PUMPKIN™
REAL-TIME SOFTWARE

AN-25
Application Note

750 Naples Street • San Francisco, CA 94112 • (415) 584-6360 • http://www.pumpkininc.com

Building a Salvo Application with Microchip's MPLAB-C18 C Compiler and MPLAB IDE v6

Introduction

This Application Note explains how to use Microchip's (http://www.microchip.com/) MPLAB-C18 C compiler and MPLAB IDE v6 together in an integrated environment to create a multitasking Salvo application on PIC18 PICmicro devices.

We will show you how to build the example program located in `\salvo\ex\ex1\main.c` for a PIC18C452 PICmicro using MPLAB v6.30. For more information on how to write a Salvo application, please see the *Salvo User Manual*.

Before You Begin

If you have not already done so, install MPLAB-C18 and MPLAB IDE v6. Familiarize yourself with the MPLAB IDE.

Related Documents

The following Salvo documents should be used in conjunction with this manual when building Salvo applications with Microchip's MPLAB-C18 C compiler and MPLAB-IDE v6:

Salvo User Manual
Salvo Compiler Reference Manual RM-MCC18

created by Andrew E. Kalman on Jul 22, 2003 updated on Jan 6, 2004

Creating and Configuring a New Project

Creating the Project

Create a new MPLAB project under Project → Project Wizard. Select the device (18C452):

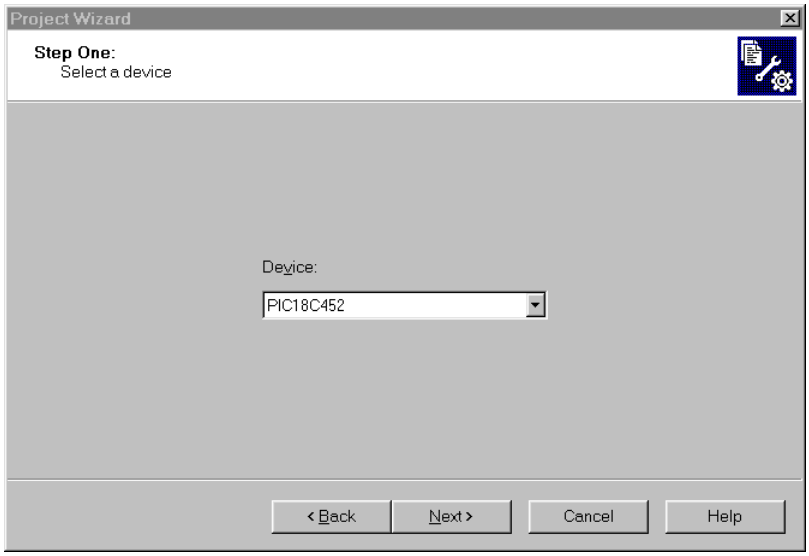

Figure 1: Selecting the Device in the Project Wizard

Click Next. Select the Microchip C18 Toolsuite:

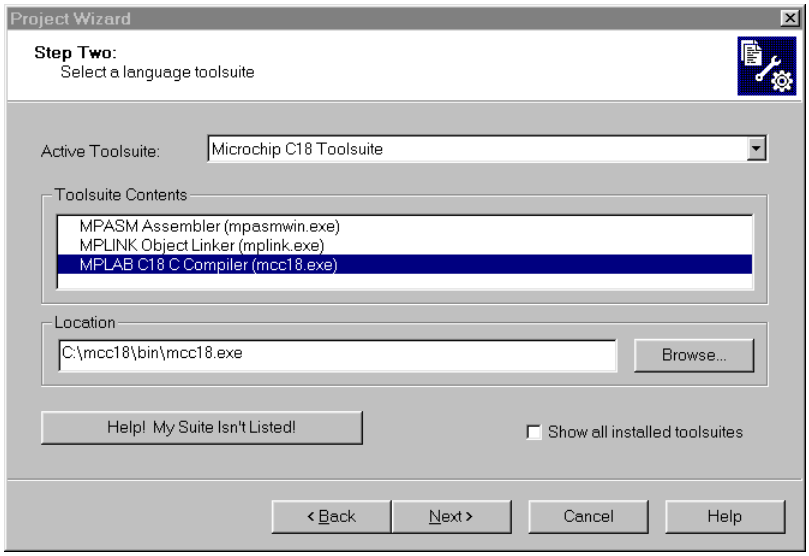

Figure 2: Selecting the ToolSuite in the Project Wizard

Click Next. Enter a Project Name (`myex1`) and Project Directory (`c:\temp`):

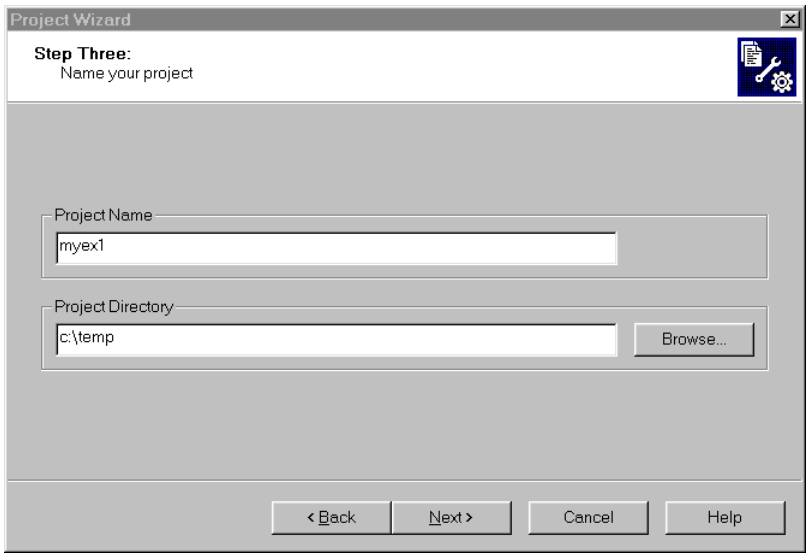

Figure 3: Selecting the ToolSuite in the Project Wizard

Click Next. Add `\salvo\src\mem.c`[1] and your project's `main.c` (and any other user source files, if present) to your project:

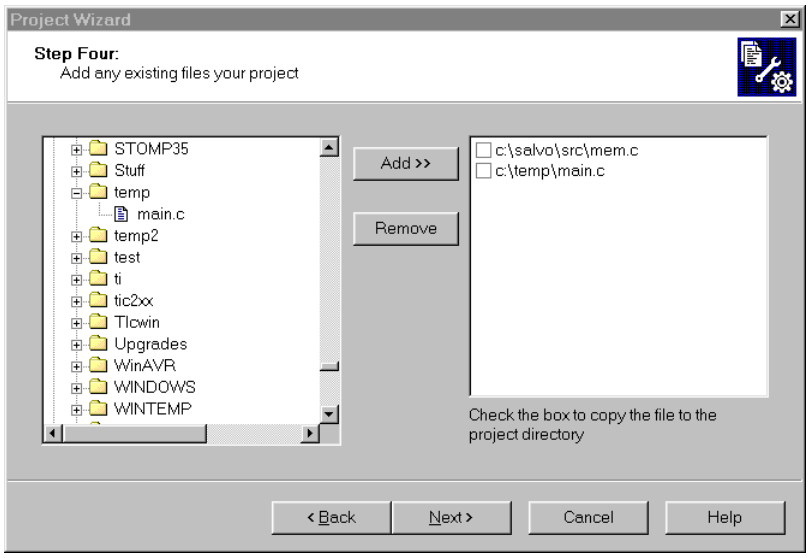

Figure 4: Adding Existing Files in the Project Wizard

Click Next, then Finish to create the project. The project window will look like this:

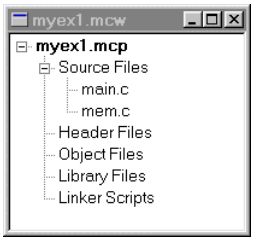

Figure 5: Project Window after Adding Source Files

Setting the Build Options

Now let's setup the project's options for Salvo's pathnames, etc. Choose Project → Build Options… → Project. Under the General tab, set the Output Directory to be the project directory. Set the Include Path to the project directory and to `\salvo\inc`. Set the Library Path and Linker-Script Path to their defaults for the MPLAB-C18 compiler:

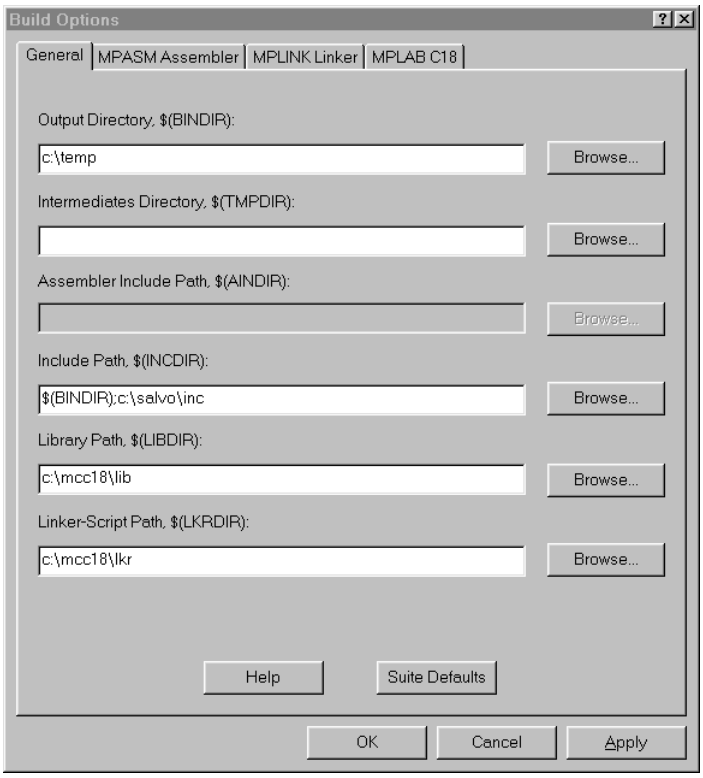

Figure 6: General Build Options

Under the MPLINK Linker tab, select Generate map file:

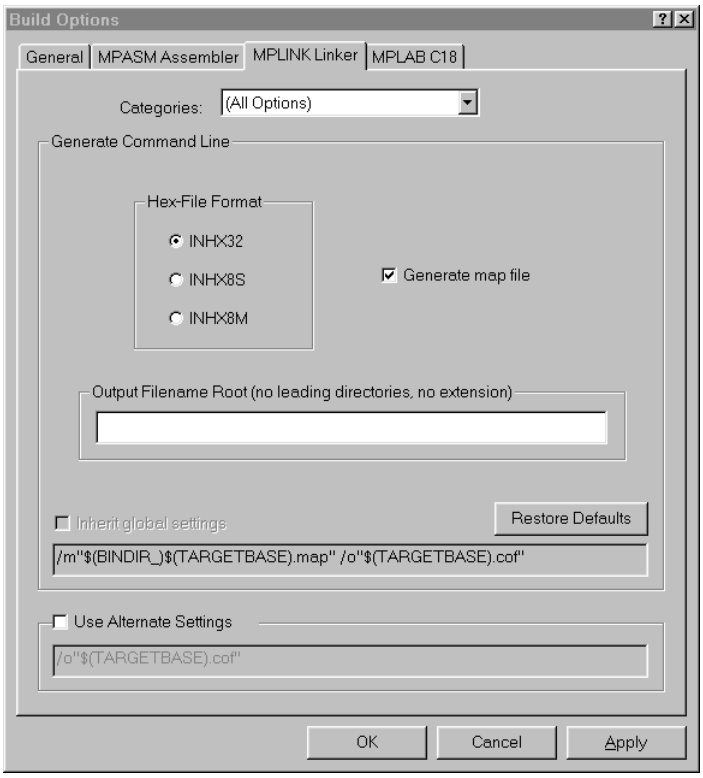

Figure 7: MPLINK Linker Build Options

Under the MPLAB C18 tab, select General under Categories and define any symbols[2] you may need for your project in the Macro Definitions window by selecting Add and entering the symbol(s), followed by OK:

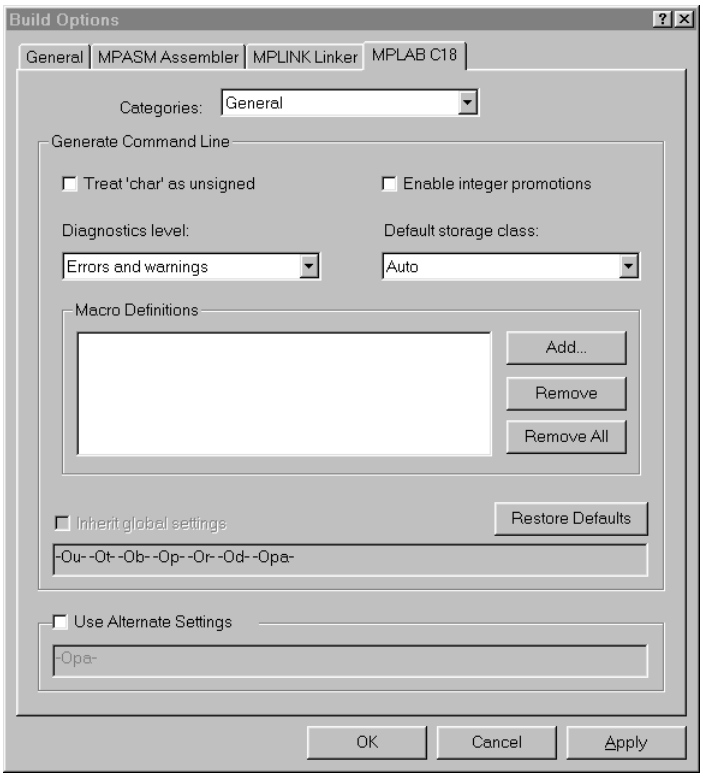

Figure 8: MPLAB C18 General Build Options

Select Optimization under Categories and select the level of optimization you wish to apply to your application:

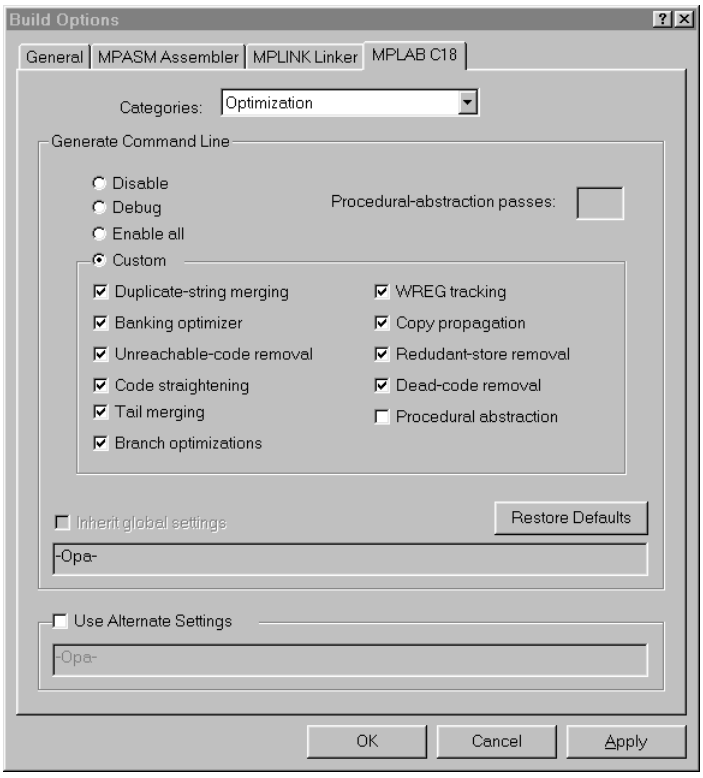

Figure 9: MPLAB C18 Optimization Build Options

Note MPLAB-C18's procedural abstraction optimization is incompatible with Salvo's context switcher. Please refer to the *Salvo Compiler Reference Manual RM-MCC18* for more information.

Click OK to finish setting your project's options.

Note This example project uses the default values for MPLAB-C18's Code, Data and Stack Models under Categories: Memory Model. Non-default values may be required when using certain Salvo libraries, etc. Please refer to the *Salvo Compiler Reference Manual RM-MCC18* for more information.

Adding the Linker Script File

In the project window, left-click on Linker Scripts, choose Add Files…, navigate to MPLAB-C18's linker script files folder (usually `c:\mcc18\lkr`) and select the linker script appropriate for your PIC18 PICmicro® MCU (`18c452.lkr` in this example):

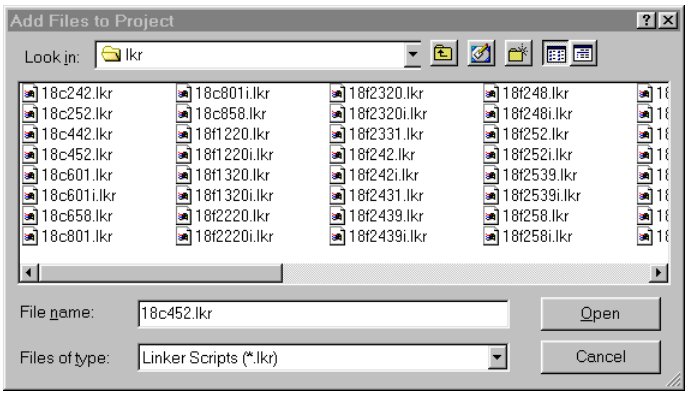

Figure 10: Adding the Linker Script File

Click Open to add the library. The project window will look like this:

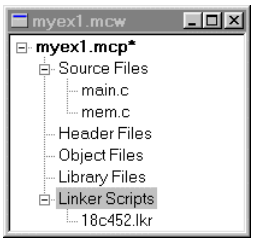

Figure 11: Project Window after Adding Linker Script File

Select Project → Save Project to save your project.

Adding Salvo-specific Files to the Project

Now it's time to add any additional Salvo files your project needs. Salvo applications can be built by linking to precompiled Salvo libraries, or with the Salvo source code files as nodes in your project.

Adding a Library

For a library build, a freeware library that's appropriate for the PIC18C452 is `sfc18sfa.lib`.[3] In the project window, left-click on Library Files, choose Add Files…, choose Files of type: Library Files (*.lib), navigate to `\salvo\lib\mcc18` and select the Salvo library `sfc18sfa.lib`:

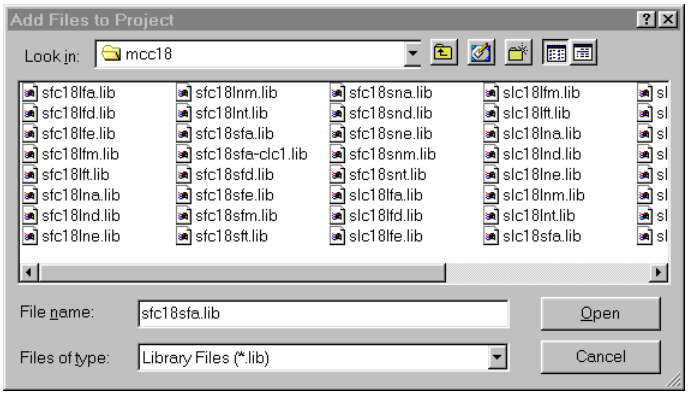

Figure 12: Adding the Salvo Library

Click Open to add the linker script file to the project. The project window will look like this:

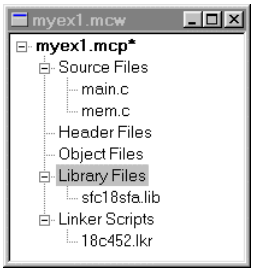

Figure 13: Project Window after Adding Salvo library

You can find more information on Salvo libraries in the *Salvo User Manual* and in the *Salvo Compiler Reference Manual RM-MCC18*.

The salvocfg.h Header File

You will also need a `salvocfg.h` file for this project. To use the library selected in Figure 12, your `salvocfg.h` should contain only:

```
#define OSUSE_LIBRARY          TRUE
#define OSLIBRARY_TYPE         OSF
#define OSLIBRARY_GLOBALS      OSF
#define OSLIBRARY_CONFIG       OSA
#define OSLIBRARY_VARIANT      OSNONE
```

Listing 1: salvocfg.h for a Library Build

Create this file and save it in your project directory, e.g. `c:\temp\salvocfg.h`. For convenience, add it to your project's by right-clicking on the Header Files folder, choosing Add Files…, and selecting the `salvocfg.h` in your project directory. The project window will now look like this:

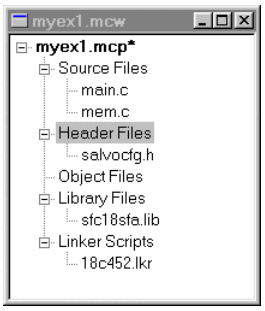

Figure 14: Project Window after Adding salvocfg.h Header File

Select Project → Save Project and proceed to *Select Project → Save Project*. *Building the Project*, below.

Adding Salvo Source Files

If you have a Salvo distribution that contains source files, you can do a *source code build* instead of a library build. The application in `\salvo\ex\ex1\main.c` contains calls to the following Salvo user services:

```
OS_Delay()              OSInit()
OS_WaitBinSem()         OSSignalBinSem()
OSCreateBinSem()        OSSched()
OSCreateTask()          OSTimer()
OSEi()
```

You must add the Salvo source files that contain these user services, as well as those that contain internal Salvo services, to

your project. The *Reference* chapter of the *Salvo User Manual* lists the source file for each user service. Internal services are in other Salvo source files. For this project, the complete list is:

```
binsem.c                mem.c
delay.c                 portpic18.c
event.c                 qins.c
idle.c                  sched.c
init.c                  timer.c
inittask.c
```

In the project window, left-click on Library Files, choose Add Files…, choose Files of type: All Source Files (*.asm, *.c), navigate to `\salvo\src` and select the Salvo source files listed above:

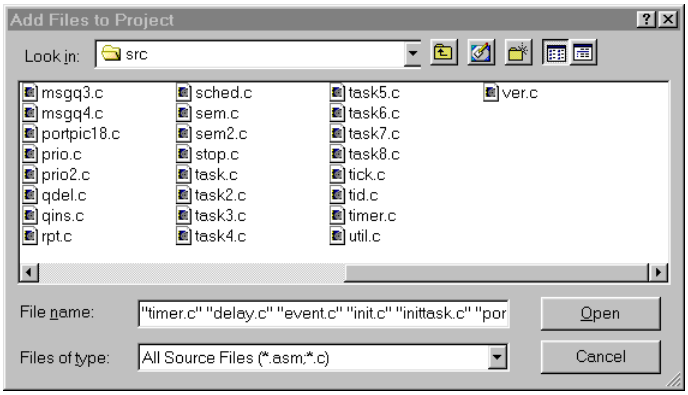

Figure 15: Adding the Salvo Source Files

Click Open to add the Salvo source files to the project. The project window will look like this:

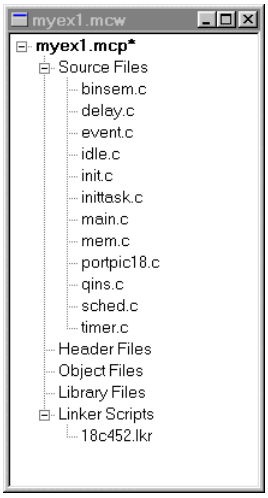

Figure 16: Project Window after Adding Salvo Source Files

The salvocfg.h Header File

You will also need a `salvocfg.h` file for this project. Configuration files for source code builds are quite different from those for library builds (see Listing 1, above). For a source code build, the `salvocfg.h` for this project contains only:

```
#define OSBYTES_OF_DELAYS           1
#define OSENABLE_IDLING_HOOK        TRUE
#define OSENABLE_BINARY_SEMAPHORES  TRUE
#define OSEVENTS                    1
#define OSTASKS                     3
```

Listing 2: salvocfg.h for a Source Code Build

Create this file and save it in your project directory, e.g. `c:\temp\salvocfg.h`. For convenience, add it to your project's by right-clicking on the Header Files folder, choosing Add Files…, and selecting the `salvocfg.h` in your project directory. The project window will now look like this:

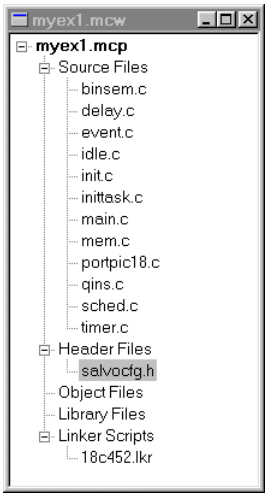

Figure 17: Project Window after Adding salvocfg.h Header File

Tip The advantage of placing the various project files in the groups shown above is that you can quickly navigate to them and open them for viewing, editing, etc.

Select Project → Save Project.

Building the Project

For a successful compile, your project must also include a header file (e.g. `#include <p18cxxx.h>`) for the particular chip you are using. Normally, this is included in each of your source files (e.g.

main.c), or in a header file that's included in each of your source files (e.g. main.h).

With everything in place, you can now build the project using Project → Build All. The Output window will reflect the MPLAB-C18 command lines:

```
Deleting intermediary files... done.
Executing: "C:\mcc18\bin\mcc18.exe" -p=18C452 "mem.c"
-fo="mem.o" /i"c:\TEMP" /i"c:\salvo\inc" -Opa-
Executing: "C:\mcc18\bin\mcc18.exe" -p=18C452 "main.c"
-fo="main.o" /i"c:\TEMP" /i"c:\salvo\inc" -Opa-
Executing: "C:\mcc18\bin\mcc18.exe" -p=18C452 "timer.c"
-fo="timer.o" /i"c:\TEMP" /i"c:\salvo\inc" -Opa-
Executing: "C:\mcc18\bin\mcc18.exe" -p=18C452 "delay.c"
-fo="delay.o" /i"c:\TEMP" /i"c:\salvo\inc" -Opa-
Executing: "C:\mcc18\bin\mcc18.exe" -p=18C452 "event.c"
-fo="event.o" /i"c:\TEMP" /i"c:\salvo\inc" -Opa-
Executing: "C:\mcc18\bin\mcc18.exe" -p=18C452 "init.c"
-fo="init.o" /i"c:\TEMP" /i"c:\salvo\inc" -Opa-
Executing: "C:\mcc18\bin\mcc18.exe" -p=18C452 "inittask.c"
-fo="inittask.o" /i"c:\TEMP" /i"c:\salvo\inc" -Opa-
Executing: "C:\mcc18\bin\mcc18.exe" -p=18C452 "portpic18.c"
-fo="portpic18.o" /i"c:\TEMP" /i"c:\salvo\inc" -Opa-
Executing: "C:\mcc18\bin\mcc18.exe" -p=18C452 "qins.c"
-fo="qins.o" /i"c:\TEMP" /i"c:\salvo\inc" -Opa-
Executing: "C:\mcc18\bin\mcc18.exe" -p=18C452 "sched.c"
-fo="sched.o" /i"c:\TEMP" /i"c:\salvo\inc" -Opa-
Executing: "C:\mcc18\bin\mcc18.exe" -p=18C452 "binsem.c"
-fo="binsem.o" /i"c:\TEMP" /i"c:\salvo\inc" -Opa-
Executing: "C:\mcc18\bin\mcc18.exe" -p=18C452 "idle.c"
-fo="idle.o" /i"c:\TEMP" /i"c:\salvo\inc" -Opa-
Executing: "C:\Program Files\MPLAB IDE
v6\MCHIP_Tools\mplink.exe" /l"c:\mcc18\lib" /k"c:\mcc18\lkr"
"C:\mcc18\lkr\18c452.lkr" "C:\salvo\src\mem.o" "C:\temp\main.o"
"C:\salvo\src\timer.o" "C:\salvo\src\delay.o"
"C:\salvo\src\event.o" "C:\salvo\src\init.o"
"C:\salvo\src\inittask.o" "C:\salvo\src\portpic18.o"
"C:\salvo\src\qins.o" "C:\salvo\src\sched.o"
"C:\salvo\src\binsem.o" "C:\salvo\src\idle.o"
/m"c:\TEMP\myex1.map" /o"myex1.cof"
MPLINK 3.50, Linker
Copyright (c) 2003 Microchip Technology Inc.
Errors    : 0

MP2COD 3.50, COFF to COD File Converter
Copyright (c) 2003 Microchip Technology Inc.
Errors    : 0

MP2HEX 3.50, COFF to HEX File Converter
Copyright (c) 2003 Microchip Technology Inc.
Errors    : 0

Loaded C:\temp\myex1.cof
BUILD SUCCEEDED: Tue Jul 22 20:33:02 2003
```

Listing 3: Build Results for A Successful Source-Code Build

The map (*.map) file located in the project's directory contains address, symbol and other useful information:

```
MPLINK 3.50, Linker
Linker Map File - Created Tue Jul 22 20:33:01 2003

                                  Section Info
                 Section       Type    Address   Location Size(Bytes)
               ---------  ---------  ---------  ---------  ---------
              _entry_scn       code   0x000000    program   0x000006
           IntVectorHigh       code   0x000008    program   0x000006
[SNIP]
                               Program Memory Usage
                                Start         End
                              ---------   ---------
                               0x000000    0x000005
                               0x000008    0x00000d
                               0x00002a    0x000037
                               0x0000c8    0x000cfc
            3151 out of 32786 program addresses used, program memory utilization is
9%

                                  Symbols - Sorted by Name
                      Name    Address   Location    Storage File
                  ---------  ---------  ---------  --------- ---------
                 IntVector   0x0009e6    program     extern C:\temp\main.c
              IntVectorHigh   0x000008    program     extern C:\temp\main.c
             OSCreateBinSem   0x000336    program     extern C:\salvo\src\binsem.c
[SNIP]
                                  Symbols - Sorted by Address
                      Name    Address   Location    Storage File
                  ---------  ---------  ---------  --------- ---------
                    _entry   0x000000    program     extern
C:\mcc18\src\startup\c018i.c
          ___return_lbl00000   0x000004    program     static
[SNIP]
```

Listing 4: Map File for a Source-Code Build

Note The projects supplied in the Salvo for PICmicro® MCUs distributions contain additional help files – see the `abstract.txt` file that accompanies each project or group of projects.

Testing the Application

You can test and debug this application with full source code integration in any of the MPLAB debugging environments. For example, to use the simulator, choose Debugger → Select Tool → MPLAB SIM. Open the Stopwatch window via Debugger → Stopwatch. After a successful build, open the project's `main.c` (i.e. `\salvo\ex\ex1\main.c`), set a breakpoint on the `PORTB ^= 0x08;` line of `Task3()`, and select Debugger → Run. Program execution will stop at the breakpoint in `Task3()`. Now zero the stopwatch in the Stopwatch window, select Debug → Run again, and wait until execution stops. The Stopwatch window now displays an elapsed time of 400ms (40 times 10ms, the `TMR0`-driven system tick rate in this application for a 4MHz clock).

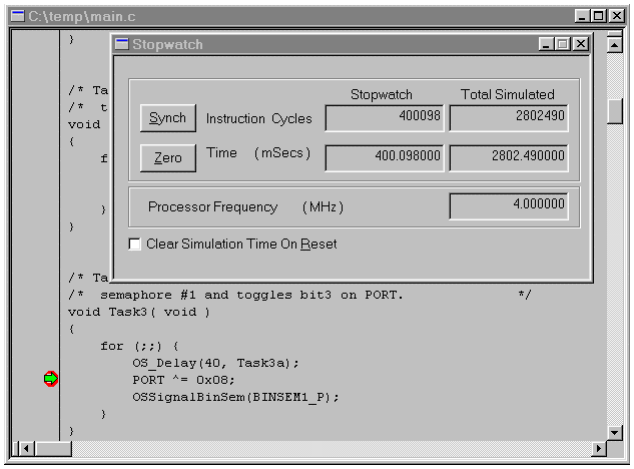

Figure 18: Measuring 400ms of Task Delay in the Simulator via a Breakpoint

Note The 98 extra microseconds (400.098ms – 400ms) shown in the Stopwatch window of Figure 18 are due to unavoidable jitter in the system timer – well under the system tick interval of 10ms (10,000 instruction cycles in this example). See the *Salvo User Manual* for more information on the system timer.

If you are doing a full source-code build, you can also trace program execution through the Salvo source code. Select Debugger → Reset → Processor Reset, Debugger → Breakpoints → Remove All → OK, and set a breakpoint at the first call to `OSCreateTask()` in `main.c`. Select Debugger → Run. Execution will stop in `main.c` at the call to `OSCreateTask()`. Now choose Debugger → Step Into. The `\salvo\src\inittask.c` file window will open, and you can step through and observe the operation of `OSCreateTask()`.

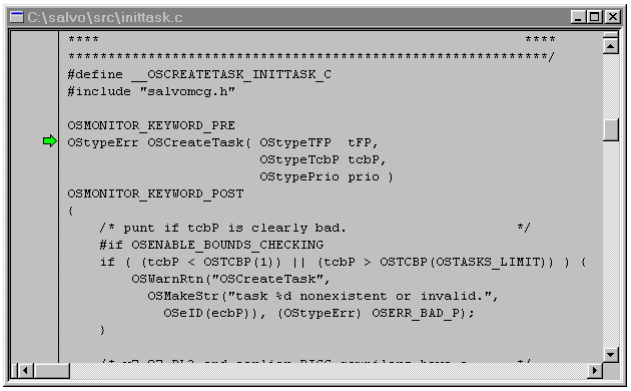

Figure 19: Stepping Through Salvo Source Code

Troubleshooting

Cannot Find and/or Read Include File(s)

If you fail to add `\salvo\inc` to the project's include paths (see Figure 6, above) the compiler will generate an error like this one:

```
Deleting intermediary files... done.
Executing: "C:\mcc18\bin\mcc18.exe" -p=18C452
"mem.c" -fo="mem.o" /i"c:\TEMP" -Opa-
C:\salvo\src\mem.c:30: unable to locate 'salvo.h'
C:\salvo\src\mem.c:190: unable to locate
'salvoprg.h'
C:\salvo\src\mem.c:204: unable to locate
'salvoprg.h'
C:\salvo\src\mem.c:211: unable to locate
'salvoprg.h'
error 1 spawning C:\mcc18\bin\cpp18
Halting build on first failure as requested.
BUILD FAILED: Tue Jul 22 21:51:34 2003
```

Figure 20: Compiler Error due to Missing \salvo\inc Include Path

By adding `\salvo\inc` to the project's include path, you enable the compiler to find the main Salvo header file `salvo.h`, as well as other included Salvo header files.

If you fail to add the project's own directory to the project's include paths (see Figure 6, above) the compiler will generate an error like this one:

```
Deleting intermediary files... done.
Executing: "C:\mcc18\bin\mcc18.exe" -p=18C452
"mem.c" -fo="mem.o" /i"c:\salvo\inc" -Opa-
c:\salvo\inc\salvo.h:343: unable to locate
'salvocfg.h'
error 1 spawning C:\mcc18\bin\cpp18
Halting build on first failure as requested.
BUILD FAILED: Tue Jul 22 21:52:23 2003
```

Figure 21: Compiler Error due to Missing Project Include Path

By adding the project's own directory to the project's include path, you enable the compiler to find the project-specific header file `salvocfg.h`.

Cannot Find Symbol Definitions

If you fail to add `\salvo\src\mem.c` to the project's source files (see *Creating the Project* and *Figure 4*), the linker will be unable to find one or more of Salvo's global objects, e.g.:

```
Executing: "C:\Program Files\MPLAB IDE v6\MCHIP_Tools\mplink.exe" /l"C:\mcc18\lib"
/k"c:\mcc18\lkr" "C:\mcc18\lkr\18c452.lkr" "C:\salvo\tut\tu4\main.o"
"C:\salvo\lib\mcc18\sfc18sfe.lib" /m"\SALVO\TUT\TU4\SYSE\tu4lite.map"
/o"tu4lite.cof"
MPLINK 3.60, Linker
Copyright (c) 2003 Microchip Technology Inc.
Error - could not find definition of symbol 'OSecbArea' in file
'C:\salvo\tut\tu4\main.o'.
Errors    : 1

BUILD FAILED: Mon Jan 05 14:22:28 2004
```

Figure 22: Linker Error due to Missing Salvo mem.c

The solution is to always have Salvo's `mem.c` in the list of the project's Source Files (see *Figure 5*).

Similarly, if there is a mismatch between the `OSLIBRARY_XYZ` configuration options in the project's `salvocfg.h`, and the Salvo library chosen for the project, the linker may again be unable to find the definitions for certain Salvo global objects, e.g.:

```
Executing: "C:\Program Files\MPLAB IDE v6\MCHIP_Tools\mplink.exe" /l"C:\mcc18\lib"
/k"c:\mcc18\lkr" "C:\mcc18\lkr\18c452.lkr" "C:\salvo\tut\tu4\main.o"
"C:\salvo\src\mem.o" "C:\salvo\lib\mcc18\sfc18sfa.lib"
/m"\SALVO\TUT\TU4\SYSE\tu4lite.map" /o"tu4lite.cof"
MPLINK 3.60, Linker
Copyright (c) 2003 Microchip Technology Inc.
Error - could not find definition of symbol 'OSdelayQP' in file 'init.o'.
Errors    : 1

BUILD FAILED: Mon Jan 05 14:33:33 2004
```

Figure 23: Linker Error due to Mismatch between OSLIBRARY_CONFIG (OSE) and Selected library (sfpc18sfa.lib)

This occurs because Salvo functions[4] are attempting to initialize objects that are not enabled by the `OSLIBRARY_XYZ` configuration options in force. The solution is to ensure that the `OSLIBRARY_XYZ` configuration options in the project's `salvocfg.h` are appropriate for the selected Salvo library.

Example Projects

Example projects for MPLAB-C18 can be found in the `salvo\tut\tu1-6\syse` directories The MPLAB Include Path for each of these projects is set to `salvo\tut\tu1\syse`, and each project defines the `SYSE` symbol.

Complete projects using Salvo freeware libraries are contained in the MPLAB project file `salvo\tut\tu1-6\syse\tu1-6lite.mcp`. These projects also define the `MAKE_WITH_FREE_LIB` symbol.

Complete projects using Salvo standard libraries are contained in the MPLAB project file `salvo\tut\tu1-6\syse\tu1-6le.mcp`. These projects also define the `MAKE_WITH_STD_LIB` symbol.

Complete projects using Salvo source code are contained in the MPLAB project file `salvo\tut\tu1-6\syse\tu1-6pro.mcp`. These projects also define the `MAKE_WITH_SOURCE` symbol.

---

1 Do not copy `\salvo\src\mem.c` to your project directory!

2 The Salvo project upon which this Application Note is based (`ex1lite.mcp`) supports a wide variety of targets and compilers. For use with MPLAB-C18, it requires the `SYSE` defined symbol, as well as the symbols `MAKE_WITH_FREE_LIB` for library builds. When you write your own projects, you may not require any symbols.

3 This library was compiled using the small memory model, and matches the node properties for `main.c`. The corresponding standard library is `slc18sfa.lib`.

4 In this case, `OSInit()` in `init.c`.