PUMPKIN™
REAL-TIME SOFTWARE

AN-26
Application Note

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

Building a Salvo Application with HI-TECH's PICC and PICC-18 C Compilers and Microchip's MPLAB IDE v6

Introduction

This Application Note explains how to use HI-TECH's (http://www.htsoft.com/) PICC and PICC-18 C compilers 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 PICC-18 and 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 PICC and/or PICC-18, as well as 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 HI-TECH's PICC and PICC-18 C compilers and MPLAB-IDE v6:

Salvo User Manual
Salvo Compiler Reference Manual RM-PICC
Salvo Compiler Reference Manual RM-PICC18

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):

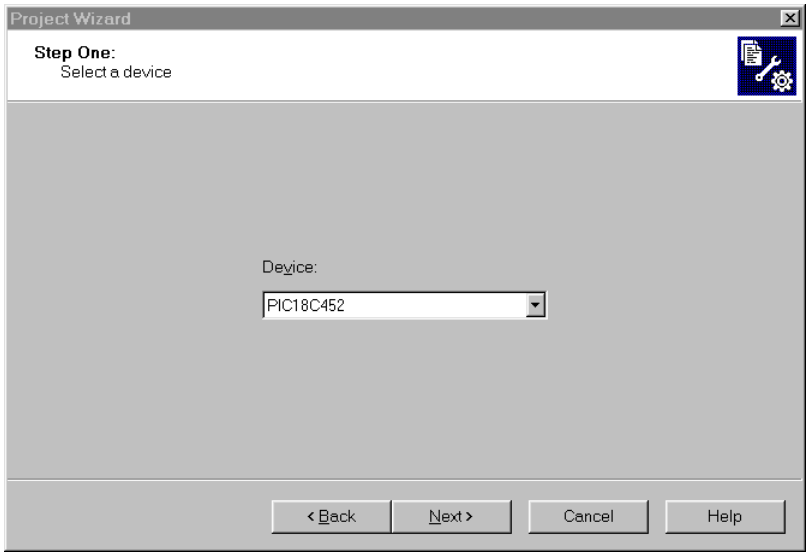

Figure 1: Selecting the Device in the Project Wizard

Click Next. Select the HI-TECH PICC-18 Toolsuite:

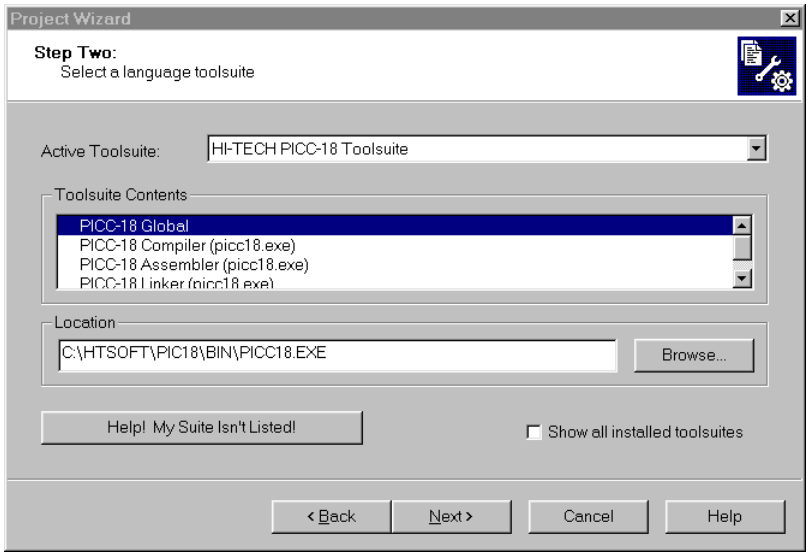

Figure 2: Selecting the ToolSuite in the Project Wizard

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

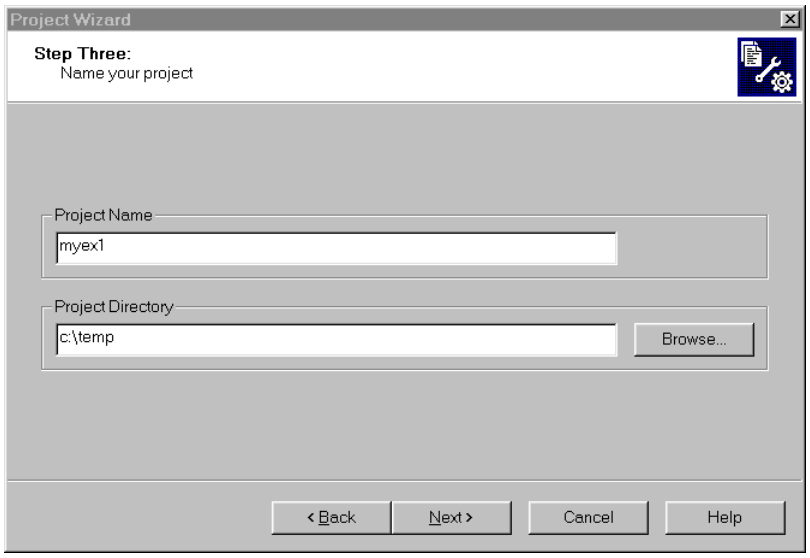

Figure 3: Selecting the ToolSuite in the Project Wizard

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:

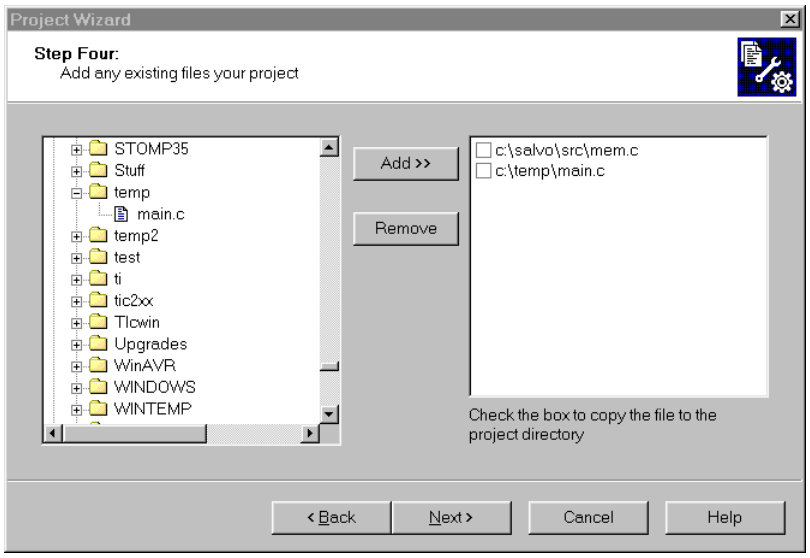

Figure 4: Adding Existing Files in the Project Wizard

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

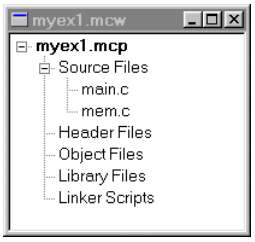

Figure 5: Project Window after Adding Source Files

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`:

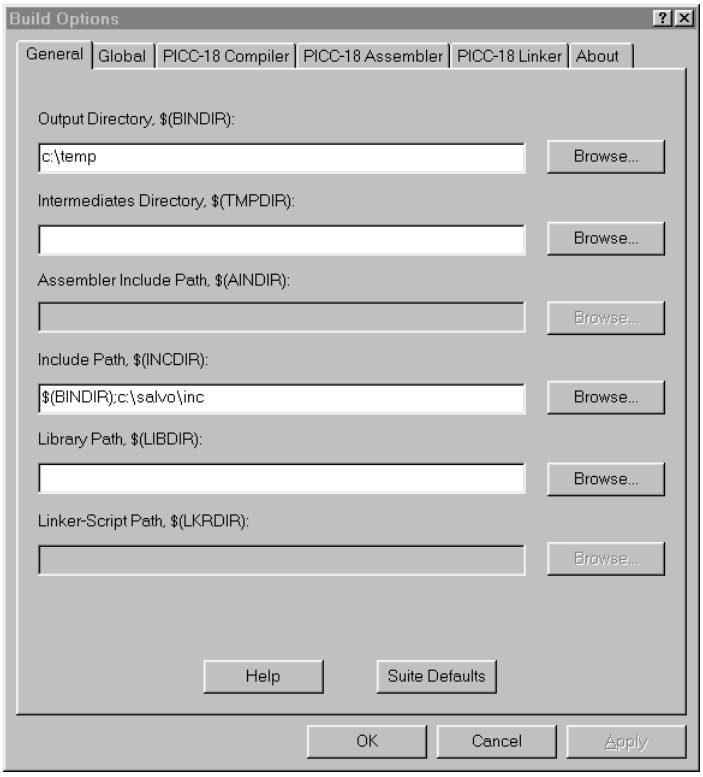

Figure 6: General Build Options

Note The screens below are for the PICC-18 compiler. Build Options for the PICC compiler will be similar.

Under the PICC-18 General tab, select (All Options) under Categories and make the appropriate selections for your project.

Tip For the PICC-18 compiler, Use 24-bit pointers to program space and Memory Model must be set to match the Salvo library used when building this application. See *Salvo Compiler Reference Manual RM-PICC18* for more information.

Select OK to continue:

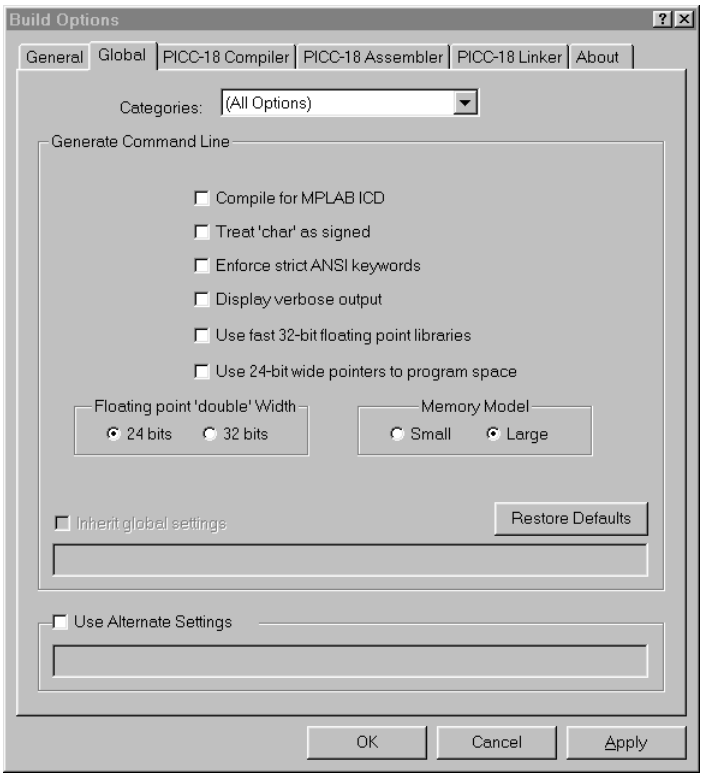

Figure 7: PICC-18 Compiler General Build Options

Under the PICC-18 Compiler 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. Also, set the desired Global optimization level, and select Enable assembler optimization:

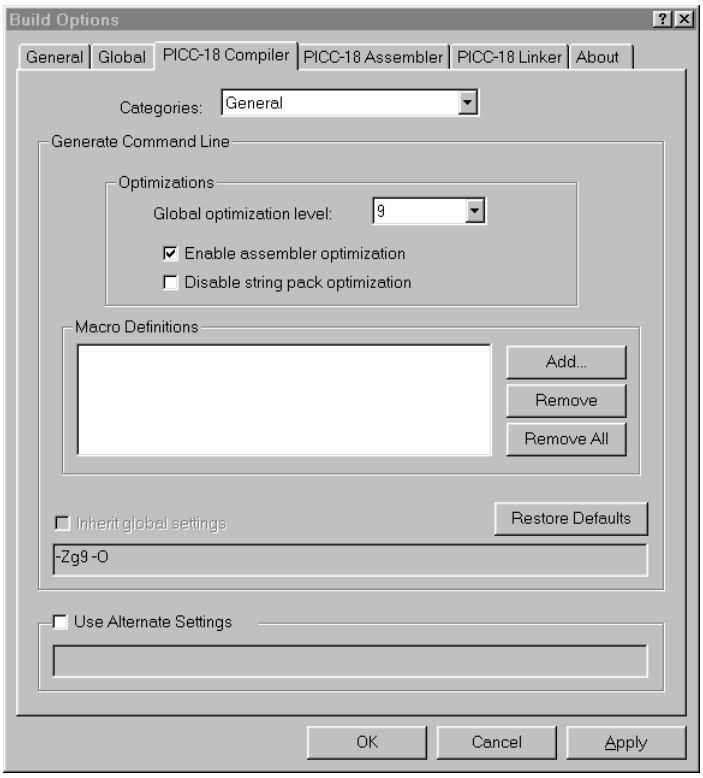

Figure 8: PICC-18 Compiler General Build Options

Under the PICC-18 Assembler tab, select Enable optimization:

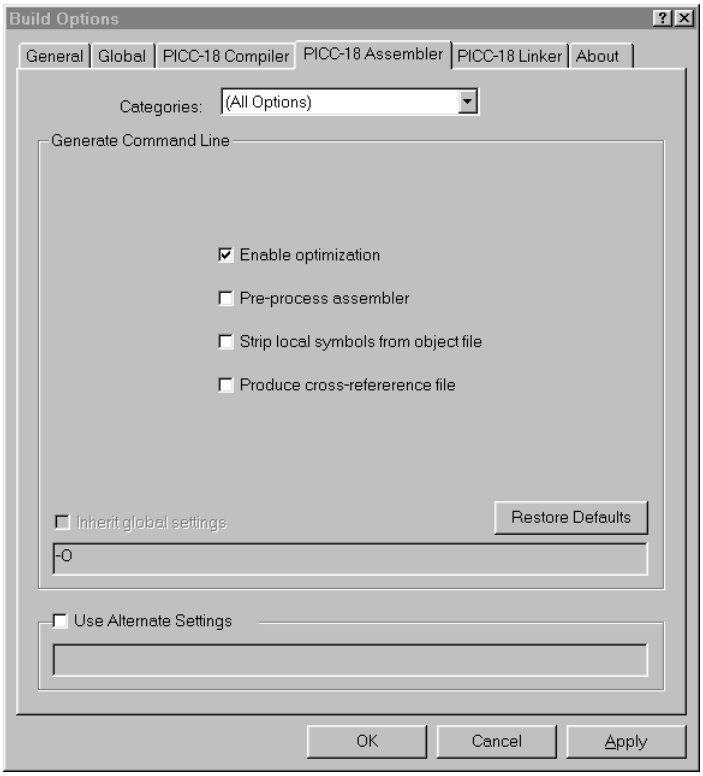

Figure 9: PICC-18 Assembler Build Options

Under the PICC-18 Linker tab, select Generate map file:

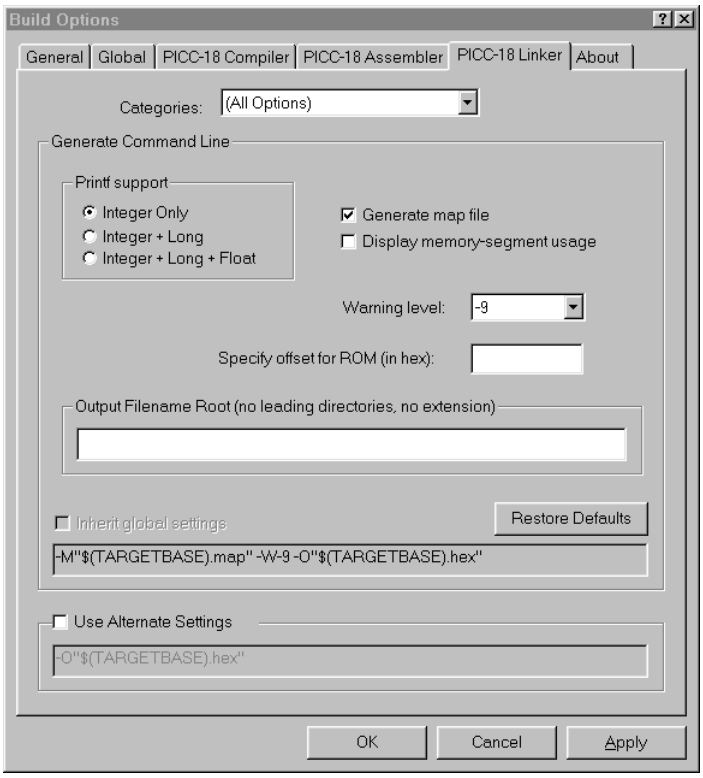

Figure 10: PICC-18 Linker Build Options

Click OK to finish setting your project's options.

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 `sfp80lab.lib`. In the project window, left-click on Library Files, choose Add Files…, choose Files of type: Library Files (*.lib), navigate to `\salvo\lib\htpicc18` and select the Salvo library `sfp80lab.lib`:

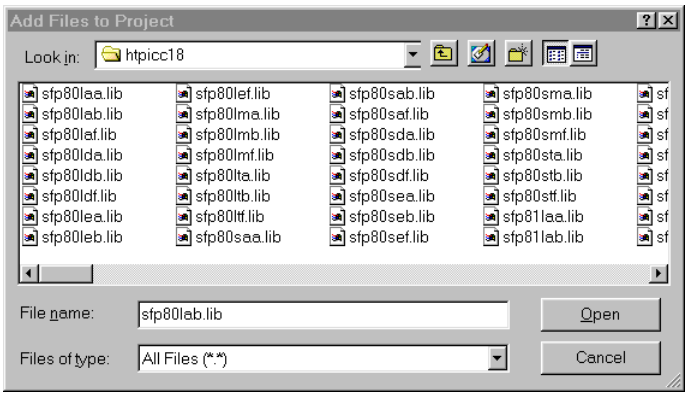

Figure 11: Adding the Salvo Library

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

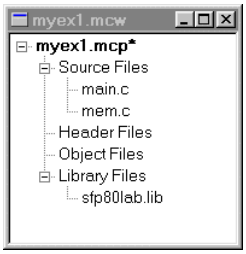

Figure 12: Project Window after Adding Salvo library

You can find more information on Salvo libraries in the *Salvo User Manual* and in the *Salvo Compiler Reference Manual RM-PICC18*.

The salvocfg.h Header File

You will also need a `salvocfg.h` file for this project. To use the library selected in Figure 11, your `salvocfg.h` should contain only:

```
#define OSUSE_LIBRARY          TRUE
#define OSLIBRARY_TYPE         OSF
#define OSLIBRARY_CONFIG       OSA
#define OSLIBRARY_VARIANT      OSB
```

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:

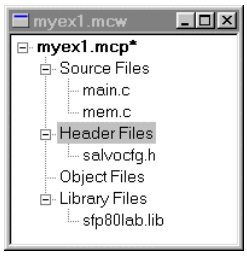

Figure 13: Project Window after Adding salvocfg.h Header File

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                inittask.c
delay.c                 mem.c
event.c                 qins.c
idle.c                  sched.c
init.c                  timer.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:

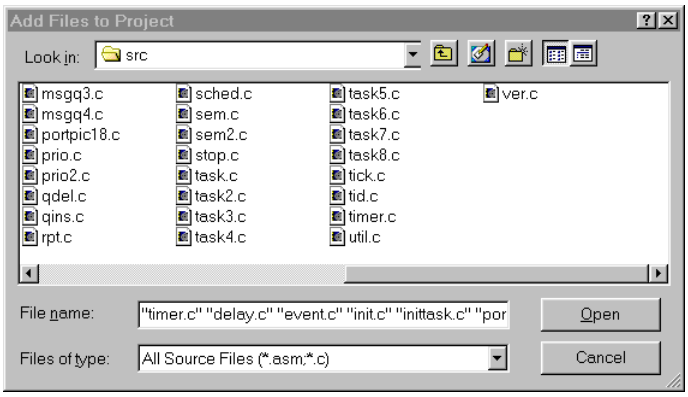

Figure 14: Adding the Salvo Source Files

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

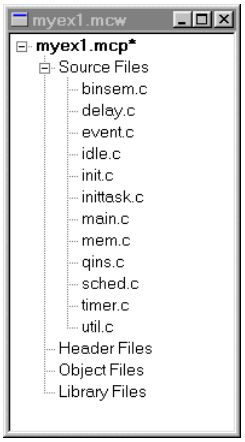

Figure 15: Project Window after Adding Salvo Source Files

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:

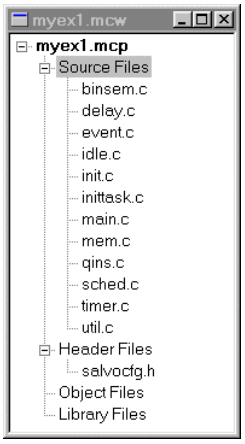

Figure 16: Project Window after Adding salvocfg.h Header File

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 <pic18.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 PICC-18 command lines:

```
Deleting intermediary files... done.
Executing: "C:\HTSOFT\PIC18\BIN\PICC18.EXE" -C -E"mem.cce" "mem.c" -O"mem.obj"
-I"c:\TEMP" -I"c:\salvo\inc" -Q -MPLAB -18C452 -Zg9 -O
Executing: "C:\HTSOFT\PIC18\BIN\PICC18.EXE" -C -E"main.cce" "main.c" -O"main.obj"
-I"c:\TEMP" -I"c:\salvo\inc" -Q -MPLAB -18C452 -Zg9 -O
Executing: "C:\HTSOFT\PIC18\BIN\PICC18.EXE" -C -E"timer.cce" "timer.c" -O"timer.obj"
-I"c:\TEMP" -I"c:\salvo\inc" -Q -MPLAB -18C452 -Zg9 -O
Executing: "C:\HTSOFT\PIC18\BIN\PICC18.EXE" -C -E"delay.cce" "delay.c" -O"delay.obj"
-I"c:\TEMP" -I"c:\salvo\inc" -Q -MPLAB -18C452 -Zg9 -O
Executing: "C:\HTSOFT\PIC18\BIN\PICC18.EXE" -C -E"event.cce" "event.c" -O"event.obj"
-I"c:\TEMP" -I"c:\salvo\inc" -Q -MPLAB -18C452 -Zg9 -O
Executing: "C:\HTSOFT\PIC18\BIN\PICC18.EXE" -C -E"idle.cce" "idle.c" -O"idle.obj"
-I"c:\TEMP" -I"c:\salvo\inc" -Q -MPLAB -18C452 -Zg9 -O
Executing: "C:\HTSOFT\PIC18\BIN\PICC18.EXE" -C -E"init.cce" "init.c" -O"init.obj"
-I"c:\TEMP" -I"c:\salvo\inc" -Q -MPLAB -18C452 -Zg9 -O
Executing: "C:\HTSOFT\PIC18\BIN\PICC18.EXE" -C -E"inittask.cce" "inittask.c"
-O"inittask.obj" -I"c:\TEMP" -I"c:\salvo\inc" -Q -MPLAB -18C452 -Zg9 -O
Executing: "C:\HTSOFT\PIC18\BIN\PICC18.EXE" -C -E"qins.cce" "qins.c" -O"qins.obj"
-I"c:\TEMP" -I"c:\salvo\inc" -Q -MPLAB -18C452 -Zg9 -O
Executing: "C:\HTSOFT\PIC18\BIN\PICC18.EXE" -C -E"sched.cce" "sched.c" -O"sched.obj"
-I"c:\TEMP" -I"c:\salvo\inc" -Q -MPLAB -18C452 -Zg9 -O
Executing: "C:\HTSOFT\PIC18\BIN\PICC18.EXE" -C -E"binsem.cce" "binsem.c"
-O"binsem.obj" -I"c:\TEMP" -I"c:\salvo\inc" -Q -MPLAB -18C452 -Zg9 -O
Executing: "C:\HTSOFT\PIC18\BIN\PICC18.EXE" -C -E"util.cce" "util.c" -O"util.obj"
-I"c:\TEMP" -I"c:\salvo\inc" -Q -MPLAB -18C452 -Zg9 -O
Executing: "C:\HTSOFT\PIC18\BIN\PICC18.EXE" -E"myex1.lde" "C:\salvo\src\mem.obj"
"C:\temp\main.obj" "C:\salvo\src\timer.obj" "C:\salvo\src\delay.obj"
"C:\salvo\src\event.obj" "C:\salvo\src\idle.obj" "C:\salvo\src\init.obj"
"C:\salvo\src\inittask.obj" "C:\salvo\src\qins.obj" "C:\salvo\src\sched.obj"
"C:\salvo\src\binsem.obj" "C:\salvo\src\util.obj" -Q -MPLAB -18C452 -M"myex1.map"
-W-9 -O"myex1.cof"

Memory Usage Map:

Program ROM   $000000 - $000003  $000004 (       4) bytes
Program ROM   $000006 - $00092B  $000926 (    2342) bytes
                                 $00092A (    2346) bytes total Program ROM

RAM data      $0000F6 - $0000FF  $00000A (      10) bytes
RAM data      $0005BB - $0005FF  $000045 (      69) bytes
                                 $00004F (      79) bytes total RAM data

Near RAM      $000000 - $00000F  $000010 (      16) bytes total Near RAM
ROM data      $000004 - $000004  $000001 (       1) bytes total ROM data

Program statistics:

Total ROM used     2347 bytes (7.2%)
Total RAM used       95 bytes (6.2%)
Near RAM used        16 bytes (12.5%)

Loaded C:\temp\myex1.cof
BUILD SUCCEEDED: Tue Jul 22 22:27:30 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:

```
HI-TECH Software PICC18 Compiler V8.20PL4

Linker command line:

-z -Mmyex1.map -ol.obj \
  -ppowerup=00h,intcode=08h,intcodelo=018h,init,end_init -ACOMRAM=00h-07Fh \
  -ptemp=COMRAM -ARAM=0-0FFhx6 -ABIGRAM=0-05FFh -pramtop=0600h \
  -ACODE=00h-07FFFh -pconfig=0300000h,idloc=0200000h,eeprom_data=0f00000h \
  -pconst=end_init+0600h \
  -prbss=COMRAM,rbit=COMRAM,rdata=COMRAM,nvrram=COMRAM,nvbit=COMRAM \
  -pstruct=COMRAM -pnvram=-600h \
  -pintsave_regs=BIGRAM,bigbss=BIGRAM,bigdata=BIGRAM -pdata=RAM,param \
  -pidata=CODE,irdata=CODE,ibigdata=CODE -Q18C452 -W-9 -h+myex1.sym -E \
  -EC:\WINDOWS\TEMP\_3VV0OM1.AAA -ver=PICC18#V8.20PL4 \
  C:\HTSOFT\PIC18\LIB\picrt80l.obj C:\salvo\src\mem.obj C:\temp\main.obj \
  C:\salvo\src\timer.obj C:\salvo\src\delay.obj C:\salvo\src\event.obj \
  C:\salvo\src\idle.obj C:\salvo\src\init.obj C:\salvo\src\inittask.obj \
  C:\salvo\src\qins.obj C:\salvo\src\sched.obj C:\salvo\src\binsem.obj \
  C:\salvo\src\util.obj C:\HTSOFT\PIC18\LIB\pic80l-c.lib

Object code version is 3.7

Machine type is 18C452

Call graph:

*_main size 0,0 offset 0
     _OSInit
     _OSCreateTask size 5,0 offset 0
         _OSInitPrioTask size 3,0 offset 5
         _OSInsPrioQ size 4,0 offset 6
[SNIP]

               Name          Link     Load   Length Selector Space Scale
C:\HTSOFT\PIC18\LIB\picrt80l.obj
               end_init        38       38        4       C     0
C:\salvo\src\mem.obj
               nvram          5DA      5DA       26     5DA     1
[SNIP]

SEGMENTS  Name      Load Length     Top Selector    Space Class

               temp            000000   000010    000010           0       1  COMRAM
               powerup         000000   000005    000005           0       0  CODE
[SNIP]

UNUSED ADDRESS RANGES

          BIGRAM           000010-0000F5
                           000100-0005BA
[SNIP]

                                  Symbol Table

?_OSCreateBinSem  param        0000F6  ?_OSCreateTask    param        0000F6
?_OSDelay         param        0000F6  ?_OSInitPrioTask  param        0000FB
[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).

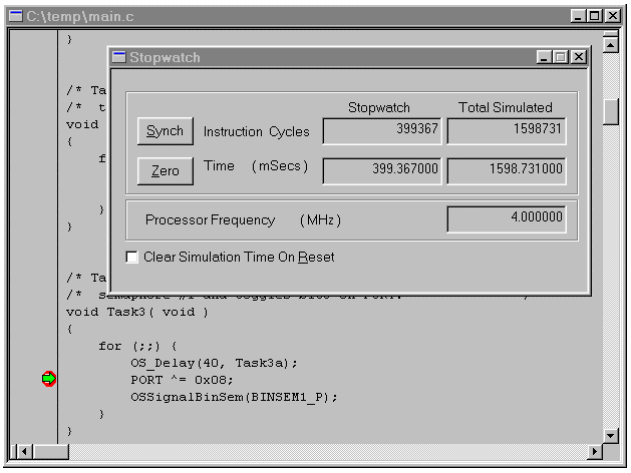

Figure 17: Measuring 400ms of Task Delay in the Simulator via a Breakpoint

Note The 633 microseconds (400ms-399.367ms) that are "short" in the Stopwatch window of Figure 17 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()`.

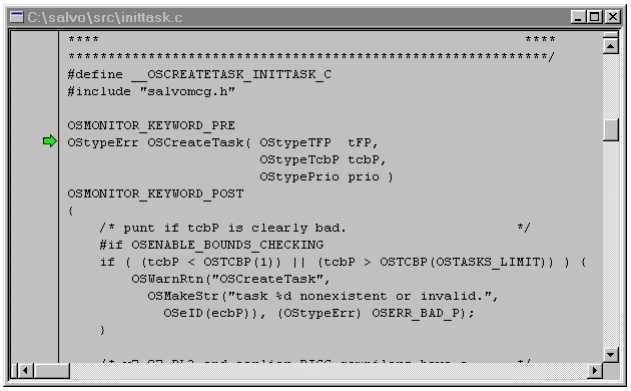

Figure 18: Stepping Through Salvo Source Code

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:

```
Executing: "C:\HTSOFT\PIC18\BIN\PICC18.EXE" -C -E"mem.cce" "mem.c" -O"mem.obj"
-I"c:\TEMP" -Q -MPLAB -18C452 -Zg9 -O
Error[000] C:\salvo\src\mem.c 30 : Cannot open include file "salvo.h"
Halting build on first failure as requested.
BUILD FAILED: Tue Jul 22 22:42:32 2003
```

Figure 19: 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:

```
Executing: "C:\HTSOFT\PIC18\BIN\PICC18.EXE" -C -E"mem.cce" "mem.c" -O"mem.obj"
-I"c:\salvo\inc" -Q -MPLAB -18C452 -Zg9 -O
Error[000] c:\salvo\inc\salvo.h 343 : Cannot open include file "salvocfg.h"
Halting build on first failure as requested.
BUILD FAILED: Tue Jul 22 22:42:58 2003
```

Figure 20: 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`.

Undefined Symbols

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:\HTSOFT\PIC18\BIN\PICC18.EXE" -C -E"main.cce" "main.c" -O"main.obj"
-I"\SALVO\TUT\TU4\SYSF" -I"\SALVO\TUT\TU4\SYSF\..\..\tu1"
-I"\SALVO\TUT\TU4\SYSF\..\..\..\inc" -Q -MPLAB -18C452 -Zg9 -O -DSYSF
-DMAKE_WITH_FREE_LIB -ASMLIST
Executing: "C:\HTSOFT\PIC18\BIN\PICC18.EXE" -E"tu4lite.lde"
"C:\salvo\tut\tu4\main.obj" "C:\salvo\lib\htpicc18\sfp80leb.lib" -Q -MPLAB -18C452
-M"tu4lite.map" -FAKELOCAL -O"tu4lite.cof"
Error[000]   : undefined symbols:
Error[000]   : _OSsigQoutP (C:\salvo\lib\htpicc18\sfp80leb.lib: binsem.obj)
Error[000]   : _OSecbArea (C:\salvo\tut\tu4\main.obj)
Error[000]   : _OStcbArea (C:\salvo\tut\tu4\main.obj)
Error[000]   : _OSeligQP (C:\salvo\lib\htpicc18\sfp80leb.lib: init.obj)
Error[000]   : _OStimerTicks (C:\salvo\lib\htpicc18\sfp80leb.lib: init.obj)
Error[000]   : _OScTcbP (C:\salvo\lib\htpicc18\sfp80leb.lib: event.obj)
Error[000]   : _OSsigQinP (C:\salvo\lib\htpicc18\sfp80leb.lib: binsem.obj)
BUILD FAILED: Tue Jan 06 11:57:19 2004
```

Figure 21: 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:\HTSOFT\PIC18\BIN\PICC18.EXE" -C -E"main.cce" "main.c" -O"main.obj"
-I"\SALVO\TUT\TU4\SYSF" -I"\SALVO\TUT\TU4\SYSF\..\..\tu1"
-I"\SALVO\TUT\TU4\SYSF\..\..\..\inc" -Q -MPLAB -18C452 -Zg9 -O -DSYSF
-DMAKE_WITH_FREE_LIB -ASMLIST
Executing: "C:\HTSOFT\PIC18\BIN\PICC18.EXE" -C -E"mem.cce" "mem.c" -O"mem.obj"
-I"\SALVO\TUT\TU4\SYSF" -I"\SALVO\TUT\TU4\SYSF\..\..\tu1"
-I"\SALVO\TUT\TU4\SYSF\..\..\..\inc" -Q -MPLAB -18C452 -Zg9 -O -DSYSF
-DMAKE_WITH_FREE_LIB -ASMLIST
Executing: "C:\HTSOFT\PIC18\BIN\PICC18.EXE" -E"tu4lite.lde"
"C:\salvo\tut\tu4\main.obj" "C:\salvo\src\mem.obj"
"C:\salvo\lib\htpicc18\sfp80lab.lib" -Q -MPLAB -18C452 -M"tu4lite.map" -FAKELOCAL
-O"tu4lite.cof"
Error[000]   : undefined symbols:
Error[000]   : _OSlostTicks (C:\salvo\lib\htpicc18\sfp80lab.lib: init.obj)
Error[000]   : _OSdelayQP (C:\salvo\lib\htpicc18\sfp80lab.lib: init.obj)
BUILD FAILED: Tue Jan 06 11:59:19 2004
```

Figure 22: Linker Error due to Mismatch between OSLIBRARY_CONFIG (OSE) and Selected library (sfp80lab.lib)

This occurs because Salvo functions[3] 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.

MPLAB DLL-related Build Problems

As of MPLAB v6.3x, the HI-TECH PICC and PICC-18 compiler are integrated into MPLAB via MPLAB DLLs supplied by HI-TECH. If you encounter difficulty, especially while linking an application built with Salvo libraries, ensure that the compiler options required by the library are truly in effect. The simplest way to do this is to examine the command lines in the project's Output (i.e. build results) window. A mismatch between MPLAB and the HI-TECH MPLAB suite DLL's (plug-ins) can result in odd compiler and linker behavior, e.g. the application of incorrect link-time command-line arguments.[4]

Always use the latest HI-TECH MPLAB suite DLL's with the appropriate version of MPLAB. You can examine the module path of each DLL that MPLAB is using via Help → About MPLAB IDE, and selecting the module (e.g. Suite_HITECH18) from the scrollable list. The About MPLAB IDE window will display the path to the module – ensure that this is the module (i.e. DLL) that MPLAB should be using.

PICC

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

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

PICC-18

Example projects for PICC-18 can be found in the `salvo\tut\tu1-6\sysf` directories The MPLAB Include Path for each of these projects is set to `salvo\tut\tu1\syse`, and each project defines the `SYSF` symbol.

Complete projects using Salvo freeware libraries are contained in the MPLAB project file `salvo\tut\tu1-6\sysf\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\sysf\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\sysf\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 PICC-18, it requires the `SYSF` 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 In this case, `OSInit()` in `init.c`.

4 For example, an out-of-date HI-TECH PICC-18 DLL, combined with MPLAB v6.40, results in problems with the selection of the memory model and the associated PICC-18 runtime library.