select to match whole word or case.
	Find Next	Find next instance of string from Find.
	Find Previous	Find previous instance of string from Find.
	Paste	Paste Generated Source code into Editor at cursor. If Editor not in focus, Output window shows a warning and Generated Source code placed into this window.

12.10.4.3 Configuration Bits Window Menu

Right click in the memory window data area to pop up this menu. Not all items may be visible for all MCUs.

Table 12-51. Configuration Bits Window Menu

Item	Description
Generate Source Code to Output	Set the configuration bits in the window, as desired. Then select this option to generate code in the Output window, which you can cut-and-paste into your program to set configuration bits. Same function as the button on the window.
Insert Source Code to Editor	Set the configuration bits as desired in the window. Then select this option to generate code into an Editor window that you can cut-and-paste into your program to set configuration bits.
Reset to Defaults	Reset the values of all configuration bits back to their MPLAB X IDE defaults.
Enable Multiline Rows	Allow wrapping of contents if column is shorter than length of contents (e.g., Category). Otherwise contents will be truncated, signified by ellipsis (...).

.....continued

Item	Description
Output To File	Write the displayed window contents to a text file using the Output to File dialog.
Print	Print the contents of this window using the Print dialog. **Note:** If you have a large memory device, the number of pages printed can become very large. A suggestion is to print the window contents to a file (Print dialog, **General** tab, "Print to File" check box) and then select which pages from the file you need to print.
Adjust Table Columns	Adjust the columns automatically.

12.10.5 CPU Registers Window

The CPU Registers window displays the contents of the SFRs that relate to the device CPU. To view only a few CPU SFRs, you may prefer to use a Watches window, which may help with speed issues when using hardware debug tools (i.e., faster window update rate).

Whenever a break occurs, the contents of the CPU Registers are updated.

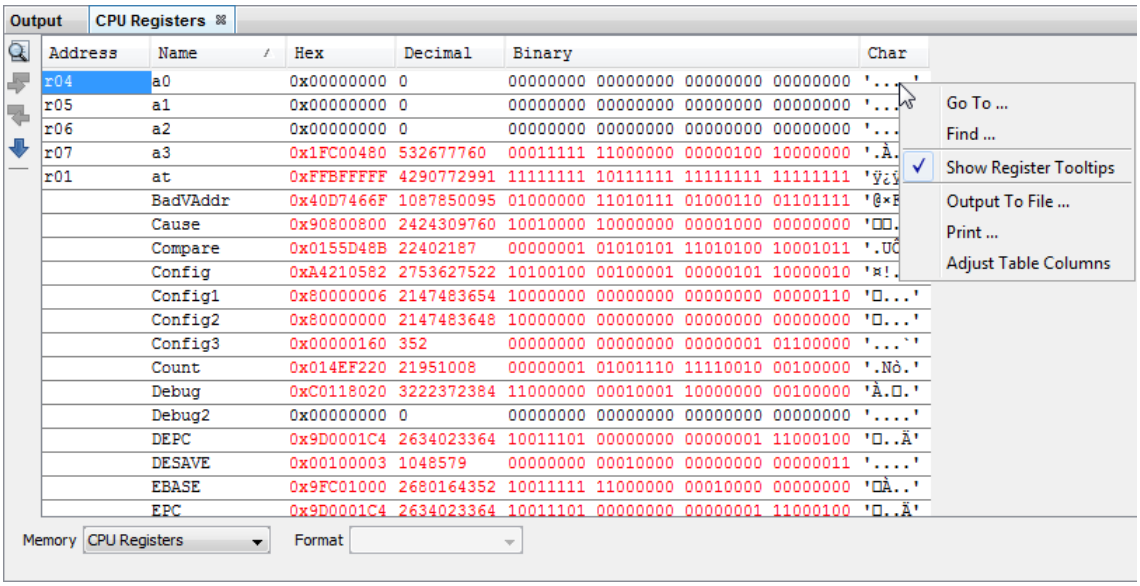
Visible Registers

If a data memory register is not physically implemented on a device, it may not appear in the SFR list. Some tools, such as simulators, may allow you to see registers that do not exist on the actual device, such as prescalers.

Single Stepping

If "Freeze Peripherals On Halt" is selected, the I/O port bits in the SFR or the Watches windows will not update when single stepping. The pin will be modified, but the read request to retrieve the new value is blocked by the freeze and cannot be updated until the next step or run command.

Figure 12-28. CPU Registers Window with Content



12.10.5.1 CPU Registers Window Display

You may specify how memory is displayed in the window by selecting from the Format drop-down box on the bottom of the window. Choices are:

- Blank - Only one view.
- Devices with FPU - For PIC32 devices with a Floating-Point Unit (FPU), the Format options are:
  - All Registers
  - Only CPU Registers

– Only FPU Registers.

Blank, All Registers or Only CPU Registers Format

Data is displayed in the following columns.

- Address – SFR physical hexadecimal address.
- Name – Symbolic name for the SFR.
- Radix Information – Hex, Decimal, Binary, Char.
- Virtual – SFR virtual hexadecimal address as defined by the Bus Matrix.

Only FPU Registers Format

Data is displayed in the following columns.

- Address – SFR physical hexadecimal address.
- Name – Symbolic name for the SFR.
- Radix Information – Hex
- Floating Point – Float, Double

12.10.5.2 CPU Registers Window Icons

Icons are located on the left side of the window.

Table 12-52. Data Memory Window Icons

Icon	Icon Text	Function
	Find	Specify a string to find in the window. Select to match whole word or case.
	Find Next	Find next instance of string from Find.
	Find Previous	Find previous instance of string from Find.
	Go To	Go to specified line number or address.

12.10.5.3 CPU Registers Window Menu

Right click in the memory window data area set to pop up the menu as in the table below.

Table 12-53. CPU Registers Window Context Menu

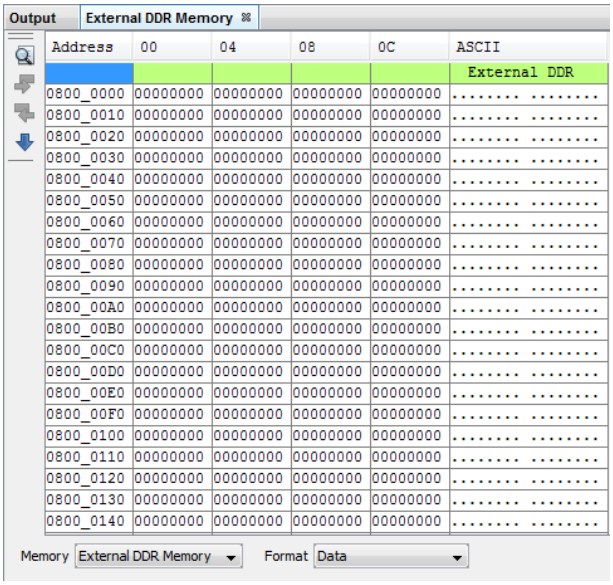
Item	Description
Go To	Go to the address/function specified using the Go To dialog.
Find	Find text specified using the Find dialog.
Show Register Tooltip	Show or hide tooltips for a register. Hover over the register name for pop-up information.
Output To File	Write the displayed window contents to a text file using the Output to File dialog.
Print	Print the contents of this window using the Print dialog. **Note:** If you have a large memory device, the number of pages printed can become very large. A suggestion is to print the window contents to a file (Print dialog, **General** tab, "Print to File" check box) and then select which pages from the file you need to print.
Adjust Table Columns	Adjust the columns automatically.

12.10.6 External EBI, SQI, or DDR Memory Window

The External Memory window displays the contents of external memory by bus type:

- External Bus Interface (EBI)
- Serial Quad Interface (SQI)
- Double Data Rate (DDR)

Figure 12-29. External SSR Memory Window



12.10.6.1 External EBI, SQI, or DDR Memory Window Display

To see external memory contents in this window:

- Allocate functions/variables to external memory in your code - see the "*MPLAB XC32/XC32++ C Compiler User's Guide*" (DS50001686) for details.
- Load symbols - *File>Properties*, "Loading" category, check "Load symbols when programming or building for production (slows process)."
- Compile your code - Build the project.

You may specify how memory is displayed in the window by selecting from the Format drop-down box on the bottom of the window:

Code Format

This window will have the following columns:

- Line – Reference line number corresponding to memory address.
- Address – Physical hexadecimal address of the opcode.
- Opcode – Hexadecimal opcode, shown in 4-byte blocks
  The opcode that is highlighted represents the current location of the program counter.
- Label – Opcode label in symbolic format.
- Disassembly – A disassembled version of the opcode mnemonic.

Data Format

This window will have the following columns:

- Address – Hexadecimal address of the data in the next column.
- Data Blocks – Hexadecimal data, shown in 4-byte blocks.
- ASCII – ASCII representation of the corresponding line of data.

12.10.6.2 External EBI, SQI, or DDR Memory Window Icons

Icons are located on the left side of the window.

Table 12-54. External Memory Window Icons

Icon	Icon Text	Function
	Refresh by Read Device Memory	Same function as the Debug toolbar "Read Device Memory" icon - uploads device memory to the MPLAB X IDE.
	Find	Specify a string to find in the window. Select to match whole word or case.
	Find Next	Find next instance of string from Find.
	Find Previous	Find previous instance of string from Find.
	Go To	Go to specified line number or address.

12.10.6.3 External EBI, SQI, or DDR Memory Window Menu

Right click in the memory window data area set to CODE FORMAT to pop up this menu. Not all items may be visible for all 32-bit MCUs.

Table 12-55. Execution Memory Window Content Menu - Code Format

Item	Description
External Address (KSEG[2:3])	Select addresses for display. See your device data sheet for details.
Physical Address	Select physical addresses for display.
Run to Cursor	Run the program to the current cursor location.
Step Instruction	Perform a step instruction.
Set PC at Cursor	Set the Program Counter (PC) to the cursor location.
Focus Cursor at PC	Move the cursor to the current PC address and centers this address in the window.
Toggle Breakpoint	Toggle (on/off) existing breakpoint.
Symbolic Mode	Display disassembled hex code with symbols.
Auto Decode 16 bit	Use MIP16 / microMIPS instruction decoding. See your device data sheet for details.
Fill Memory	Fill memory from Start Address to End Address with the value in Data. Specify other options in the Fill Memory dialog.
Go To	Go to the address/function specified using the Go To dialog.
Go To Source Line	Go to the corresponding line in source code in the editor.
Find	Find text specified using the Find dialog.
Output To File	Write the displayed window contents to a text file using the Output to File dialog.
Print	Print the contents of this window using the Print dialog. **Note:** If you have a large memory device, the number of pages printed can become very large. A suggestion is to print the window contents to a file (Print dialog, **General** tab, "Print to File" check box) and then select which pages from the file you need to print.
Adjust Table Columns	Adjust the columns automatically.

Right click in the memory window data area set to DATA FORMAT to pop up the menu. Not all items may be visible for all 32-bit MCUs.

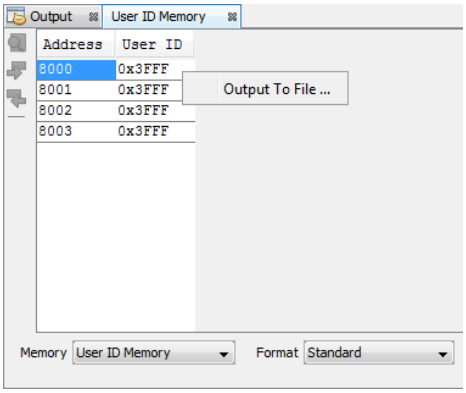
Table 12-56. Execution Memory Window Content Menu - Data Format

Item	Description
External Address (KSEG[2:3])	Select addresses for display. See your device data sheet for details.
Physical Address	Select physical addresses for display.
Hex Width Display	Set the hexadecimal display width (Options depend on the device selected.) Example: One byte, e.g., 00 01 02 ... 0E 0F Two bytes, e.g., 00 02 04 ... 0C 0E Four bytes, e.g., 00 04 08 0C
Fill Memory	Fill memory from Start Address to End Address with the value in Data. Specify other options in the Fill Memory dialog.
Go To	Go to the address/function specified using the Go To dialog.
Find	Find text specified using the Find dialog.
Output To File	Write the displayed window contents to a text file using the Output to File dialog.
Import Table	Import tabular data from a file into a Memory window using the Import Table dialog.
Export Table	Export tabular data from a Memory window into a file using the Export Table dialog.
Print	Print the contents of this window using the Print dialog. **Note:** If you have a large memory device, the number of pages printed can become very large. A suggestion is to print the window contents to a file (Print dialog, **General** tab, "Print to File" check box) and then select which pages from the file you need to print.
Adjust Table Columns	Adjust the columns automatically.

12.10.7 User ID Memory Window

Some devices have memory locations where you can store checksum or other code identification (ID) numbers. These locations are readable and writable during program/verify. Depending on the device, they also may be accessible during normal execution through the TBLRD and TBLWT instructions.

Figure 12-30. User ID Memory Window with Content

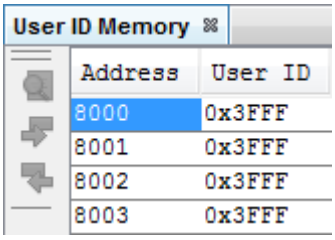


12.10.7.1 User ID Memory Window Display

Consult your device programming specification to determine what values may be entered here. For most devices, this sets the low nibble of the device ID word; the high nibble is set to '0'. The high nibble can be only be written to programmatically, such as by using Table Writes.

You may specify how memory is displayed in the window by selecting from the Format drop-down box on the bottom of the window.

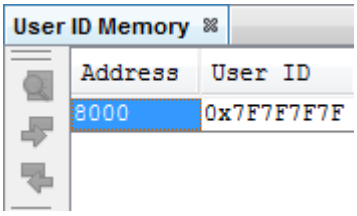
Figure 12-31. Standard Display



In the Standard display, data is shown in the following columns.

- Address – User ID hexadecimal address(es).
- User ID – Contents (in hex) of User ID memory.

Figure 12-32. Legacy Display



In the Legacy display, data is shown in the following columns.

- Address – User ID hexadecimal starting address.
- User ID – Contents (in hex) of User ID memory.

12.10.7.2 User ID Memory Window Icon

Icons are located on the left side of the window.

Table 12-57. User ID Memory Window Icons

Icon	Icon Text	Function
	Find	Specify a string to find in the window. Select to match whole word or case.
	Find Next	Find next instance of string from Find.
	Find Previous	Find previous instance of string from Find.

12.10.7.3 User ID Memory Window Menu

Right click in the memory window data area set to pop up the menu, as in the table below.

Table 12-58. User ID Memory Window Context Menu

Item	Description*
Output To File	Write the displayed window contents to a text file using the Output to File dialog.

12.11 Memory Windows Associated Dialogs

Memory windows have associated dialogs to support formatting, management and output of window data. These dialogs are accessed from content menus.

12.11.1 Memory Windows Go To Dialog

Go to a location in target memory. Access this dialog from some target memory window context menus.

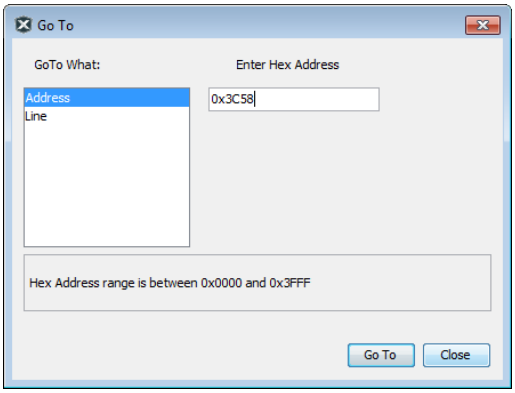


Table 12-59. Go To Dialog

Item	Description
Go To What	Enter what you want to go to. Click on the item to see a description below. **Address**: Enter a hexadecimal address. **Symbol**: Enter a program symbol. **Line**: Enter a line number. **Label**: Select a program label. **Function**: Select a program function.
Go to description	Select a "Go To What" item to see a description.

12.11.2 Memory Windows Find Dialog

Find a location in target memory. Access this dialog from some target memory window context menus.

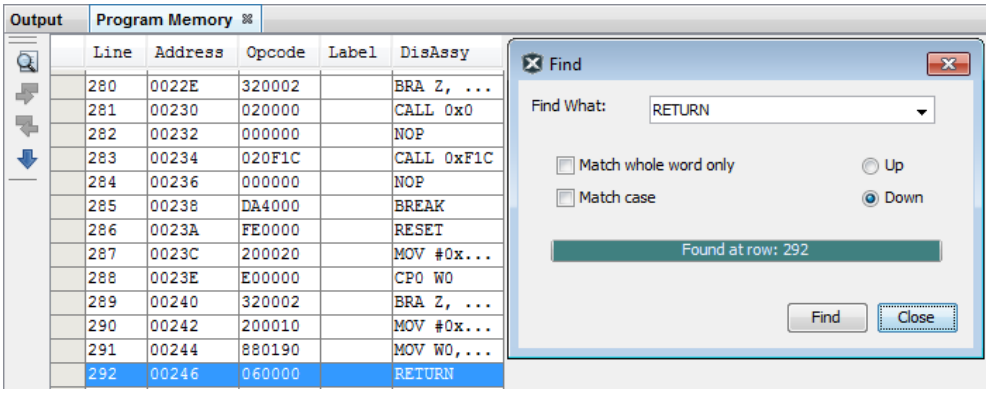


Table 12-60. Find Dialog

Item	Description
Find What	Enter the text you want to find.
Match	Select to "Match whole word only" and/or "Match case."
Direction	Select to search "Up" or "Down" the window.
Found At bar	When an item is found, the location is shown on this bar.

12.11.3 Memory Windows Output to File Dialog

Output specified memory range to a file. Access this dialog from some target memory window context menus.

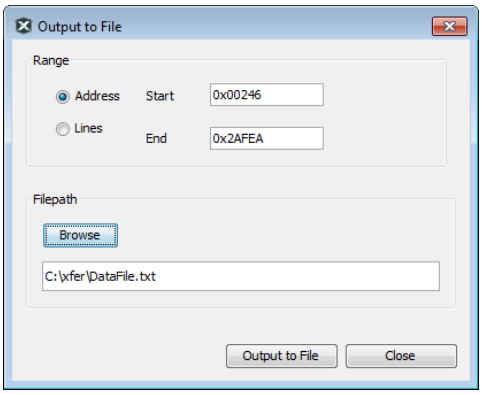


Table 12-61. Output to File Dialog

Item	Description
Range	Enter a range of data to output to a file. **Address**: Enter an Address range - Start to End Address. **Lines**: Enter a range of Lines - Start line to End line.
Filepath	Browse to a location for the output file or enter in the text box.

12.11.4 Memory Windows Import/Export Table Dialog

Import from or export to a file target memory in tabular format. Access this dialog from some target memory window context menus.

Figure 12-33. Export Table Dialog

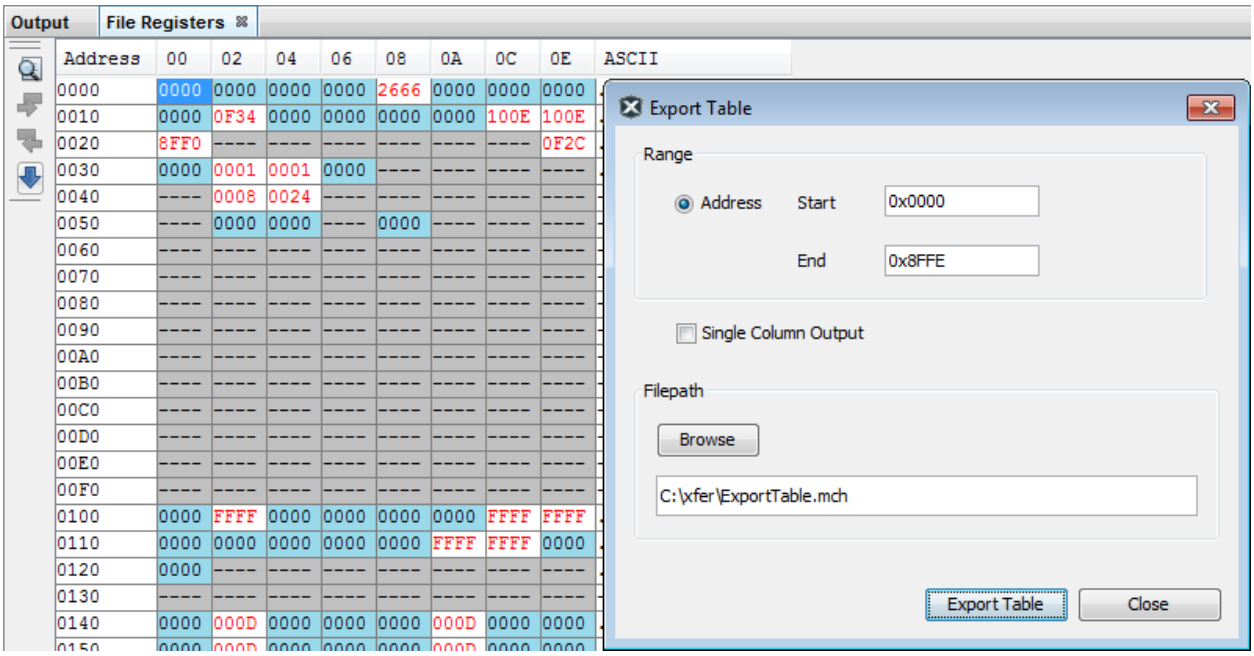


Figure 12-34. Export Table File

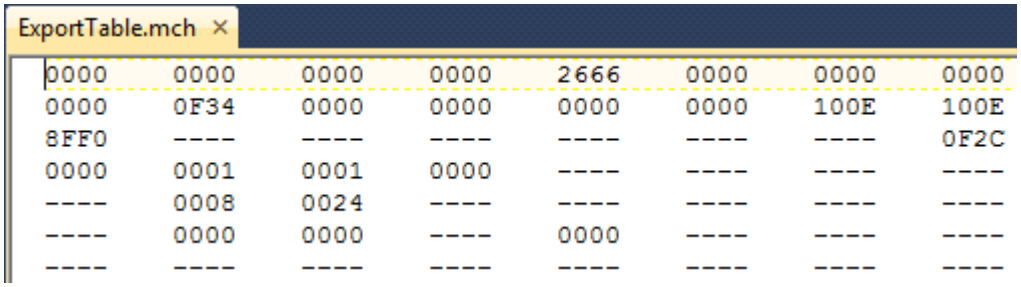


Table 12-62. Import/Export Table Dialog

Item	Description
Range	Enter a range of data to import or export in tabular format to a file. **Address**: Enter an Address range - Start to End Address.
Single Column Format	**Export** table as a single column.
Filepath	Browse to a location for the input or output file or enter in the text box.

12.11.5 Memory Windows Print Dialog

Print out the memory window contents. Access this dialog from some target memory window context menus.

Print options available may depend on printer selected.

Figure 12-35. Print Dialog

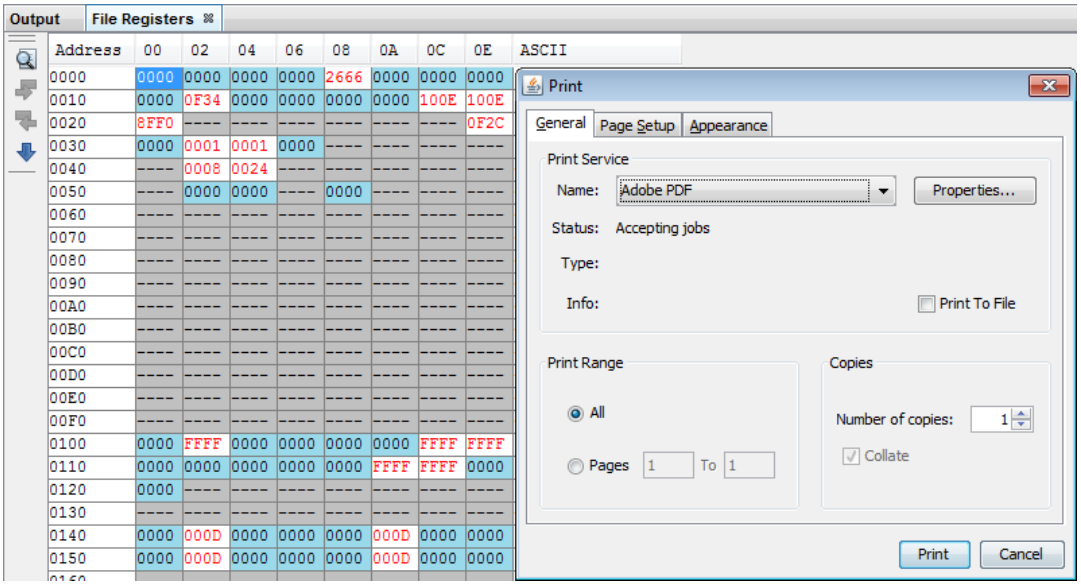


Figure 12-36. Print Dialog Output

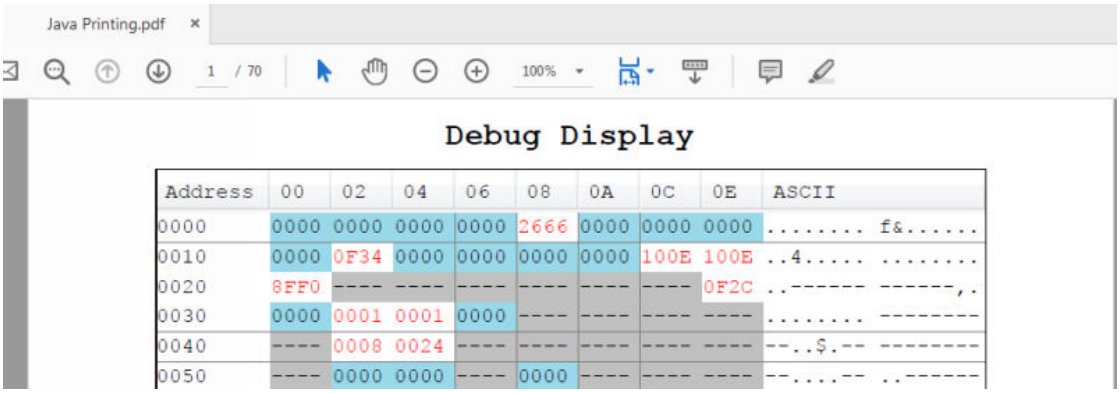


Table 12-63. Print Dialog

Item	Description
General Tab	Set up general print options. **Print Service**: Select a printer/print service. Click Properties to set up the print service. Also, check to print to a file. **Print Range**: Range of pages to print. **Copies**: Number of copies to print.
Page Setup	Set up page options. **Media**: Specify page size and source. **Orientation**: Specify page orientation. **Margin**: Specify page margins.
Appearance	Set up appearance of print out. **Color appearance**: Specify color or monochrome. **Quality**: Specify draft, normal or high. **Sides**: Specify sides and multi-side orientation. **Job Attributes**: Print a banner, specify job name and user.

12.12 Message Center

Open the Message Center window from *Window>Debugging>Output>Message Center*.

The Message Center is a window that collects all the output window messages and then presents them chronologically into one view. Different filters can be selected to display different relevant messages.

Figure 12-37. Message Center Window

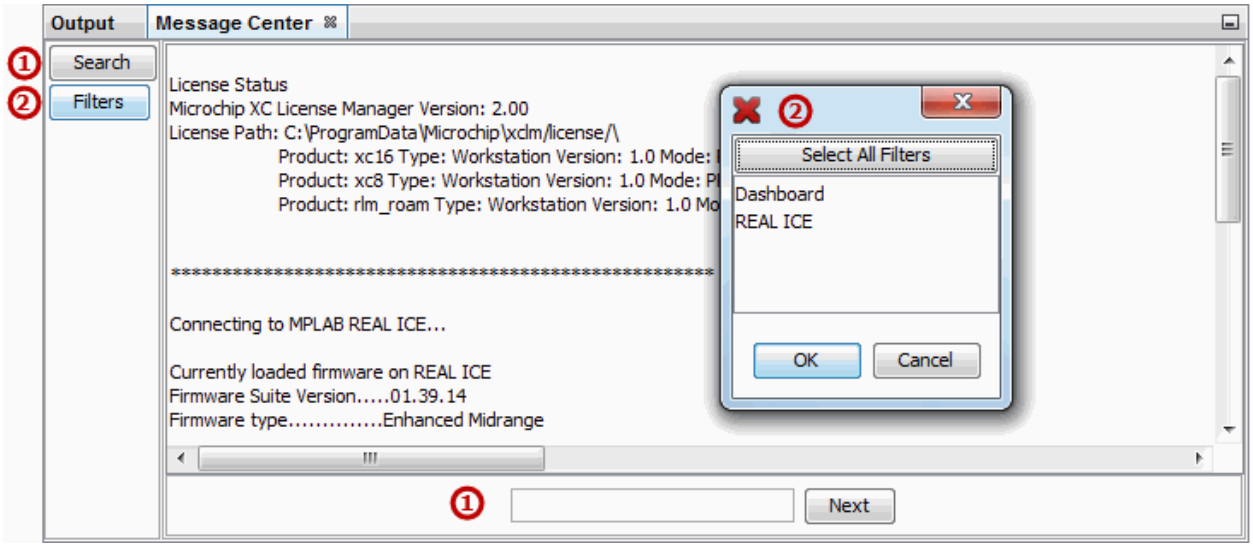


Table 12-64. Message Center Buttons

Number	Button	Function
1	Search	Toggle the search text box. When the search box is visible, enter text to search for that text string in the Message Center window.
2	Filters	By default, all messages that appear in the Output window are displayed in the Message Center window. Click this button to open a window with a list of Output window tabbed items. Filter messages displayed by selecting one or more from the list.

12.13 Output Window

Open this window by selecting *Window>Output*.

The Task pane contains many windows, some inherited from the NetBeans platform and some that are specific to MPLAB X IDE. The Output window contains the MPLAB X IDE output information. It is shown on tabs within the window.

Table 12-65. Output Window Tab Items

Item	Description
Debugger Console	Shows main debug actions, such as "User program running."
Tool-specific	Shows tool firmware version, device ID, and action status.
Build, Load	Shows information and status on the build, and program loading.
Clean, Build, Load	Shows information and status on the clean, build, and program loading.
Peripheral Output	Shows technical output from peripherals such as the UART, with the Simulator as the debug tool.

12.13.1 Content Scrolling

Programming actions taking place will be shown and scrolled in an output window. The last action executed will be shown at the bottom of the window.

If you scroll up to view the content and then program again, the cursor will not move from your current location. This is to keep you from losing your place even though more content is filing the window below. To see the new content, you must manually scroll down.

12.13.2 Context Menu

Right clicking in the Output window will display various options as shown below.

Table 12-66. Output Window Context Menu

Item	Description
Copy	Copies selected text from the Output window to the clipboard.
Paste	Pastes selected text from the clipboard to the Output window.
Find	Finds the selected text, or enters other text to find, in the Output window You may use regular expressions and match case.
Find Next	Finds the next occurrence of the Find text.
Find Previous	Finds the previous occurrence of the Find text.
Filter	Filters the output by text or regular expression.
Wrap Text	Wraps the text in the Output window.
Larger Font	Makes the font larger.
Smaller Font	Makes the font smaller.
Choose Font	Selects the font type, style and size.
Save As	Saves the selected text to a file.
Clear	Clears all text in the Output window tab.
Close	Closes the Output window.

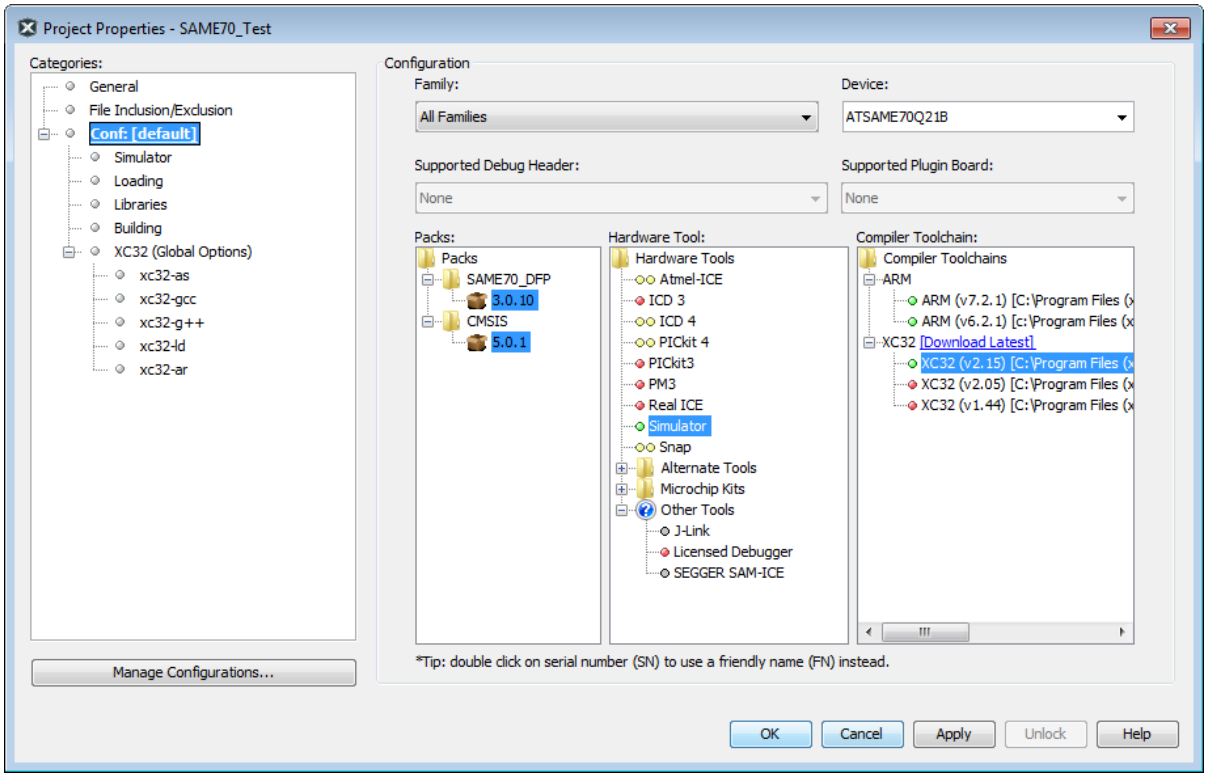
12.14 Project Properties Window

There are several ways to open this window. See 4.3 Open Project Properties.

This window is used to view or change the project device, tools and tool settings. For more information, see:

- 4.4 View or Make Changes to Project Properties
- 4.5 Set Up or Change Debugger/Programmer Tool Options
- 4.6 Set Up or Change Language Tool Options
- 5.5 Loadable Projects, Files and Symbols
- 4.9.5 Add and Set Up Library and Object Files
- 4.10 Set Build Properties

Figure 12-38. Project Properties Window



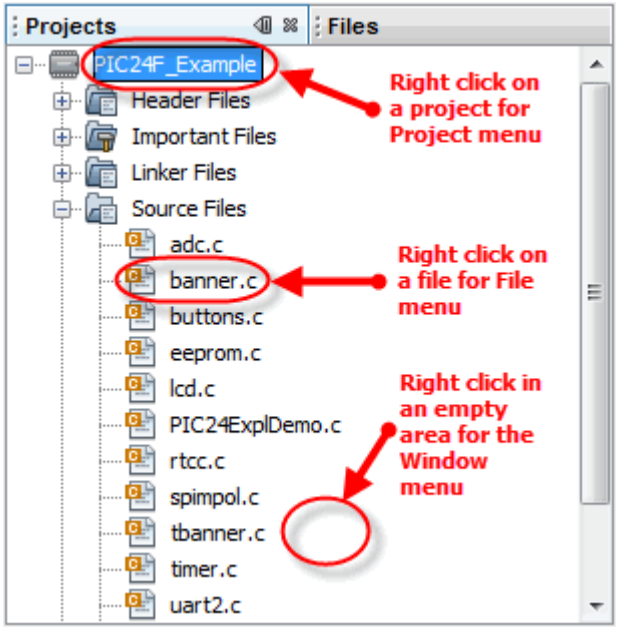
12.15 Projects Window

Open this window by selecting *Window>Projects*.

The Projects window is a NetBeans platform window. However, it has been customized to show the relevant logical (virtual) folders for an MPLAB X IDE project. Also, the context (right click) menus have been customized with items that are specific to MPLAB X IDE.

- Projects Window – Logical Folders
- Projects Window – Project Menu
- Projects Window – File Menu
- Projects Window – Window Menu

Figure 12-39. Projects Window Context Menus



Projects Window – Logical Folders

Additional logical folders have been added that apply to MPLAB X projects. Therefore, all folders displayed can be categorized as one of the following types of files:

- **Header Files** – MPLAB X IDE does not use this category when building. Consider it as a means to document a project's dependency on a header file and a convenient method to access these files. You can double click on a file in the Projects window to open the file in an editor.
- **Important Files** – Any file that does not fit into any of the other categories will end up in this folder.
  You can add project-specific data sheets (PDFs) to this location in the Projects window. Then, you can double click on the PDF to launch the data sheet (this requires that a PDF reader is installed).
- **Linker Files** – You do not need to add a linker script to your project, because the project language tools will find the appropriate generic linker script for your device. So this folder will be empty, unless you have created your own linker script.
  There should be only one file in this folder. If you have more than one linker script, only the first one has any effect – it is the linker script that the tool will use in the link step.
- **Source Files** – These are the only files that the toolchain will accept as input to its file commands.
- **Libraries** – The toolchain should take all of the files in this folder, as well as the object files, and include them in the final link step.
- **Loadables** – Projects and files to combine with or replace your project hex files.

Projects Window – Project Menu

Right click on a project name in the Projects window to pop up the Project menu. Menu items are shown in the table below.

Table 12-67. Project Context Menu Items

Menu Item	Description
New	Adds a new item For MPLAB® X IDE; embedded file types are available.
Add Existing Item	Adds an existing file to the project. The path to the file may be auto (selected by MPLAB X IDE), absolute or relative. Additionally, you may choose to copy the file into the project folder.

.....continued

Menu Item	Description
Add Existing Items from Folders	Adds all files contained in the specified folder(s). You may also specify a pattern to ignore certain folders from inclusion. Click **Default** to see the default pattern. See also *Tools>Options*, **Miscellaneous** button, **Files** tab, "Files ignored by the IDE."
New Logical Folder	Creates a new logical folder. To view actual folders, see the Files window.
Locate Headers	Locates all the header files (.h) called out in the C code project files and presents them in a checklist window so they may be added to the project There are several reasons why you might want to add headers to your project: • The files can be opened from the Projects window under "Header Files" instead of hunting for them on your computer. • The project created by "Save Project As" will not contain header files unless they are added to the project; therefore this project may not build if any user-generated header files are required. • The zipped project created by "Package" will not contain header files unless they are added to the project; therefore the unzipped project may not build if any user-generated header files are required. This feature is able to locate user-generated header files for all Microchip toolchains.
Add Item to Important Files	Adds an existing item to the project. The path to the file may be auto (selected by MPLAB X IDE), absolute or relative. Additionally, you may choose to copy the file into the project folder. This is useful for adding simulator files, data sheet PDFs, and other files to the project for your reference.
Export Hex	Exports the project build as a Hex file. In MPLAB IDE v8, export is an extraction of memory objects. In MPLAB X IDE, export is the hex file result of a build; therefore, it needs to include all necessary auxiliary memory settings for configuration and EEPROM in code.
Build	Builds the project according to the options selected in the Project Properties window. **Note:** When you Debug or Run, your project is built automatically.
Clean and Build	Cleans and then builds the project according to the options selected in the Project Properties window. **Note:** When you Debug or Run, your project is built automatically.
Clean	Cleans the project by deleting the outputs of previous builds.
Package	Packages the current project into a .zip file.
Set Configuration	Opens the Project Properties window so you can set the project configuration.
Run	Executes the project code.
Debug	Executes the project code in a debug environment.
Step In	Steps through Paused code running in the debug environment.
Make and Program Device	Programs the target device and holds in Reset (do not run).
Set as Main Project	Sets this project as the main project. This is useful when you are working with multiple projects.

.....continued

Menu Item	Description
Open Required Projects	Loads all other projects that are required for the selected project to run.
Close	Closes the selected project.
Rename	Renames the project.
Move	Moves the project to another location. Along with the project, it moves the source files that are inside the project directory. Files outside the project directory are not moved. However, the project still maintains reference to the files outside the project directory – this is by design.
Copy	Creates a copy of the project. If the source files are within the project directory, the source files are also copied to the new location. Files outside the project directory are not copied. However, the project still maintains reference to the files outside the project directory – this is by design.
Delete	Deletes the project files. The project file is deleted but not the contents under the project directory. Only on selecting "Also delete sources" are all the source files deleted.
Code Assistance	Selects assistance in creating code: code folding, code completion, etc.
Find	Finds specific text in files in this project.
Share on Team Server	Shares this project on a Team Server.
Versioning	Controls the version of the project by using a version control system.
Local History	To view, or revert to, the local history for the project.
Properties	Sets project properties. For MPLAB X IDE, the Project Properties window is specific to embedded development.

Projects Window – File Menu

Right click on a file name in the Projects window to pop up the File menu. The table below shows the specific menu items.

Table 12-68. File Context Menu Items

Menu Item	Description
Open	Opens this file in a tabbed Editor window.
Cut	Removes the file from the project, but places a copy of it on the clipboard.
Copy	Places a copy of the file on the clipboard.
Paste	To paste the clipboard copy of a file into the project.
Exclude file(s) from current configuration	Select a file or files to exclude from the current project configuration. Once a file has been excluded, right clicking on it again will show "Include file(s) from the current configuration." The same as opening the Project Properties window, selecting "File Inclusion/Exclusion" and including/excluding the file.
Compile File	Compiles only the current file.
Remove from Project	Removes the file from the project. This does not delete the file from the PC. To delete the file from the project and the computer, use the Delete key.

.....continued	
Menu Item	**Description**
Rename	Renames the file.
Save as Template	Saves the current file as a template file.
Local History	To view, or revert to, the local history for the file.
Tools	File tools include: • **Apply Diff Patch**: applies an existing patch created by Diff. • **Diff to**: shows the differences between this file and the one specified here. • **Add to Favorites**: adds this file to the Favorites window.
Properties	Sets file properties differently from the project properties. Select to exclude the file from the build or override the project build options by selecting a different configuration.

Projects Window – Window Menu

Right click in an empty area in the Projects window to pop up the Window menu. The table below shows the specific menu items.

Table 12-69. File Context Menu Items

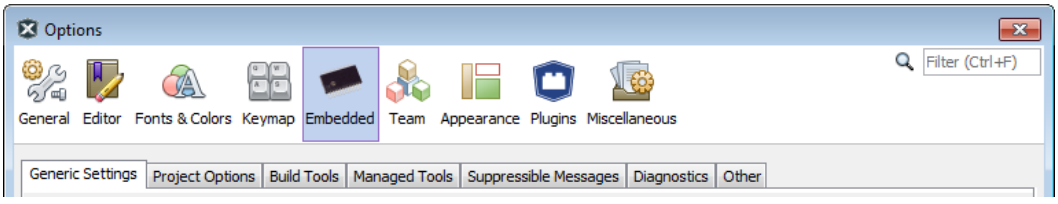
Menu Item	Description
New Project	Launches the New Project wizard.
New File	Launches the New File wizard.
Open Project	Opens an existing project.
Open Recent Project	Opens an existing project for the list of recent projects.
Open Project Group	Opens an existing project group from the list of project groups.
Run Project	Runs the main project.
Set Main Project	Sets the main project from the list of open projects.

12.16 Tools Options Window, Embedded

Select *Tools>Options* (*MPLAB X IDE>Preferences* for macOS) and then click the **Embedded** button to see the tabs and options listed below.

The Options window is a NetBeans platform window. However, it has been customized for MPLAB X IDE projects.

Figure 12-40. Tools Options Embedded Tabs



12.16.1 Generic Settings Tab

Select *Tools>Options* (*MPLAB X IDE>Preferences* for macOS) and then click the **Embedded** button, **Generic Settings** tab, to see the options listed below.

Table 12-70. Generic Settings Tab Items

Item	Description
Open source file and locate line in editor when debugger halts	Jump to a line of code on halt, such as caused by a breakpoint, to examine the code. Disable to simply halt execution.
Clear tool output window on new session (debug, program, upload)	Clear out the contents of the Output window when you begin a Run, Debug or upload.
Halt build on first failure	When building, halt the process on the first failure. The selected project language tool can be set up to determine which errors are produced. Go to the Project Properties window and select the language tool under "Categories." Then look for through the Option Categories to find one that allows you to set up errors, warnings, and/or messages.
Maintain active connection to hardware tool	If selected, keep hardware tool connected always, not just at runtime (MPLAB® IDE v8 behavior). When switching projects with this option selected (e.g., when developing bootloading applications), ensure tool and device are the same to avoid error messages. **Note:** This option applies only to the current project and configuration. If you switch projects or change items in the Project Properties, a reconnect will occur.
Read Device Memory to File: Export only memory used	For the "Read Device Memory to File" icon/function, save only device memory used to a file.
Silent build	Build without generating messages in the Output window.
Enable alternate watch list views during debug session	Display three watch view diamonds in the Watches window Associate a watch view with a watch variable. When you click on a watch view diamond, only the variables associated with that view will be displayed. So, this feature works like a filter.
Disable auto refresh for call stack view during debug sessions	Enable to automatically update contents on Pause or Halt. If the window is closed or not focused, selecting the window and clicking this button will display the updates without having to run and pause/halt again. Disable to manually update contents on Pause or Halt. This button must be clicked to update window contents on a pause/halt.
On mouse over structure and array expressions, evaluate integral members only	Checked shows only integral members when mousing over structures or array expressions in code. Unchecked shows all members when mousing over structures or array expressions in code.
Show unresolved variable names in watch window during debug session	Check to make unresolved variables visible in Watches window.
Allow switch between pack versions	Check to be able to switch between packs displayed in Project Properties.
Enable gathering of compiler symbols	Run the compiler behind-the-scenes, requesting information on compiler built-ins and private macros. This option is disabled by default. Enabling it will increase the time it takes the IDE to gather symbol information. **Note:** If you do not rely on compiler's internal information, you do not need to enable this feature.

.....continued

Item	Description
On mouseover source lines in editor, evaluate break point status	Select what triggers evaluation. **Disable**: No evaluation. **Mouse Only, Alt+Mouse, Shirt+Mouse, Shift+Alt+Mouse**: Evaluate values at a breakpoint based on the one of these keystrokes.
Hold-off period before memory view synchronization: Give priority to debug events (step over, step in, continue)	Specify a delay between when the device halts and when the device data is synced with a memory window. A longer delay can be set so that more single step executions can occur before an update. This is only applicable to general PIC Memory Views like file registers and data memory.
Debug Reset @ (following reset action during paused debug session)	Select action for a debug reset, e.g., *Debug>Reset*: **Main**: Stop at `main()` on Reset. **Reset Vector**: Stop at the Reset vector on Reset. **Note:** This is not related to a hardware reset of the device.
Debug start-up (following debug project action)	Specify code execution for *Debug>Debug Project*: **Run**: Start execution immediately. **Main**: Stop at `main()`. **Reset Vector**: Stop at the Reset vector.
Default Charset	Select the default character set for the project.

12.16.2 Project Options Tab

Select *Tools>Options* (*MPLAB X IDE>Preferences* for macOS) and then click the **Embedded** button, **Project Options** tab, to see the options listed below.

Table 12-71. Project Options Tab Items

Item	Description
Make Options	Enter make options to use when building projects. These options are toolchain dependent. See your language tool documentation.
File Path Mode	Specifies how to store file path information in a project: **Auto**: paths to files inside project folder stored as relative; paths to files outside project folder stored as absolute. **Always Relative**: all paths stored as relative to project folder. **Always Absolute**: all paths stored as absolute (full path).
Save All Modified Files Before Running Make	If selected, saves all unsaved files in the IDE before running make. It is recommended to leave this property selected because modifications to files in the IDE are not recognized by make unless they are first saved to disk.
Reuse Output Tabs from Finished Process	When selected, output messages are displayed in the same tab in the Output window. When deselected, a new tab is opened for each new process.

.....continued	
Item	**Description**
Use parallel make (`make -j 2n`)	If selected, make will execute several processes at a time, where `-j` (or `--jobs`) is the option to run in parallel and 2n is the number of processes, where n is the number of processors available on your computer. If your computer does not support parallel processing, parallel make will be disabled. If you wish, you can specify more processes by using "Make Options." Example: `-j 10`. **Note:** For MPLAB XC16 or MPLAB C30, Procedural Abstraction needs to be turned off to use parallel make (*File>Project Properties*, compiler category, "Optimizations" option category: uncheck "Unlimited procedural abstraction"). **Note:** MPASMX cannot run under parallel make, either as a toolchain or part of an MPLAB C18 project. The parallel make option is therefore ignored in projects using MPASMX toolchain or in projects using the C18 toolchain that contain at least one `.asm` file.
Force makefile regeneration when opening a project	Uncheck to allow MPLAB X IDE to determine whether or not a makefile should be regenerated when a project has opened (e.g., opening a project on a different computer). This is the default. Check to force regeneration of the makefile on project open. Use this if you suspect the makefile is not being regenerated when it needs to be. **Note:** Regeneration can take some time.
Use "clean" target from Makefile	Uncheck to use the standard clean, i.e., delete all build artifacts. This is faster. Check to use make clean, i.e., use the make file when performing clean. This gives more control to what is cleaned.

12.16.3 Build Options Tab

Select *Tools>Options* (*MPLAB X IDE>Preferences* for macOS) and then click the **Embedded** button, **Build Options** tab, to see the options listed below.

Ensure that you have INSTALLED THE LANGUAGE TOOL or it will not show up on the "Toolchain" list. If you know you have installed it but it is not on the list, click Scan for compilers. If it is still not found click Add to add the tool to the list.

The following language tools are included with MPLAB X IDE:

- MPASM toolchain – includes MPASM assembler, MPLINK linker and utilities.

Other tools may be obtained from the Microchip web site (www.microchip.com) or third parties.

Table 12-72. Build Tools Tab Items

Item	Description
Toolchain	Shows a list of language tools installed on your computer. Select the tool for your project and ensure that the paths (that apply) to the right are correct. **Base directory**: Path to the tool's main folder. **C compiler**: Path to the C compiler (if available). **Assembler**: Path to the assembler (if available). **Make command**: Name of the make command generated by MPLAB® X IDE.
Add	Add a new language tool item to the list. Also consider using **Scan for Build Tools**.
Add Custom Compiler	Add a customized compiler. In this case, you must specify Microchip devices supported.

.....continued

Item	Description
Remove	Remove a language tool item from the list. This does not remove the language tool from the computer.
Default	Click on a tool and then click **Default** to make this tool the default compiler/assembler for the selected device.
Scan for Build Tools	Scan the computer for installed compilers/assemblers in various default locations, **not** the whole system. If you install in a different location, add the compiler manually.

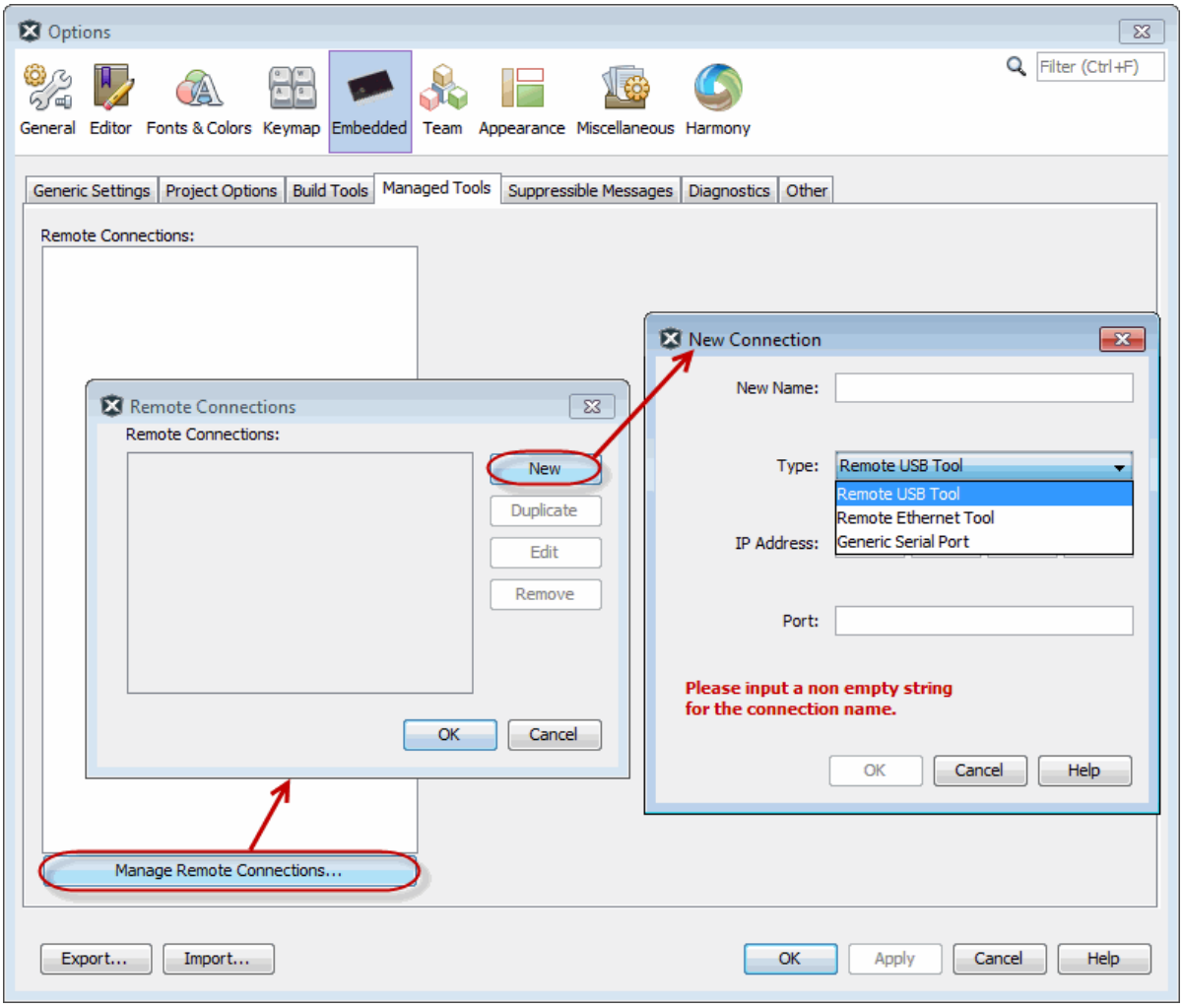
12.16.4 Managed Tools Tab

Select *Tools>Options* (*MPLAB X IDE>Preferences* for macOS) and then click the **Embedded** button, **Managed Tools** tab, to see the options listed below.

Select remote connections to use. Add a new connection for a:

- Remote USB Tool (Plugin required)
- Remote Ethernet Tool
- Generic Serial Port

Figure 12-41. New Managed Tool

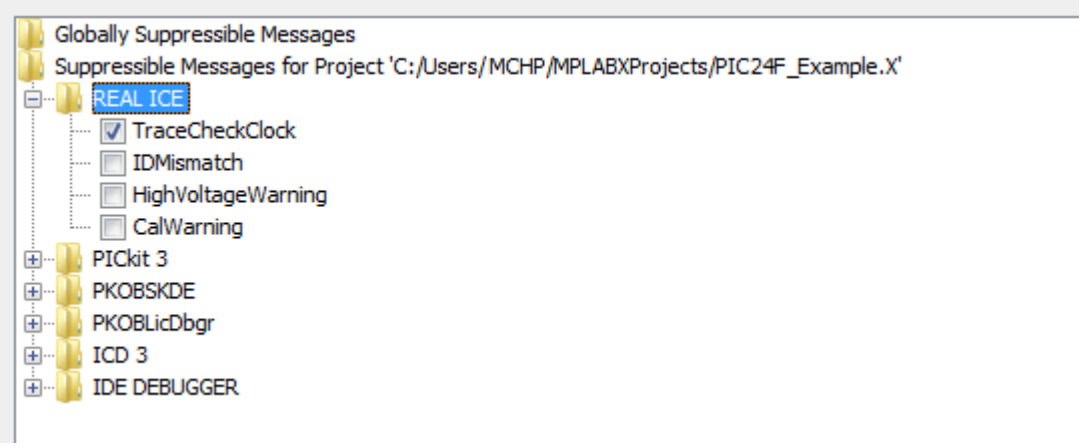


12.16.5 Suppressible Messages Tab

Select *Tools>Options* (*MPLAB X IDE>Preferences* for macOS) and then click he **Embedded** button, **Suppressible Messages** tab, to see the options listed below.

Select error and warming messages to suppress. Double click on available folders to drill down to available items to suppress.

Figure 12-42. Suppressible Messages List



12.16.6 Diagnostics Tab

Select *Tools>Options* (*MPLAB X IDE>Preferences* for macOS) and then click the **Embedded** button, **Diagnostics** tab, to see the options listed below.

Set up the log file and other diagnostic features.

Table 12-73. Diagnostics Tab Items

Item	Description
Logging Level	Set the message logging level: **OFF**: no logging. **SEVERE**: log severe (error) messages only. **WARNING**: log warning messages only. **INFO**: log informational messages only. **CONFIG**: log configuration information only. **FINE**: log some module to module communication. **FINER**: log more module to module communication. **FINEST**: log all module to module communication.
Log File	The path and name of log file.
USB Circular Log	Applies only to MPLAB REAL ICE in-circuit emulator, MPLAB ICD 3 and PICkit 3.
Start New Logging Session/ Pause Logging	Click buttons to begin a new logging session or pause the logging session in progress.
Circular USB log file location	Specify the path for the log file.
Circular USB log file max size in KBs	Specify the size of the log file.

12.16.7 Other Tab

Select *Tools>Options* (*MPLAB X IDE>Preferences* for macOS) and then click the **Embedded** button, **Other** tab, to see the options listed below.

Edit the lists of accepted file extensions for C/C++ and assembler source files and header files. Also, set the default extension for each type (displayed as bold).

Table 12-74. Other Tab Items

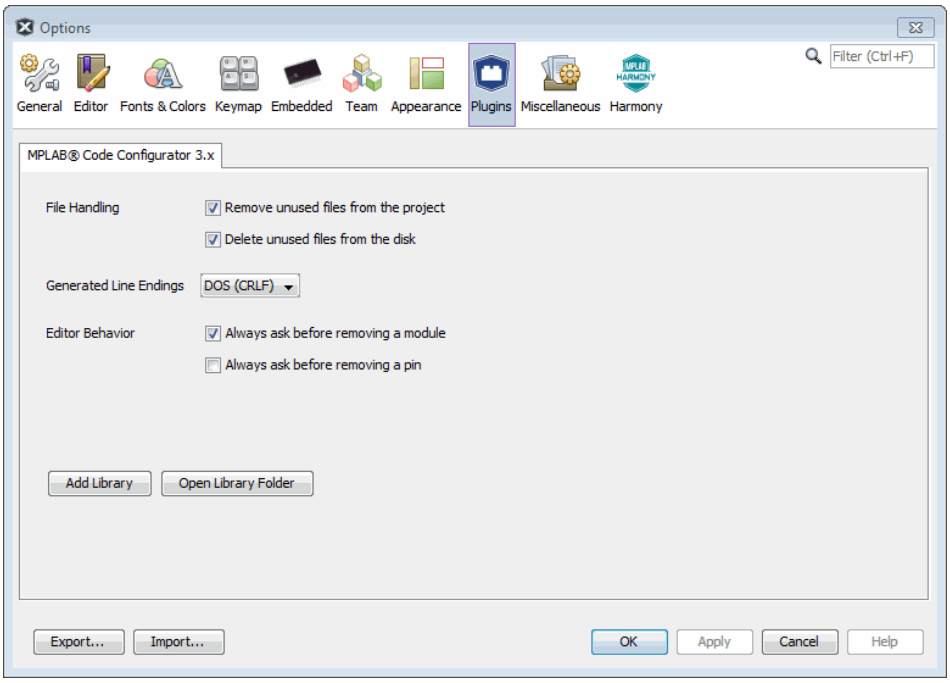
File Extensions	Default Extensions
C/C++ Header	H, SUNWCCh, **h**, hpp, hxx, tcc
C++	C, c++, cc, **cpp**, cxx, mm
C	**c**, i, m
Fortran	Not applicable. MPLAB X IDE does not support Fortran programming.
Assembler	AS, ASM, S, as, asm, **s**
Assembler Include	INC, **inc**

12.17 Tools Options Window, Plugins

Open this window using *Tools>Options* (*MPLAB X IDE>Preferences* for macOS) and then click the **Plugins** button.

The Options window is a NetBeans platform window. However, it has been customized for MPLAB X IDE projects. If plugin(s) have been installed (using *Tools>Plugins*) and they have plugin options, go to this window to see the available plugin options.

Figure 12-43. Plugin Options Example



12.18 Trace Window

Open the Trace window by selecting *Window>Debugging>Trace*. You must be in debug mode to see trace data.

Tracing allows you to record the step-by-step execution of your code and examine this recording in the Trace window. Trace is currently available for the following tools:

- Simulator
- MPLAB REAL ICE in-circuit emulator

Right clicking on a trace column in the window will pop up the context menu (first table). Depending on the tool you are using, you may or may not see all menu items.

Dialogs associated with trace are defined in the second table.

Table 12-75. Trace Window Context Menu

Menu Item	Description
Symbolic Mode	For the "Instruction" column, toggle between displaying literal register addresses (e.g., 0x5) or symbolic register macros (e.g., PORTA).
Go To	Opens a Go To dialog. **Trigger**: move to the trigger line (0). **Top**: move to the top trace line. **Bottom**: move to the bottom trace line. **Trace Line**: specifies and goes to a trace line location.
Go To Source Line	Selects a trace line and then selects this option to go to the corresponding line in source code.
Display Time	For the "Cycle" column (will display if not previously visible): **As Hex Cycle Count**: displays cycle count as hexadecimal. **As Decimal Cycle Count**: displays cycle count as decimal. **In Seconds Elapsed**: displays cycle count in seconds elapsed. **In Engineering Format**: displays cycle count in the appropriate engineering format (powers of 10^3).
Clear Trace File	Clears the data in the trace display.
Find	Finds items in trace display. Opens a Find dialog.
Output To File	Saves the trace data to a file. Opens Define Range dialog, which opens a Save dialog.
Print	Print the data. Opens a Print dialog.
Adjust Table Columns	Adjusts the columns in the trace display to fit the data automatically.
Reload View	Reloads the original data view for the trace display at pause.

Table 12-76. Trace Dialogs

Dialog	Description
Go To	Specifies a trace line to go to.
Find	Finds a line number or other data in the trace display.
Output to File Range	Specifies a range of lines to output to a file. Click **OK** to proceed to the Save dialog to save the data to a text file.
Save	Saves data to a text file.

.....continued	
Dialog	**Description**
Print	Specifies the printer, page setup, and appearance before printing.

12.19 Watches Window

To open the Watches window, select *Window>Debugging>Watches*. You must be in debug mode to watch symbol values change.

Select global symbols and SFRs and watch values change on program halt or during runtime (if supported).

For information on using the Watches window, see 4.16 Watch Symbol Values Change.

- Watches Display
- Watches Menu
- Fractional Integer Properties Dialog

12.19.1 Watches Display

The following display features are discussed in the sections below: icons, columns and actions (buttons).

Icons

Icons are found to the left of the name object in the Name column:

Table 12-77. Name Column Icons

Icon	Description
	Watch object
	Watch object in Program Memory
	Static field of an object
	Non-static field of an object

Columns

You can change the columns displayed in the window by:

- Right clicking on a heading and checking/unchecking a heading to be displayed.
- Clicking on the "Change Visible Columns" icon in the top right of the window  . In the Change Visible Columns dialog, check/uncheck a heading to be displayed.

Actions

Actions are on buttons on the left side of the window:

Table 12-78. Action Button Icons

Button	Action
	Toggle button: • Show verbose (qualified) names of member fields. • Show brief (relative) names of member fields.
	Import watches from list file. Right click for options.
	Export all watches to list file.

.....continued

Button	Action
	Set the default numeric format for the Value field.

12.19.2 Watches Menus

Right click in a blank area of a Watches window (not a row) to open a Watches window menu with basic functions.

Table 12-79. Watches Window Menu – Window

Menu Item	Description
New Watch*	Adds a new symbol to watch.
New Runtime Watch*	**MPLAB REAL ICE In-Circuit Emulator, Simulator Only.** Adds a new runtime watch for the selected symbol.
Export All Watches to List File	Exports information about the symbols to be watched to a file (.xwatch).
Delete All	Removes all the watched symbols from the Watches window.
* You can also right click in a symbol in code to add a New Watch or New Runtime Watch.	

Right click on a row to open a Watches menu with additional functions that depend on whether a symbol is in the row or what debug tool you are using.

Table 12-80. Watches Window Menu – Row

Menu Item	Description
New Watch	Adds a new symbol to watch.
New Runtime Watch	**MPLAB REAL ICE In-Circuit Emulator Only.** Adds a new runtime watch for the selected symbol.
Run Time Update Interval	**MPLAB REAL ICE In-Circuit Emulator Only.** Specifies the rate at which the symbol value will be updated. "No Delay" will update as quickly as your personal computer is able. If you are seeing errors in the runtime data, you can add a delay to decrease the update speed.
Export Selected Watches to List File	Exports information about the symbol(s) to be watched to a file (`.xwatch`). You can select more than one row using Shift click (select several rows in order) or Ctrl click (select several rows individually).
Export Data	Exports watch data to a CSV file or as a literal string.
Delete	Deletes selected symbol row(s) from the watch list. You can also select a row and hit the <Delete> key.
Delete All	Removes all the watched symbols from the Watches window.
Display Value Column as	Displays the symbol value in the selected row in one of the formats listed. See also 12.19.3 Fractional Integer Properties Dialog.
User Defined Size	Specifies the size of the program memory symbol from the list.

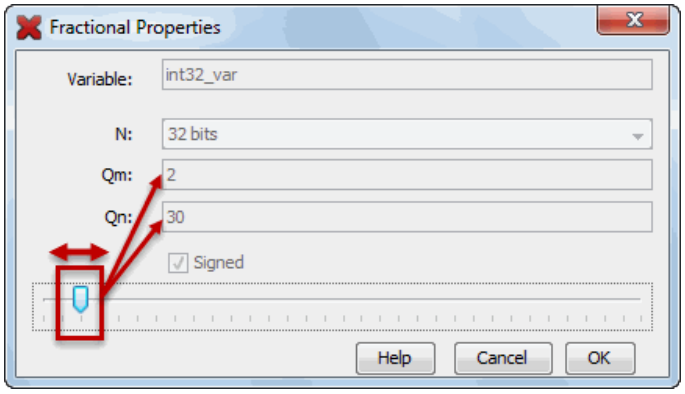
12.19.3 Fractional Integer Properties Dialog

This dialog is intended to improve support for evaluating all fixed-point operation modes supported by dsPIC and PIC32 devices. View this dialog from the pop-up menu item *Display Value Column as>Fractional Integer* to select this feature and *Display Value Column as>Fractional Properties* optionally afterwards.

Table 12-81. Fractional Properties Dialog Items

Dialog Item	Description
Variable	The name of the variable, for reference.
N*	The total number of bits available to represent the integer number.
Qm	The whole part of the fixed data format. 'm' represents the number bits used to specify the whole part.
Qn	The fractional part of the fixed data format. 'n' represents the number of bits used to specify the fractional part.
Signed*	Check to use a bit to represent a signed value.
Slider	Click and drag the slider to select the fixed point position (Qm vs. Qn) to be applied to the calculation of the fixed point fractional value.
* If the variable type is unknown, these items are made editable.	

Figure 12-44. Fractional Properties Dialog

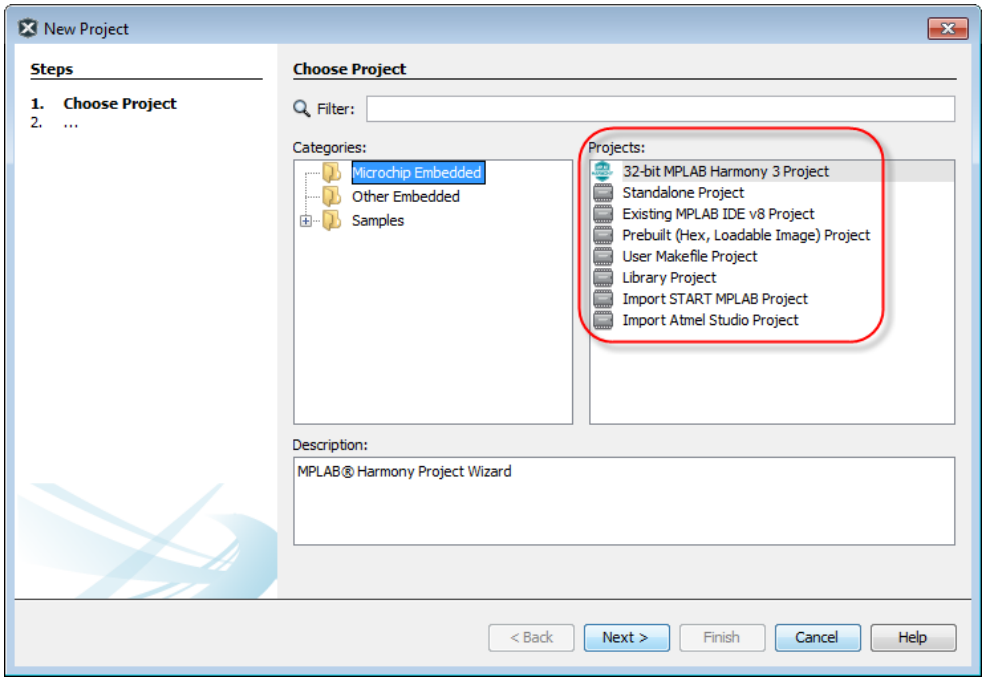


12.20 Wizards

A wizard is a sequential progression of dialog windows that guide you through a process. Two common wizards in MPLAB X IDE are:

- The New Project wizard (see Create a New Project).
- The New File wizard (see Create a New File).

Figure 12-45. New Project Types



13. NetBeans Windows and Dialogs

MPLAB X IDE windows and dialogs are a combination of basic NetBeans windows and dialogs and MPLAB X IDE specific windows and dialogs. Help topics are displayed with a white background for NetBeans items and a yellow background for MPLAB X IDE items.

NetBeans windows and dialogs are discussed here. For information on MPLAB X IDE specific windows and dialogs see 12. MPLAB X IDE Windows and Dialogs.

13.1 NetBeans Specific Windows and Window Menus

NetBeans windows are discussed in NetBeans help or web documentation (see either "About This Help File" or "Preface"). To open help about a window, click on the window tab to select it and then hit the <F1> key. If no help can be found, select *Help>Help Contents* and click the help file's **Search** tab to search for information on that window, or see the table of contents for the NetBeans help topic "Managing IDE Windows."

To open most windows, see the 10.1.11 Window Menu.

Windows may be docked and undocked (right click on the window tab) and have window-specific pop-up, or context, menus with items such as:

- Fill
- Goto
- Edit Cells

Right clicking in a window or on an item in the window will pop up a context menu. Most content menu items are also located on menus on the desktop menu bar (see 10.1 Menus).

Set up window properties by selecting *Tools>Options* (*MPLAB X IDE>Preferences* for macOS), **Miscellaneous** button, **Appearance** tab.

13.2 NetBeans Specific Dialogs

NetBeans dialogs are discussed in NetBeans help. To open help about a dialog, click the **Help** button on the dialog or, if there is no Help button, press the <F1> key. If no help can be found, select *Help>Help Contents* and click the help file's **Search** tab to search for information on that dialog.

To open most dialogs, see 10.1 Menus.

14. Configuration Settings Summary

Examples of how to set configuration bits in code for different language tools and related devices are shown below. For more information on how to set configuration bits, see your language tool documentation. For some language tools, a configurations settings document is available listing all configuration settings for a device. Otherwise, consult your device header file for macros.

Another option is to use the Configuration Memory window to set bits and then click "Generate Source Code to Output." See 4.19 Set Configuration Values in the Configuration Bits Window.

The MPLAB Code Configurator (MCC) for 8- and 16-bit devices and MPLAB Harmony for 32-bit devices are tools that can be used to generate configuration bits, as well as project code. See the Microchip website for more information on these tools.

Current Toolchains: AVR and Arm GCC compilers, MPLAB XC C compilers and MPASM assembler.

Legacy Toolchains: HI-TECH PICC and PICC18 compilers, MPLAB C18 compiler, ASM30 assembler, as well as MPLAB C30 and MPLAB C32 compilers.

14.1 AVR GCC Toolchain

Most configuration bits are contained in device Fuse bytes. To program the fuse bits, the Fuse API is used. To use this API, include the `<avr/io.h>` header file, which provides everything necessary to set the AVR fuses.

To program code protection configuration bits, or lock bits, the Lockbit API is used. To use this API, include the `<avr/io.h>` header file.

An example using the `FUSES` and `LOCKBITS` macros is provided below. For more information on the Fuse and Lockbit APIs, see:

https://www.microchip.com/webdoc/AVRLibcReferenceManual/group__avr__fuse.html

https://www.microchip.com/webdoc/AVRLibcReferenceManual/group__avr__lock.html

Example: ATtiny861 MCU

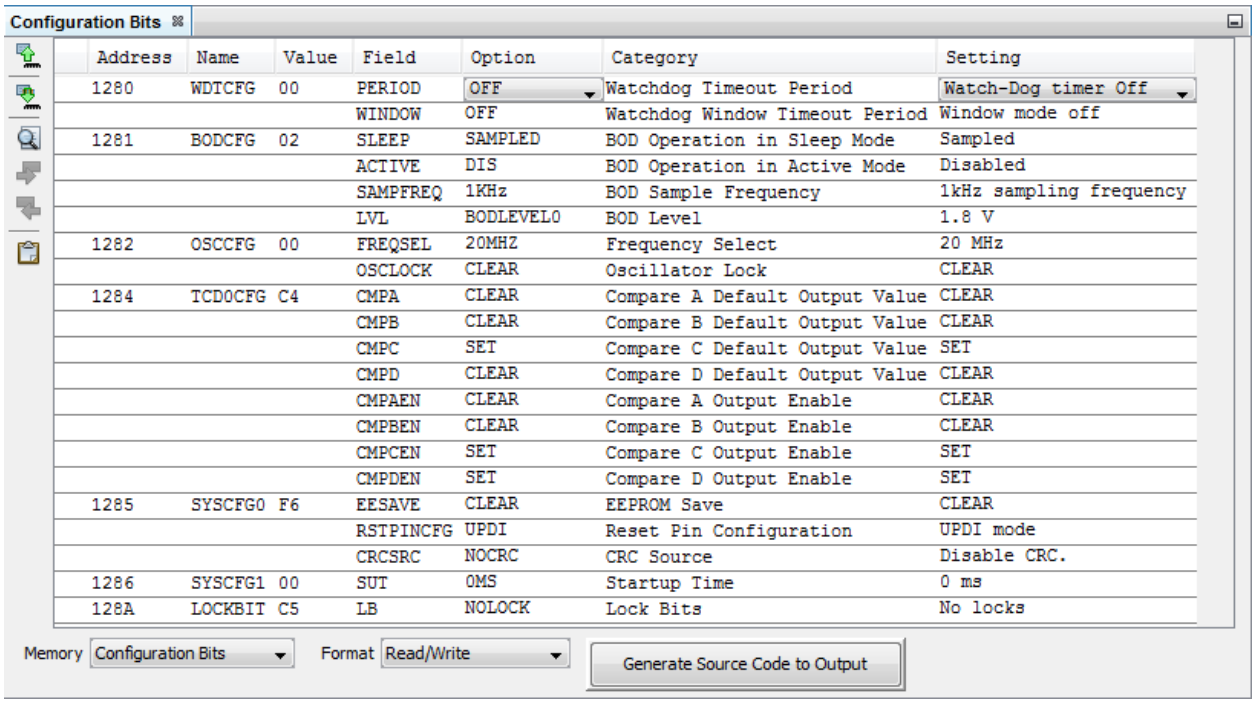
```
// ATtiny817 Configuration Bit Settings

// 'C' source line config statements

#include <avr/io.h>

FUSES = {
    0x00, // WDTCFG{PERIOD=OFF, WINDOW=OFF}
    0x02, // BODCFG{SLEEP=SAMPLED, ACTIVE=DIS, SAMPFREQ=1KHz, LVL=BODLEVEL0}
    0x00, // OSCCFG{OSCLOCK=CLEAR}
    0x00, // Reserved
    0xC4, // TCD0CFG{CMPA=CLEAR, CMPB=CLEAR, CMPC=SET, CMPD=CLEAR, CMPAEN=CLEAR,
CMPBEN=CLEAR, CMPCEN=SET, CMPDEN=SET}
    0xF6, // SYSCFG0{EESAVE=CLEAR, RSTPINCFG=UPDI, CRCSRC=NOCRC}
    0x00, // SYSCFG1{SUT=0MS}
    0x00, // APPEND
    0x00, // BOOTEND
};

LOCKBITS = {
    0xC5, // LOCKBIT{LB=NOLOCK}
};
```

The example above was generated from the Configuration Bits window (*Window>Target Memory Views>Configuration Bits*).

When you first open the Configuration Bits window, the data will be red, meaning you need to read from device memory. Use the "Read Configuration Bits" icon. Make any changes to the data in the window and then write changes back to the device by using the "Program Configuration Bits" icon.

To preserve your settings in code, click either the "Generate Source Code to Output" button, to put the code in the Output window for copy-and-paste into your code, or the "Insert Source Code into Editor" icon, to place the code at the cursor in an editor window.

Figure 14-1. Example: ATtiny817 MCU



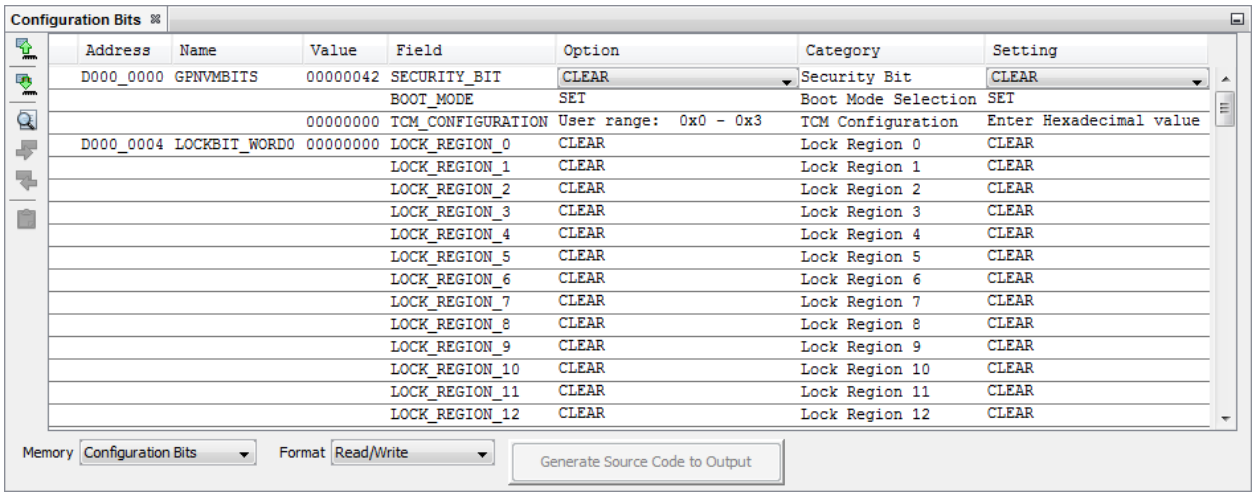
14.2 Arm GCC Toolchain

Most configuration bits are contained in device GPNVM bits. Specifically, the GPNVM0 represents the Security bit.

To program code protection configuration bits, Lock bits are used.

To program configuration bits, use the Configuration Bits window (*Window>Target Memory Views>Configuration Bits*). When you first open the Configuration Bits window, the data will be red, meaning you need to read from device memory. Use the "Read Configuration Bits" icon. Make any changes to the data in the window and then write changes back to the device by using the "Program Configuration Bits" icon.

Example: ATSAME70Q21 MCU



14.3 XC Toolchains

To create code that is as portable as possible, refer to the "Common Compiler Interface (CCI)" chapter, `config` macro, in each of the following documents:

- MPLAB XC8 C Compiler User's Guide (DS50002053) or related help file
- MPLAB XC16 C Compiler User's Guide (DS50002071) or related help file
- MPLAB XC32 C Compiler User's Guide (DS50001686) or related help file

To read, change, and program configuration bits, see 4.19 Set Configuration Values in the Configuration Bits Window.

14.4 MPASM Toolchain

Two types of assembler directives are used to set device configuration bits: `__config` and `config`. DO NOT mix `__config` and `config` in the same code.

14.4.1 __config

The directive `__config` is used for PIC10/12/16 MCUs. It may be used for PIC18 MCUs (excluding PIC18FXXJ devices) but the `config` directive is recommended. The syntax is as follows:

```
__config expr        ;For a single configuration word
```

or

```
__config addr, expr  ;For multiple configuration word
```

where:

`addr`	Address of the Configuration Word. May be literal but usually represented by a macro. **Note:** Macros must be listed in ascending register order.
`expr`	Expression representing the value to which the specified Configuration bits will be set. May be literal but usually represented by a macro or macros ANDed together.

Macros are specified in the device include file (`*.inc`) that is located in the Windows operating system (OS) default directory:

```
C:\Program Files\Microchip\MPLABX\mpasmx
```

Directive case does **not** matter; `__CONFIG` or `__config` is acceptable. Macro case should match what is in the header.

In the "*MPASM Assembler, MPLINK Object Linker, MPLIB Object Librarian User's Guide*" (DS30003014), see "__config - Set Processor Configuration Bits."

Example – PIC10/12/16 MCUs

```
#include p16f877a.inc
;Set oscillator to HS, watchdog time off, low-voltage prog. off
__CONFIG _HS_OSC & _WDT_OFF & _LVP_OFF
```

Example – PIC18 MCUs

```
#include p18f452.inc
;Oscillator switch enabled, RC oscillator with OSC2 as I/O pin.
__CONFIG    _CONFIG1, _OSCS_OFF_1 & _RCIO_OSC_1
;Watch Dog Timer enable, Watch Dog Timer PostScaler count - 1:128
__CONFIG    _CONFIG3, _WDT_ON_3 & _WDTPS_128_3
```

14.4.2 config

The directive `config` is used for PIC18 MCUs (including PIC18FXXJ devices). The syntax is as follows:

```
config setting=value [, setting=value]
```

where:

`setting`	Macro representing a Configuration bit or bits.
`value`	Macro representing the value to which the specified Configuration bit(s) will be set. Multiple settings may be defined on a single line, separated by commas. Settings for a single configuration byte may also be defined on separate lines.

Macros are specified in the device include file (*.inc) that is located in the Windows OS default directory:

```
C:\Program Files\Microchip\MPLABX\mpasmx
```

Directive case does not matter; __CONFIG or __config is acceptable. Macro case should match what is in the header.

In the "*MPASM Assembler, MPLINK Object Linker, MPLIB Object Librarian User's Guide*" (DS30003014), see "config - Set Processor Configuration Bits (PIC18 MCUs)."

Example – PIC18 MCUs

```
#include p18f452.inc
;Oscillator switch enabled, RC oscillator with OSC2 as I/O pin.
CONFIG     OSCS=ON, OSC=LP
;Watch Dog Timer enable, Watch Dog Timer PostScaler count - 1:128
CONFIG     WDT=ON, WDTPS=128
```

14.5 HI-TECH® PICC™ Toolchain

A macro specified in the htc.h header file is used to set device Configuration Words for PIC10/12/16 MCUs: `__CONFIG(x);`

where

x	Expression representing the value to which the specified Configuration bits will be set. May be literal but usually represented by a macro or macros ANDed together.

Macros are specified in the device header file (*.h) that is located in the Windows OS default directory:

```
C:\Program Files\HI-TECH Software\PICC\vx.xx\include
```

where vx.xx is the version number of the compiler.

For devices that have more than one Configuration Word location, each subsequent invocation of __CONFIG() will modify the next Configuration Word in sequence.

Macro case should match what is in the relevant header. For htc.c, __CONFIG() is correct but __config() is not.

In "*HI-TECH C for PIC10/12/16 User's Guide*" (DS51865), see "Configuration Fuses."

PICC Example

```
#include <htc.h>
__CONFIG(WDTDIS & XT & UNPROTECT); // Program config. word 1
__CONFIG(FCMEN);                  // Program config. word 2
```

14.6 HI-TECH® PICC-18™ Toolchain

A macro specified in the htc.h header file is used to set device Configuration Words for PIC18 MCUs:

```
__CONFIG(n,x);
```

where

n	Configuration register number.
x	Expression representing the value to which the specified Configuration bits will be set. May be literal but usually represented by a macro or macros ANDed together.

Macros are specified in the device header file (*.h) that is located in the Windows OS default directory:

```
C:\Program Files\HI-TECH Software\PICC\vx.xx\include
```

where vx.xx is the version number of the compiler.

Macro case should match what is in the relevant header. For htc.c, __CONFIG() is correct but __config() is not.

In "*HI-TECH C Tools for the PIC18 MCU Family*," see "Configuration Fuses."

PICC-18 Example

```
#include <htc.h>
//Oscillator switch enabled, RC oscillator with OSC2 as I/O pin.
__CONFIG(1, LP);
//Watch Dog Timer enable, Watch Dog Timer PostScaler count - 1:128
__CONFIG(2, WDTEN & WDTPS128);
```

14.7 C18 Toolchain

The #pragma config directive specifies the device-specific configuration settings (i.e., Configuration bits) to be used by the application:

```
#pragma config setting-list
```

where

`setting-list`	A list of one or more `setting-name = value-name` macro pairs, separated by commas.

Macros are specified in the device header file (`*.h`) that is located in the Windows default directory:

```
C:\program files\microchip\mplabc18\vx.xx\.h
```

Pragma case does **not** matter; either `#PRAGMA CONFIG` or `#pragma config` is acceptable. Macro case should match what is in the header.

In the "*MPLAB C18 C Compiler User's Guide*" (DS50001288), see "Pragms," "#pragma config."

Example

```
#include <p18cxxx.h>
/*Oscillator switch enabled, RC oscillator with OSC2 as I/O pin.*/
#pragma config OSCS = ON, OSC = LP
/*Watch Dog Timer enable, Watch Dog Timer PostScaler count - 1:128*/
#pragma config WDT = ON, WDTPS = 128
```

14.8 ASM30 Toolchain

A macro specified in the device include file is used to set Configuration bits:

```
config __reg, value
```

where

`__reg`	Configuration register name macro.
`value`	Expression representing the value to which the specified Configuration bits will be set. May be literal but usually represented by a macro or macros ANDed together.

Macros are specified in the device include file (`*.inc`) that is located in the Windows OS default directory:

```
C:\Program Files\Microchip\MPLAB ASM30 Suite\Support\device\inc
```

where `device` is the abbreviation of the selected 16-bit device, such as PIC24H or dsPIC33F.

Macro case should match what is in the relevant header. For example, `config` is correct but `CONFIG` is not.

In the "*MPLAB® Assembler, Linker and Utilities for PIC24 MCUs and dsPIC® DSCs User's Guide*" (DS50001317), see "Output Sections in Configuration Memory."

Example

```
.include "p30fxxxx.inc"
;Clock switching off, Fail-safe clock monitoring off,
; Use External Clock
config __FOSC, CSW_FSCM_OFF & XT_PLL16
;Turn off Watchdog Timer
config __FWDT, WDT_OFF
```

14.9 C30 Toolchain

Two types of compiler macros are used to set device Configuration bits: one type for PIC24F MCUs and one type for dsPIC30F and dsPIC33F/PIC24H devices.

14.9.1 PIC24F Configuration Settings

Macros are provided in device header files to set Configuration bits:

```
_configin(value);
```

where

n	Configuration register number.
value	Expression representing the value to which the specified Configuration bits will be set. May be literal but usually represented by a macro or macros ANDed together.

Macros are specified in the device header file (*.h) that is located in the Windows OS default directory:

```
C:\Program Files\Microchip\MPLAB C30\support\PIC24F\h
```

Macro case should match what is in the relevant header. For example, _CONFIG1 is correct but _config1 is not.

Example – PIC24F MCUs

```
#include "p24Fxxxx.h"
//JTAG off, Code Protect off, Write Protect off, COE mode off, WDT off
_CONFIG1( JTAGEN_OFF & GCP_OFF & GWRP_OFF & COE_OFF & FWDTEN_OFF )
//Clock switching/monitor off, Oscillator (RC15) on,
// Oscillator in HS mode, Use primary oscillator (no PLL)
_CONFIG2( FCKSM_CSDCMD & OSCIOFNC_ON & POSCMOD_HS & FNOSC_PRI )
```

14.9.2 dsPIC30F/33F/PIC24H Configuration Settings

Macros are provided in device header files to set Configuration bits:

```
_reg(value);
```

where

_reg	Configuration register name macro.
value	Expression representing the value to which the specified Configuration bits will be set. May be literal but usually represented by a macro or macros ANDed together.

Macros are specified in the device header file (*.h) that is located in the Windows OS default directory:

```
C:\Program Files\Microchip\MPLAB C30\support\device\h
```

where device is the abbreviation of the selected 16-bit device, i.e., PIC24H, dsPIC30F, or dsPIC33F.

Macro case should match what is in the relevant header. For example, _FOSC is correct but, _fosc is not.

In the "*MPLAB® C Compiler for PIC24 MCUs and dsPIC® DSCs User's Guide*" (DS50001284), see "Configuration Bits Setup Macros."

Example – dsPIC30F DSCs

```
#include "p30fxxxx.h"
//Clock switching and failsafe clock monitoring off,
// Oscillator in HS mode
_FOSC(CSW_FSCM_OFF & HS);
//Watchdog timer off
_FWDT(WDT_OFF);
//Brown-out off, Master clear on
_FBORPOR(PBOR_OFF & MCLR_EN);
```

Example – dsPIC33F/PIC24H Devices

```
#include "p33fxxxx.h"
// Use primary oscillator (no PLL)
_FOSCSEL(FNOSC_PRI);
//Oscillator in HS mode
_FOSC(POSCMD_HS);
//Watchdog timer off
_FWDT(FWDTEN_OFF);
//JTAG off
_FICD(JTAGEN_OFF);
```

14.10 C32 Toolchain

The `#pragma config` directive specifies the device-specific configuration settings (i.e., Configuration bits) to be used by the application:

```
# pragma config setting-list
```

where

`setting-list:`	A list of one or more `setting-name = value-name` macro pairs, separated by commas.

Macros are specified in the device header file (`*.h`) that is located in the Windows OS default directory:

```
C:\Program Files\Microchip\MPLAB C32\pic32mx\include\proc
```

Pragma case does **not** matter; either `#PRAGMA CONFIG` or `#pragma config` is acceptable. Macro case should match what is in the header.

In the "*MPLAB® C Compiler for PIC32 MCUs User's Guide*" (DS50001686), see "Configuration Bit Access."

Example

```
#include "p32xxxx.h"
//Enables the Watchdog Timer,
// Sets the Watchdog Postscaler to 1:128
#pragma config FWDTEN = ON, WDTPS = PS128
//Selects the HS Oscillator for the Primary Oscillator
#pragma config POSCMOD = HS
```

15. Glossary

Absolute Section
A GCC compiler section with a fixed (absolute) address that cannot be changed by the linker.

Absolute Variable/Function
A variable or function placed at an absolute address using the OCG compiler's @ address syntax.

Access Memory
PIC18 Only – Special registers on PIC18 devices that allow access regardless of the setting of the Bank Select Register (BSR).

Access Entry Points
Access entry points provide a way to transfer control across segments to a function which may not be defined at link time. They support the separate linking of boot and secure application segments.

Address
A value that identifies a location in memory.

Alphabetic Character
Alphabetic characters are those characters that are letters of the Roman alphabet (a, b, …, z, A, B, …, Z).

Alphanumeric
Alphanumeric characters are comprised of alphabetic characters and decimal digits (0,1, …, 9).

ANDed Breakpoints
Set up an ANDed condition for breaking, i.e., breakpoint 1 AND breakpoint 2 must occur at the same time before a program halt. This can only be accomplished if a data breakpoint and a program memory breakpoint occur at the same time.

Anonymous Structure
16-bit C Compiler – An unnamed structure.

PIC18 C Compiler – An unnamed structure that is a member of a C union. The members of an anonymous structure may be accessed as if they were members of the enclosing union. For example, in the following code, hi and lo are members of an anonymous structure inside the union caster.

```
union castaway
int intval;
struct {
char lo; //accessible as caster.lo
char hi; //accessible as caster.hi
};
} caster;
```

ANSI
The American National Standards Institute is an organization responsible for formulating and approving standards in the United States.

Application
A set of software and hardware that may be controlled by a PIC® microcontroller.

Archive/Archiver
An archive/library is a collection of relocatable object modules. It is created by assembling multiple source files to object files, and then using the archiver/librarian to combine the object files into one archive/library file. An archive/library can be linked with object modules and other archives/libraries to create executable code.

ASCII
The American Standard Code for Information Interchange is a character set encoding that uses 7 binary digits to represent each character. It includes upper and lower case letters, digits, symbols and control characters.

Assembly/Assembler
Assembly is a programming language that describes binary machine code in a symbolic form. An assembler is a language tool that translates assembly language source code into machine code.

Assigned Section
A GCC compiler section which has been assigned to a target memory block in the linker command file.

Asynchronously
Multiple events that do not occur at the same time. This is generally used to refer to interrupts that may occur at any time during processor execution.

Asynchronous Stimulus
Data generated to simulate external inputs to a simulator device.

Attribute
GCC Characteristics of variables or functions in a C language program, which are used to describe machine-specific properties.

Attribute, Section
GCC Characteristics of sections, such as "executable", "read-only", or "data" that can be specified as flags in the assembler `.section` directive.

Binary
The base two numbering system that uses the digits 0-1. The rightmost digit counts ones, the next counts multiples of 2, then $2^2 = 4$, etc.

Bookmarks
Use bookmarks to easily locate specific lines in a file.

Select Toggle Bookmarks on the Editor toolbar to add/remove bookmarks. Click other icons on this toolbar to move to the next or previous bookmark.

C/C++
C is a general-purpose programming language which features economy of expression, modern control flow and data structures, as well as a rich set of operators. C++ is the object-oriented version of C.

Calibration Memory
A special function register or registers used to hold values for calibration of a PIC microcontroller on-board RC oscillator or other device peripherals.

Central Processing Unit
The part of a device that is responsible for fetching the correct instruction for execution, decoding that instruction, and then executing that instruction. When necessary, it works in conjunction with the arithmetic logic unit (ALU) to complete the execution of the instruction. It controls the program memory address bus, the data memory address bus, and accesses to the stack.

Clean
Clean removes all intermediary project files, such as object, hex and debug files, for the active project. These files are recreated from other files when a project is built.

COFF
Common Object File Format. An object file of this format contains machine code, debugging and other information.

Command Line Interface
A means of communication between a program and its user based solely on textual input and output.

Compiled Stack
A region of memory managed by the compiler in which variables are statically allocated space. It replaces a software or hardware stack when such mechanisms cannot be efficiently implemented on the target device.

Compiler
A program that translates a source file written in a high-level language into machine code.

Conditional Assembly
Assembly language code that is included or omitted based on the assembly-time value of a specified expression.

Conditional Compilation
The act of compiling a program fragment only if a certain constant expression, specified by a preprocessor directive, is true.

Configuration Bits
Special-purpose bits programmed to set PIC MCU and dsPIC DSC modes of operation. A Configuration bit may or may not be preprogrammed.

Control Directives
Directives in assembly language code that cause code to be included or omitted based on the assembly-time value of a specified expression.

CPU
See *Central Processing Unit*.

Cross Reference File
A file that references a table of symbols and a list of files that references the symbol. If the symbol is defined, the first file listed is the location of the definition. The remaining files contain references to the symbol.

Data Directives
Data directives are those that control the assembler's allocation of program or data memory and provide a way to refer to data items symbolically; that is, by meaningful names.

Data Memory
On Microchip MCU and DSC devices, data memory (RAM) is comprised of General Purpose Registers (GPRs) and Special Function Registers (SFRs). Some devices also have EEPROM data memory.

Data Monitor and Control Interface (DMCI)
The Data Monitor and Control Interface, or DMCI, is a tool in MPLAB X IDE. The interface provides dynamic input control of application variables in projects. Application-generated data can be viewed graphically using any of 4 dynamically-assignable graph windows.

Debug/Debugger
See *ICE/ICD*.

Debugging Information
Compiler and assembler options that, when selected, provide varying degrees of information used to debug application code. See compiler or assembler documentation for details on selecting debug options.

Deprecated Features
Features that are still supported for legacy reasons, but will eventually be phased out and no longer used.

Device Programmer
A tool used to program electrically programmable semiconductor devices such as microcontrollers.

Digital Signal Controller
A digital signal controller (DSC) is a microcontroller device with digital signal processing capability, i.e., Microchip dsPIC DSC devices.

Digital Signal Processing\Digital Signal Processor
Digital signal processing (DSP) is the computer manipulation of digital signals, commonly analog signals (sound or image) which have been converted to digital form (sampled). A digital signal processor is a microprocessor that is designed for use in digital signal processing.

Directives
Statements in source code that provide control of the language tool's operation.

Download
Download is the process of sending data from a host to another device, such as an emulator, programmer or target board.

DWARF
Debug With Arbitrary Record Format. DWARF is a debug information format for ELF files.

EEPROM
Electrically Erasable Programmable Read Only Memory. A special type of PROM that can be erased electrically. Data is written or erased one byte at a time. EEPROM retains its contents even when power is turned off.

ELF
Executable and Linking Format. An object file of this format contains machine code. Debugging and other information is specified in with DWARF. ELF/DWARF provide better debugging of optimized code than COFF.

Emulation/Emulator
See *ICE/ICD*.

Endianness
The ordering of bytes in a multi-byte object.

Environment
MPLAB PM3 – A folder containing files on how to program a device. This folder can be transferred to a SD/MMC card.

Epilogue
A portion of compiler-generated code that is responsible for deallocating stack space, restoring registers and performing any other machine-specific requirement specified in the runtime model. This code executes after any user code for a given function, immediately prior to the function return.

EPROM
Erasable Programmable Read Only Memory. A programmable read-only memory that can be erased usually by exposure to ultraviolet radiation.

Error/Error File
An error reports a problem that makes it impossible to continue processing your program. When possible, an error identifies the source file name and line number where the problem is apparent. An error file contains error messages and diagnostics generated by a language tool.

Event
A description of a bus cycle which may include address, data, pass count, external input, cycle type (fetch, R/W) and time stamp. Events are used to describe triggers, breakpoints and interrupts.

Executable Code
Software that is ready to be loaded for execution.

Export
Send data out of the MPLAB IDE/MPLAB X IDE in a standardized format.

Expressions
Combinations of constants and/or symbols separated by arithmetic or logical operators.

Extended Microcontroller Mode
In extended microcontroller mode, on-chip program memory as well as external memory is available. Execution automatically switches to external if the program memory address is greater than the internal memory space of the PIC18 device.

Extended Mode (PIC18 MCUs)
In Extended mode, the compiler will utilize the extended instructions (i.e., `ADDFSR`, `ADDULNK`, `CALLW`, `MOVSF`, `MOVSS`, `PUSHL`, `SUBFSR`, and `SUBULNK`) and the indexed with literal offset addressing.

External Label
A label that has external linkage.

External Linkage
A function or variable has external linkage if it can be referenced from outside the module in which it is defined.

External Symbol
A symbol for an identifier which has external linkage. This may be a reference or a definition.

External Symbol Resolution
A process performed by the linker in which external symbol definitions from all input modules are collected in an attempt to resolve all external symbol references. Any external symbol references which do not have a corresponding definition cause a linker error to be reported.

External Input Line
An external input signal logic probe line (TRIGIN) for setting an event based upon external signals.

External RAM
Off-chip Read/Write memory.

Fatal Error
An error that halts compilation immediately. No further messages will be produced.

File Registers
On-chip data memory, including General Purpose Registers (GPRs) and Special Function Registers (SFRs).

Filter
Determine by selection what data is included/excluded in a trace display or data file.

Fixup
The process of replacing object file symbolic references with absolute addresses after relocation by the linker.

Flash
A type of EEPROM where data is written or erased in blocks instead of bytes.

FNOP
Forced No Operation. A forced NOP cycle is the second cycle of a two-cycle instruction. Since the PIC microcontroller architecture is pipelined, it prefetches the next instruction in the physical address space while it is executing the current instruction. However, if the current instruction changes the program counter, this prefetched instruction is explicitly ignored, causing a forced NOP cycle.

Frame Pointer
A pointer that references the location on the stack that separates the stack-based arguments from the stack-based local variables. Provides a convenient base from which to access local variables and other values for the current function.

Free-Standing
An implementation that accepts any strictly conforming program that does not use complex types and in which the use of the features specified in the library clause (ANSI '89 standard clause 7) is confined to the contents of the standard headers `<float.h>`, `<iso646.h>`, `<limits.h>`, `<stdarg.h>`, `<stdbool.h>`, `<stddef.h>`, and `<stdint.h>`.

GPR
General Purpose Register. The portion of device data memory (RAM) available for general use.

Halt
A stop of program execution. Executing Halt is the same as stopping at a breakpoint.

Heap
An area of memory used for dynamic memory allocation where blocks of memory are allocated and freed in an arbitrary order determined at run-time.

Hex Code/Hex File
Hex code is executable instructions stored in a hexadecimal format code. Hex code is contained in a hex file.

Hexadecimal
The base 16 numbering system that uses the digits 0-9 plus the letters A-F (or a-f). The digits A-F represent hexadecimal digits with values of (decimal) 10 to 15. The rightmost digit counts ones, the next counts multiples of 16, then 16^2 = 256, etc.

High Level Language
A language for writing programs that is further removed from the processor than assembly.

ICE/ICD
In-Circuit Emulator/In-Circuit Debugger: A hardware tool that debugs and programs a target device. An emulator has more features than an debugger, such as trace.

In-Circuit Emulation/In-Circuit Debug: The act of emulating or debugging with an in-circuit emulator or debugger.

-ICE/-ICD: A device (MCU or DSC) with on-board in-circuit emulation or debug circuitry. This device is always mounted on a header board and used to debug with an in-circuit emulator or debugger.

ICSP
In-Circuit Serial Programming. A method of programming Microchip embedded devices using serial communication and a minimum number of device pins.

IDE
Integrated Development Environment, as in MPLAB IDE/MPLAB X IDE.

Identifier
A function or variable name.

IEEE
Institute of Electrical and Electronics Engineers.

Import
Bring data into the MPLAB IDE/MPLAB X IDE from an outside source, such as from a hex file.

Initialized Data
Data which is defined with an initial value. In C,

```
int myVar=5;
```

defines a variable, which will reside in an initialized data section.

Instruction Set
The collection of machine language instructions that a particular processor understands.

Instructions
A sequence of bits that tells a central processing unit to perform a particular operation and can contain data to be used in the operation.

Internal Linkage
A function or variable has internal linkage if it can not be accessed from outside the module in which it is defined.

International Organization for Standardization
An organization that sets standards in many businesses and technologies, including computing and communications. Also known as ISO.

Interrupt
A signal to the CPU that suspends the execution of a running application and transfers control to an Interrupt Service Routine (ISR) so that the event may be processed. Upon completion of the ISR, normal execution of the application resumes.

Interrupt Handler
A routine that processes special code when an interrupt occurs.

Interrupt Service Request (IRQ)
An event which causes the processor to temporarily suspend normal instruction execution and to start executing an interrupt handler routine. Some processors have several interrupt request events allowing different priority interrupts.

Interrupt Service Routine (ISR)
Language tools: A function that handles an interrupt.

MPLAB IDE/MPLAB X IDE: User-generated code that is entered when an interrupt occurs. The location of the code in program memory will usually depend on the type of interrupt that has occurred.

Interrupt Vector
Address of an interrupt service routine or interrupt handler.

L-value
An expression that refers to an object that can be examined and/or modified. An l-value expression is used on the left-hand side of an assignment.

Latency
The time between an event and its response.

Library/Librarian
See *Archive/Archiver*.

Linker
A language tool that combines object files and libraries to create executable code, resolving references from one module to another.

Linker Script Files
Linker script files are the command files of a linker. They define linker options and describe available memory on the target platform.

Listing Directives
Listing directives are those directives that control the assembler listing file format. They allow the specification of titles, pagination and other listing control.

Listing File
A listing file is an ASCII text file that shows the machine code generated for each C source statement, assembly instruction, assembler directive, or macro encountered in a source file.

Little Endian
A data ordering scheme for multi-byte data, whereby the least significant byte is stored at the lower addresses.

Local Label
A local label is one that is defined inside a macro with the LOCAL directive. These labels are particular to a given instance of a macro's instantiation. In other words, the symbols and labels that are declared as local are no longer accessible after the ENDM macro is encountered.

Logic Probes
Up to 14 logic probes can be connected to some Microchip emulators. The logic probes provide external trace inputs, trigger output signal, +5V, and a common ground.

Loop-Back Test Board
Used to test the functionality of the MPLAB REAL ICE in-circuit emulator.

LVDS
Low Voltage Differential Signaling. A low noise, low-power, low amplitude method for high-speed (gigabits per second) data transmission over copper wire.

With standard I/O signaling, data storage is contingent upon the actual voltage level. Voltage level can be affected by wire length (longer wires increase resistance, which lowers voltage). But with LVDS, data storage is distinguished only by positive and negative voltage values, not the voltage level. Therefore, data can travel over greater lengths of wire while maintaining a clear and consistent data stream.

Source: http://www.webopedia.com/TERM/L/LVDS.html

Machine Code
The representation of a computer program that is actually read and interpreted by the processor. A program in binary machine code consists of a sequence of machine instructions (possibly interspersed with data). The collection of all possible instructions for a particular processor is known as its "instruction set".

Machine Language
A set of instructions for a specific central processing unit, designed to be usable by a processor without being translated.

Macro
Macro instruction. An instruction that represents a sequence of instructions in abbreviated form.

Macro Directives
Directives that control the execution and data allocation within macro body definitions.

Makefile
Export to a file the instructions to Make the project. Use this file to Make your project outside of MPLAB IDE/MPLAB X IDE, i.e., with a make.

Make Project
A command that rebuilds an application, recompiling only those source files that have changed since the last complete compilation.

MCU
Microcontroller Unit. An abbreviation for microcontroller. Also uC.

Memory Model
For C compilers, a representation of the memory available to the application. For the PIC18 C compiler, a description that specifies the size of pointers that point to program memory.

Message
Text displayed to alert you to potential problems in language tool operation. A message will not stop operation.

Microcontroller
A highly integrated chip that contains a CPU, RAM, program memory, I/O ports and timers.

Microcontroller Mode
One of the possible program memory configurations of PIC18 microcontrollers. In microcontroller mode, only internal execution is allowed. Thus, only the on-chip program memory is available in microcontroller mode.

Microprocessor Mode
One of the possible program memory configurations of PIC18 microcontrollers. In microprocessor mode, the on-chip program memory is not used. The entire program memory is mapped externally.

Mnemonics
Text instructions that can be translated directly into machine code. Also referred to as opcodes.

Module
The preprocessed output of a source file after preprocessor directives have been executed. Also known as a translation unit.

MPASM™ Assembler
Microchip Technology's relocatable macro assembler for PIC microcontroller devices, KeeLoq® devices and Microchip memory devices.

MPLAB Language Tool for Device
Microchip's C compilers, assemblers and linkers for specified devices. Select the type of language tool based on the device you will be using for your application, e.g., if you will be creating C code on a PIC18 MCU, select the MPLAB C Compiler for PIC18 MCUs.

MPLAB ICD
Microchip in-circuit debugger that works with MPLAB IDE/MPLAB X IDE. See ICE/ICD.

MPLAB IDE/MPLAB X IDE
Microchip's Integrated Development Environment. MPLAB IDE/MPLAB X IDE comes with an editor, project manager and simulator.

MPLAB PM3
A device programmer from Microchip. Programs PIC18 microcontrollers and dsPIC digital signal controllers. Can be used with MPLAB IDE/MPLAB X IDE or stand-alone. Replaces PRO MATE II.

MPLAB REAL ICE™ In-Circuit Emulator
Microchip's next-generation in-circuit emulator that works with MPLAB IDE/MPLAB X IDE. See ICE/ICD.

MPLAB SIM
Microchip's simulator that works with MPLAB IDE/MPLAB X IDE in support of PIC MCU and dsPIC DSC devices.

MPLAB Starter Kit for Device
Microchip's starter kits contains everything needed to begin exploring the specified device. View a working application and then debug and program you own changes.

MPLIB™ Object Librarian
Microchip's librarian that can work with MPLAB IDE/MPLAB X IDE. MPLIB librarian is an object librarian for use with COFF object modules created using either MPASM assembler (mpasm or mpasmwin v2.0) or MPLAB C18 C Compiler.

MPLINK™ Object Linker
MPLINK linker is an object linker for the Microchip MPASM assembler and the Microchip C18 C compiler. MPLINK linker also may be used with the Microchip MPLIB librarian. MPLINK linker is designed to be used with MPLAB IDE/MPLAB X IDE, although it is not required.

MRU
Most Recently Used. Refers to files and windows available to be selected from MPLAB IDE/MPLAB X IDE main pull down menus.

Native Data Size
For Native trace, the size of the variable used in a Watches window must be of the same size as the selected device's data memory: bytes for PIC18 devices and words for 16-bit devices.

Nesting Depth
The maximum level to which macros can include other macros.

Node
MPLAB IDE/MPLAB X IDE project component.

Non-Extended Mode (PIC18 MCUs)
In Non-Extended mode, the compiler will not utilize the extended instructions nor the indexed with literal offset addressing.

Non Real Time
Refers to the processor at a breakpoint or executing single-step instructions or MPLAB IDE/MPLAB X IDE being run in simulator mode.

Non-Volatile Storage
A storage device whose contents are preserved when its power is off.

NOP
No Operation. An instruction that has no effect when executed except to advance the program counter.

Object Code/Object File
Object code is the machine code generated by an assembler or compiler. An object file is a file containing machine code and possibly debug information. It may be immediately executable or it may be relocatable, requiring linking with other object files, e.g., libraries, to produce a complete executable program.

Object File Directives
Directives that are used only when creating an object file.

Octal
The base 8 number system that only uses the digits 0-7. The rightmost digit counts ones, the next digit counts multiples of 8, then $8^2 = 64$, etc.

Off-Chip Memory
Off-chip memory refers to the memory selection option for the PIC18 device where memory may reside on the target board, or where all program memory may be supplied by the emulator. The Memory tab accessed from Options>Development Mode provides the Off-Chip Memory selection dialog box.

Opcodes
Operational Codes. See *Mnemonics*.

Operators
Symbols, like the plus sign '+' and the minus sign '-', that are used when forming well-defined expressions. Each operator has an assigned precedence that is used to determine order of evaluation.

OTP
One Time Programmable. EPROM devices that are not in windowed packages. Since EPROM needs ultraviolet light to erase its memory, only windowed devices are erasable.

Pass Counter
A counter that decrements each time an event (such as the execution of an instruction at a particular address) occurs. When the pass count value reaches zero, the event is satisfied. You can assign the Pass Counter to break and trace logic, and to any sequential event in the complex trigger dialog.

PC
Personal Computer or Program Counter.

PC Host
Any PC running a supported Windows operating system.

Persistent Data
Data that is never cleared or initialized. Its intended use is so that an application can preserve data across a device Reset.

Phantom Byte
An unimplemented byte in the dsPIC architecture that is used when treating the 24-bit instruction word as if it were a 32-bit instruction word. Phantom bytes appear in dsPIC hex files.

PIC MCUs
PIC microcontrollers (MCUs) refers to all Microchip microcontroller families.

PICkit 2 and 3
Microchip's developmental device programmers with debug capability through Debug Express. See the Readme files for each tool to see which devices are supported.

Plug-ins
The MPLAB IDE/MPLAB X IDE has both built-in components and plug-in modules to configure the system for a variety of software and hardware tools. Several plug-in tools may be found under the Tools menu.

Pod
The enclosure for an in-circuit emulator or debugger. Other names are *Puck*, if the enclosure is round, and *Probe*, not be confused with logic probes.

Power-on-Reset Emulation
A software randomization process that writes random values in data RAM areas to simulate uninitialized values in RAM upon initial power application.

Pragma
A directive that has meaning to a specific compiler. Often a pragma is used to convey implementation-defined information to the compiler.

Precedence
Rules that define the order of evaluation in expressions.

Production Programmer
A production programmer is a programming tool that has resources designed in to program devices rapidly. It has the capability to program at various voltage levels and completely adheres to the programming specification. Programming a device as fast as possible is of prime importance in a production environment where time is of the essence as the application circuit moves through the assembly line.

Profile
For MPLAB SIM simulator, a summary listing of executed stimulus by register.

Program Counter
The location that contains the address of the instruction that is currently executing.

Program Counter Unit
16-bit assembler – A conceptual representation of the layout of program memory. The program counter increments by 2 for each instruction word. In an executable section, 2 program counter units are equivalent to 3 bytes. In a read-only section, 2 program counter units are equivalent to 2 bytes.

Program Memory
MPLAB IDE/MPLAB X IDE: The memory area in a device where instructions are stored. Also, the memory in the emulator or simulator containing the downloaded target application firmware.

16-bit assembler/compiler: The memory area in a device where instructions are stored.

Project
A project contains the files needed to build an application (source code, linker script files, etc.) along with their associations to various build tools and build options.

Prologue
A portion of compiler-generated code that is responsible for allocating stack space, preserving registers and performing any other machine-specific requirement specified in the run-time model. This code executes before any user code for a given function.

Prototype System
A term referring to a user's target application, or target board.

Psect
The OCG equivalent of a GCC section, short for program section. A block of code or data which is treated as a whole by the linker.

PWM Signals
Pulse Width Modulation Signals. Certain PIC MCU devices have a PWM peripheral.

Qualifier
An address or an address range used by the Pass Counter or as an event before another operation in a complex trigger.

Radix
The number base, hex, or decimal, used in specifying an address.

RAM
Random Access Memory (Data Memory). Memory in which information can be accessed in any order.

Raw Data
The binary representation of code or data associated with a section.

Read Only Memory
Memory hardware that allows fast access to permanently stored data but prevents addition to or modification of the data.

Real Time
When an in-circuit emulator or debugger is released from the halt state, the processor runs in Real Time mode and behaves exactly as the normal chip would behave. In Real Time mode, the real time trace buffer of an emulator is enabled and constantly captures all selected cycles, and all break logic is enabled. In an in-circuit emulator or debugger, the processor executes in real time until a valid breakpoint causes a halt, or until the user halts the execution.

In the simulator, real time simply means execution of the microcontroller instructions as fast as they can be simulated by the host CPU.

Recursive Calls
A function that calls itself, either directly or indirectly.

Recursion
The concept that a function or macro, having been defined, can call itself. Great care should be taken when writing recursive macros; it is easy to get caught in an infinite loop where there will be no exit from the recursion.

Re-entrant
A function that may have multiple, simultaneously active instances. This may happen due to either direct or indirect recursion or through execution during interrupt processing.

Relaxation
The process of converting an instruction to an identical, but smaller instruction. This is useful for saving on code size. MPLAB XC16 currently knows how to relax a CALL instruction into an RCALL instruction. This is done when the symbol that is being called is within +/- 32k instruction words from the current instruction.

Relocatable
An object whose address has not been assigned to a fixed location in memory.

Relocatable Section
16-bit assembler – A section whose address is not fixed (absolute). The linker assigns addresses to relocatable sections through a process called relocation.

Relocation
A process performed by the linker in which absolute addresses are assigned to relocatable sections and all symbols in the relocatable sections are updated to their new addresses.

ROM
Read Only Memory (Program Memory). Memory that cannot be modified.

Run
The command that releases the emulator from halt, allowing it to run the application code and change or respond to I/O in real time.

Run-time Model
Describes the use of target architecture resources.

Run-time Watch
A Watches window where the variables change in as the application is run. See individual tool documentation to determine how to set up a run-time watch. Not all tools support run-time watches.

Scenario
For MPLAB SIM simulator, a particular setup for stimulus control.

Section
The GCC equivalent of an OCG psect. A block of code or data which is treated as a whole by the linker.

Section Attribute
A GCC characteristic ascribed to a section (e.g., an access section).

Sequenced Breakpoints
Breakpoints that occur in a sequence. Sequence execution of breakpoints is bottom-up; the last breakpoint in the sequence occurs first.

Serialized Quick Turn Programming
Serialization allows you to program a serial number into each microcontroller device that the Device Programmer programs. This number can be used as an entry code, password or ID number.

Shell
The MPASM assembler shell is a prompted input interface to the macro assembler. There are two MPASM assembler shells: one for the DOS version and one for the Windows operating system version.

Simulator
A software program that models the operation of devices.

Single Step
This command steps though code, one instruction at a time. After each instruction, MPLAB IDE/MPLAB X IDE updates register windows, watch variables, and status displays so you can analyze and debug instruction execution. You can also single step C compiler source code, but instead of executing single instructions, MPLAB IDE/MPLAB X IDE will execute all assembly level instructions generated by the line of the high level C statement.

Skew
The information associated with the execution of an instruction appears on the processor bus at different times. For example, the executed opcodes appears on the bus as a fetch during the execution of the previous instruction, the source data address and value and the destination data address appear when the opcodes is actually executed, and the destination data value appears when the next instruction is executed. The trace buffer captures the information that is on the bus at one instance. Therefore, one trace buffer entry will contain execution information for three instructions. The number of captured cycles from one piece of information to another for a single instruction execution is referred to as the skew.

Skid
When a hardware breakpoint is used to halt the processor, one or more additional instructions may be executed before the processor halts. The number of extra instructions executed after the intended breakpoint is referred to as the skid.

Source Code
The form in which a computer program is written by the programmer. Source code is written in a formal programming language which can be translated into machine code or executed by an interpreter.

Source File
An ASCII text file containing source code.

Special Function Registers (SFRs)
The portion of data memory (RAM) dedicated to registers that control I/O processor functions, I/O status, timers or other modes or peripherals.

SQTP
See *Serialized Quick Turn Programming*.

Stack, Hardware
Locations in PIC microcontroller where the return address is stored when a function call is made.

Stack, Software
Memory used by an application for storing return addresses, function parameters, and local variables. This memory is dynamically allocated at run-time by instructions in the program. It allows for re-entrant function calls.

Stack, Compiled
A region of memory managed and allocated by the compiler in which variables are statically assigned space. It replaces a software stack when such mechanisms cannot be efficiently implemented on the target device. It precludes re-entrancy.

Static RAM or SRAM
Static Random Access Memory. Program memory you can read/write on the target board that does not need refreshing frequently.

Status Bar
The Status Bar is located on the bottom of the MPLAB IDE/MPLAB X IDE window and indicates such current information as cursor position, development mode and device, and active tool bar.

Step Into
This command is the same as Single Step. Step Into (as opposed to Step Over) follows a CALL instruction into a subroutine.

Step Over
Step Over allows you to debug code without stepping into subroutines. When stepping over a `CALL` instruction, the next breakpoint will be set at the instruction after the `CALL`. If for some reason the subroutine gets into an endless loop or does not return properly, the next breakpoint will never be reached. The Step Over command is the same as Single Step except for its handling of `CALL` instructions.

Step Out
Step Out allows you to step out of a subroutine which you are currently stepping through. This command executes the rest of the code in the subroutine and then stops execution at the return address to the subroutine.

Stimulus
Input to the simulator, i.e., data generated to exercise the response of simulation to external signals. Often the data is put into the form of a list of actions in a text file. Stimulus may be asynchronous, synchronous (pin), clocked and register.

Stopwatch
A counter for measuring execution cycles.

Storage Class
Determines the lifetime of the memory associated with the identified object.

Storage Qualifier
Indicates special properties of the objects being declared (e.g., const).

Symbol
A symbol is a general purpose mechanism for describing the various pieces which comprise a program. These pieces include function names, variable names, section names, file names, struct/enum/union tag names, etc. Symbols in MPLAB IDE/MPLAB X IDE refer mainly to variable names, function names and assembly labels. The value of a symbol after linking is its value in memory.

Symbol, Absolute
Description

System Window Control
The system window control is located in the upper left corner of windows and some dialogs. Clicking on this control usually pops up a menu that has the items "Minimize," "Maximize," and "Close."

Target
Refers to user hardware.

Target Application
Software residing on the target board.

Target Board
The circuitry and programmable device that makes up the target application.

Target Processor
The microcontroller device on the target application board.

Template
Lines of text that you build for inserting into your files at a later time. The MPLAB Editor stores templates in template files.

Term
Represents an immediate value such as a definition through the assembly .equ directive.

Toolbar
A row or column of icons that you can click on to execute MPLAB IDE/MPLAB X IDE functions.

Trace
An emulator or simulator function that logs program execution. The emulator logs program execution into its trace buffer which is uploaded to the MPLAB IDE/MPLAB X IDE trace window.

Trace Memory
Trace memory contained within the emulator. Trace memory is sometimes called the trace buffer.

Trace Macro
A macro that will provide trace information from emulator data. Since this is a software trace, the macro must be added to code, the code must be recompiled or reassembled, and the target device must be programmed with this code before trace will work.

Trigger Output
Trigger output refers to an emulator output signal that can be generated at any address or address range, and is independent of the trace and breakpoint settings. Any number of trigger output points can be set.

Trigraphs
Three-character sequences, all starting with ??, that are defined by ISO C as replacements for single characters.

Unassigned Section
A section which has not been assigned to a specific target memory block in the linker command file. The linker must find a target memory block in which to allocate an unassigned section.

Uninitialized Data
Data which is defined without an initial value. In C,

```
int myVar;
```

defines a variable which will reside in an uninitialized data section.

Upload
The Upload function transfers data from a tool, such as an emulator or programmer, to the host computer or from the target board to the emulator.

USB
Universal Serial Bus. An external peripheral interface standard for communication between a computer and external peripherals over a cable using bi-serial transmission. USB 1.0/1.1 supports data transfer rates of 12 Mbps. Also referred to as high-speed USB, USB 2.0 supports data rates up to 480 Mbps.

Vector
The memory locations that an application will jump to when either a Reset or interrupt occurs.

Volatile
A variable qualifier which prevents the compiler applying optimizations that affect how the variable is accessed in memory.

Warning
Warning
MPLAB IDE/MPLAB X IDE: An alert that is provided to warn you of a situation that would cause physical damage to a device, software file, or equipment.

16-bit assembler/compiler: Warnings report conditions that may indicate a problem, but do not halt processing.

Watch Variable
A variable that you may monitor during a debugging session in a Watches window.

Watches Window
Watches windows contain a list of watch variables that are updated at each breakpoint.

Watchdog Timer (WDT)
A timer on a PIC microcontroller that resets the processor after a selectable length of time. The WDT is enabled or disabled and set up using Configuration bits.

Workbook
For MPLAB SIM stimulator, a setup for generation of SCL stimulus.

The Microchip Website

Microchip provides online support via our website at http://www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **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 design partner 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

Product Change Notification Service

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

To register, go to http://www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: http://www.microchip.com/support

Microchip Devices Code Protection Feature

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.

Legal Notice

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 unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TempTrackr, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, Vite, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, INICnet, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, 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.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

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

ISBN: 978-1-5224-5064-1

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit http://www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: http://www.microchip.com/support Web Address: http://www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4450-2828 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-72400 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-72884388 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820