PUMPKIN™
REAL-TIME SOFTWARE

AN-12
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 v5

Note This Application Note has been superceded by *AN-25 Building a Salvo Application with Microchip's MPLAB-C18 C Compiler and MPLAB IDE v6*.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 ANSI C compiler and MPLAB IDE v5.x 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 v5.50.

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:

Salvo User Manual
Salvo Compiler Reference Manual RM-MCC18

Configuring the Compiler

If you have not already done so, install the Microchip MPLAB-C18 compiler. The install directory is normally `c:\mcc18`. See the MPLAB-C18 documentation for more information. Remember to

add the following line to your `autoexec.bat` file or verify that it is already in place:

```
SET MCC_INCLUDE=c:\mcc18\h
```

Also, verify that `c:\mcc18\bin` and `c:\mcc18` have been added to your `PATH` environment variable.

Launch MPLAB but do not open any projects. Open the Project → Install Language Tool … window and select Language Suite: Microchip and Tool Name: MPLAB-C18. Browse to or type in the full pathname for `mcc18.exe` on your system:

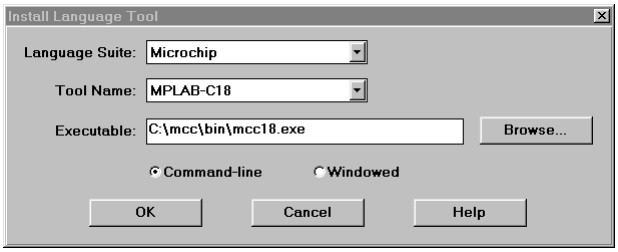

Figure 1: Installing the MPLAB-C18 Language Tool

Ensure that the Command-line radio button is selected. Repeat for the Tool Name: MPASM (`mpasm.exe`) and Tool Name: MPLINK (`mplink.exe`).[1] Click OK to continue.

Creating and Configuring a New Project

Create a new MPLAB project under Project → New Project. Navigate to your working directory (in this case we've chosen `c:\temp`) and create an MPLAB project named `myex1.pjt`:

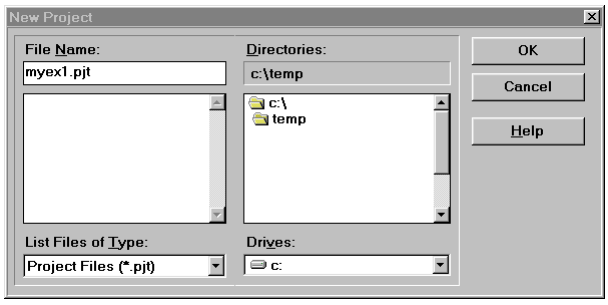

Figure 2: Creating the New Project

Click OK to continue.

In the Edit Project window, select the appropriate Development Mode and Language Tool Suite: Microchip. To aid MPLAB-C18 in finding the project's `main.c` source and `salvocfg.h`

include files, set the Include Path to the directory in which your project is located:

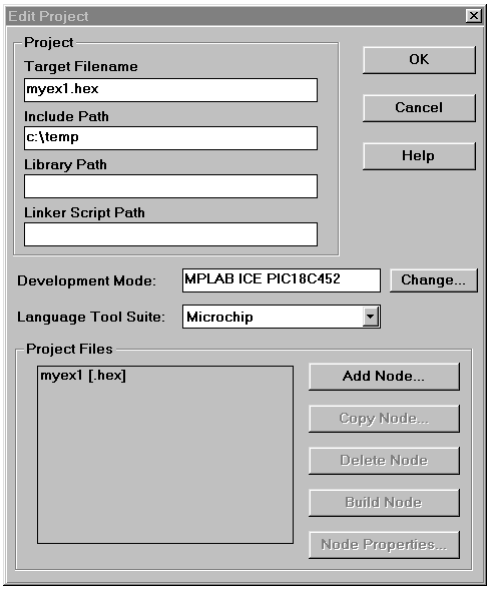

Figure 3: Setting the Include Path, Development Mode and Language Tool Suite

Note Do not add any additional pathnames to the Include Path shown in Figure 3 – this can cause problems when nested header files are present. Where necessary, add them on a node-by-node basis. See *Adding your Source File(s)* and *Troubleshooting: Seemingly Random Build Errors*, below.

Click on `myex1[.hex]` and click on Node Properties. Select Language Tool: MPLINK. Select the following options in the Node Properties window by clicking the corresponding box:

- Map File: On (the filename `myex1.map` will automatically be specified under Data)

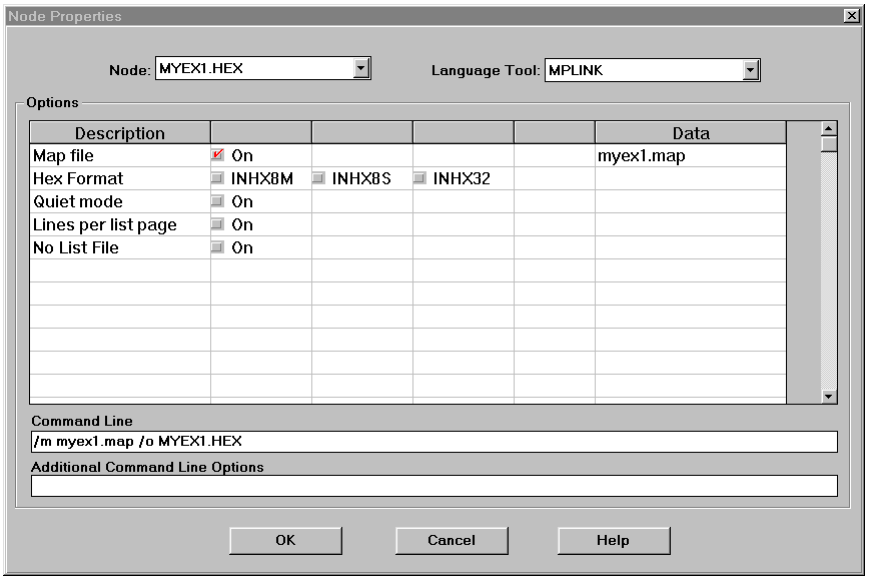

Figure 4: Setting Linker Options

Click OK to continue.

Now that MPLINK has been specified as the linker, you can specify the required Library Path (`c:\mcc\lib`) and Linker Script Path (`c:\mcc\lkr`) in the Edit Project window:

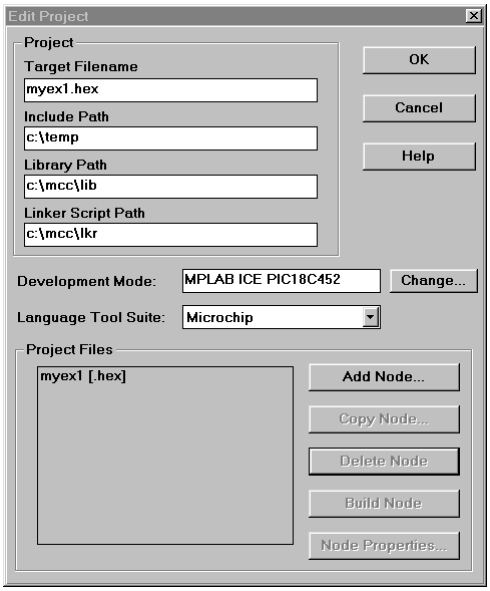

Figure 5: Setting Library Path and Linker Script Path Options

Click OK to continue.

Adding your Source File(s) to the Project

Click on `myex1[.hex]` in the Project Files pane of the Edit Project window and click on Add Node. Choose `main.c` in the `\salvo\ex\ex1` directory and click OK:

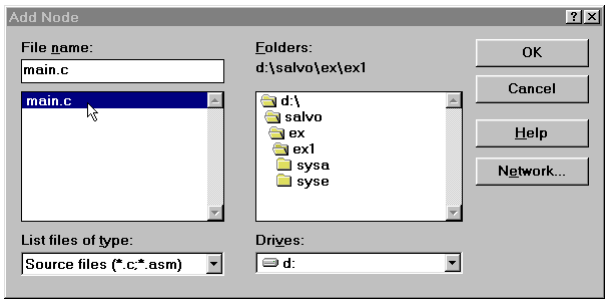

Figure 6: Selecting main.c Source File

Click on `main[.c]` in the Project Files pane of the Edit Project window and click on Node Properties. Under Language Tool: select MPLAB-C18, and select the following options:

- Memory Model: Small[2]
- Enable All Optimizations: On
- Include Path: `\salvo\inc`

Since this project is intended to run on Salvo's `syse` test system, we'll also define the symbol `SYSE` via Define: On SYSE.[3] Where necessary, additional include paths and defined symbols can be added to the Additional Command Line Options field with `-i`*pathname* and `-d`*symbolname*.

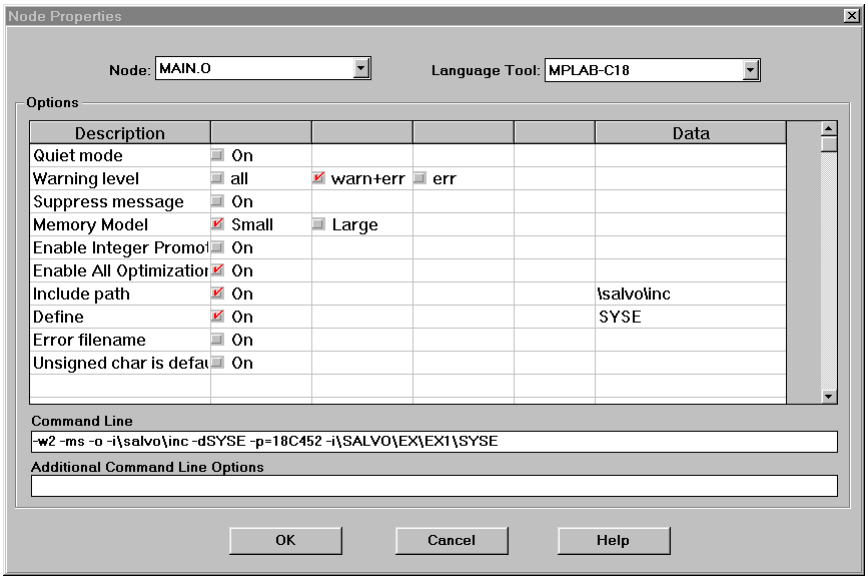

Figure 7: Setting Compiler Options

The `\salvo\inc` Include Path enables MPLAB-C18 to find the Salvo header files. Click OK to continue.

Note This project has just one source file, `main.c`. The easiest way to add additional source files is to select an existing source file (e.g. `main.c`) in the Project Files pane of the Edit Project window and then use the Copy Node button to add the source file(s). This method copies the node properties of the existing source file (i.e. `main.c`) and saves you from having to re-enter them via the Node Properties button. All source files should be added *before* you add any libraries to your project (see below).

Adding a Linker Script to the Project

MPLAB-C18 requires a processor-specific linker script. To add it, click on Add Node in the Project Files pane of the Edit Project window, navigate to the linker script folder (usually `c:\mcc\lkr`) and select the appropriate script:

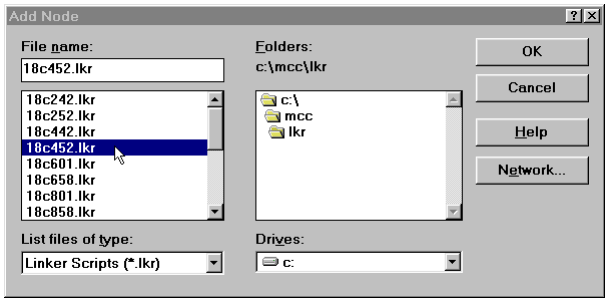

Figure 8: Selecting the Appropriate Linker Script

Click OK to continue. The Edit Project window will now look like this:

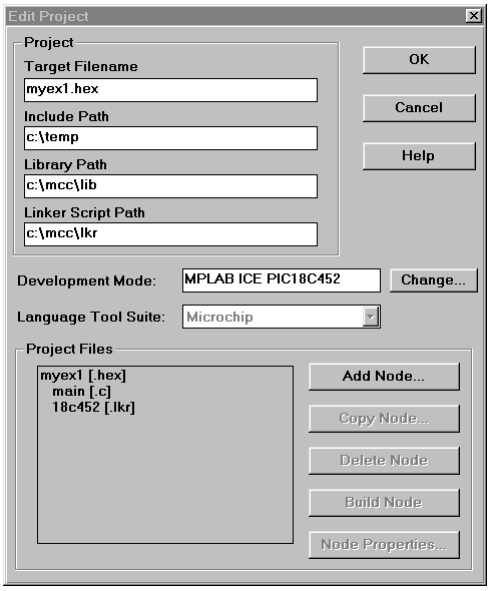

Figure 9: Project Window prior to Adding Salvo Files

Adding Salvo-specific Files to the Project

Now it's time to add the 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 Salvo's mem.c

Salvo *library builds* require Salvo's `mem.c` source file as part of each project. Click on `myex1[.hex]` in the Project Files pane of the Edit Project window and click on Add Node. Choose `mem.c` in the `\salvo\src` directory and click OK:

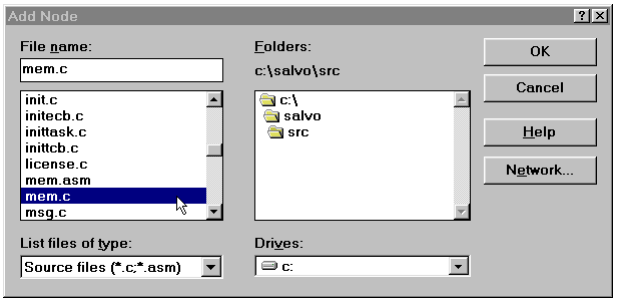

Figure 10: Adding Salvo's mem.c Source File

Adding a Library

For a library build, a freeware library that's appropriate for the PIC18C452 is `sfc18sfa.lib`.[4] In the Project Files pane of the

Edit Project window, click on Add Node, in the Add Node window select List files of type: Libraries (*.lib), navigate to `\salvo\lib\mcc18`, click on `sfc18sfa.lib` and click OK:

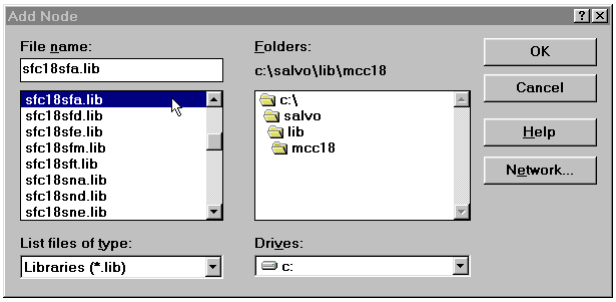

Figure 11: Adding the Library to the Project

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

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_GLOBALS      OSF
#define OSLIBRARY_CONFIG       OSA
#define OSLIBRARY_VARIANT      OSNONE
```

Listing 1: salvocfg.h for a Library Build

and should be located in your project directory (in this case, `c:\temp`). Your Edit Project window should now look like this:

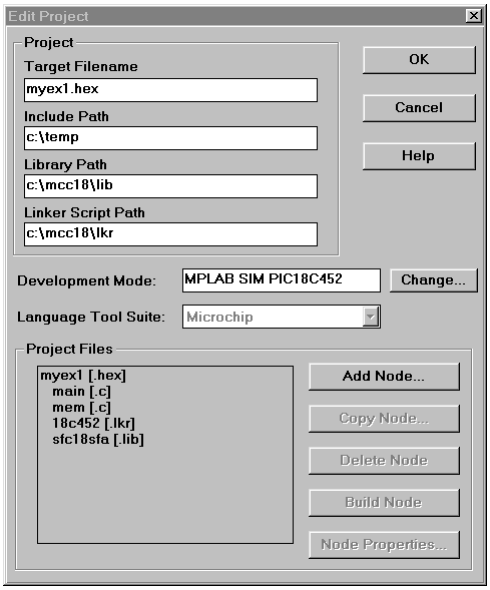

Figure 12: Edit Project Window for a Library Build

Note When using a Salvo library, it must appear at the *end* of your project's list of nodes.

Proceed to *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
init.c                  sched.c
inittask.c              timer.c
```

To add these files to your project, in the Edit Project window select `main.c`, click on Copy Node, and navigate in the Copy Node window to the `\salvo\src` directory. Add each one of the files listed above,[5] and click OK.

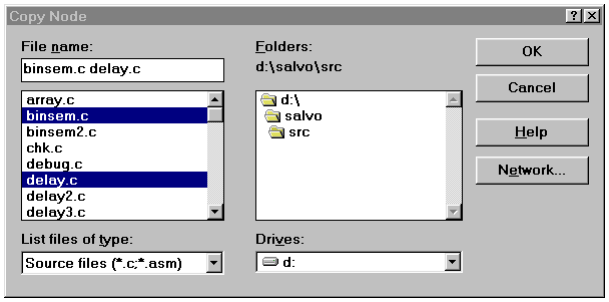

Figure 13: Adding Salvo Source Files to the Project

Your Edit Project window should now look like this:

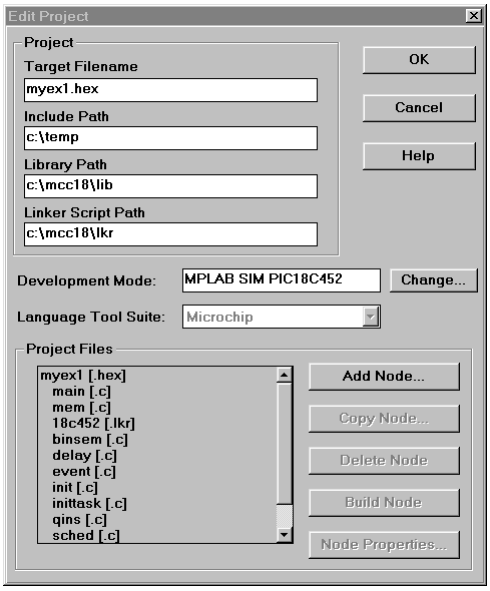

Figure 14: Complete Project Window

Click OK, then select Project → Save Project to save the project.

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, your `salvocfg.h` should contain only:

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

Listing 2: salvocfg.h for a Source Code Build

and should be located in your project directory (in this case, `c:\temp`).

Building the Project

With everything in place, you can now build the project. Here are the results[6] when building `\salvo\ex\ex1\syse\ex1lite.pjt`, which links to a freeware library:

```
Building EX1LITE.HEX...

Compiling MAIN.C:
Command line: "C:\mcc\bin\mcc18.exe -w2 -ms -o
-i\salvo\inc -dSYSE -p=18C452 -i\SALVO\EX\EX1\SYSE
-dMAKE_WITH_FREE_LIB D:\SALVO\EX\EX1\MAIN.C"
MPLAB C18 v1.10 Copyright 2001 Microchip Technology Inc.
Errors:    0
Warnings:  0

Linking:
Command line: "C:\PROGRA~1\MPLAB\MPLINK.EXE /m
ex1lite.map /o EX1LITE.HEX /l C:\MCC\LIB  /k C:\MCC\LKR
D:\SALVO\EX\EX1\MAIN.O C:\MCC\LKR\18C452.LKR
D:\SALVO\LIB\SFC18SFA.LIB "
MPLINK 2.50, Linker
Copyright (c) 2001 Microchip Technology Inc.
Errors    : 0

MP2COD 2.50, COFF to COD File Converter
Copyright (c) 2001 Microchip Technology Inc.
Errors    : 0

MP2HEX 2.50, COFF to HEX File Converter
Copyright (c) 2001 Microchip Technology Inc.
Errors    : 0


Build completed successfully.
```

Listing 3: Build Results with Freeware Library

The results for a project that uses a standard library in place of a freeware library differ only in that the standard library `slc18sfa.lib` is used.

You can examine the project's map file under Window → Map File for further information on ROM and RAM utilization, etc.

Testing the Application

You can test and debug this application with full source code integration in the MPLAB Simulator. After a successful build, select File → Open, navigate to `\salvo\ex\ex1\main.c` in the Open Existing File window, click OK, set a breakpoint on the `PORTB ^= 0x08;` line of `Task3()`, and select Debug → Run → Run F9. Program execution will stop at the breakpoint in `Task3()`. Now zero the stopwatch in the Stopwatch window, select Debug → Run → Run F9, 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).

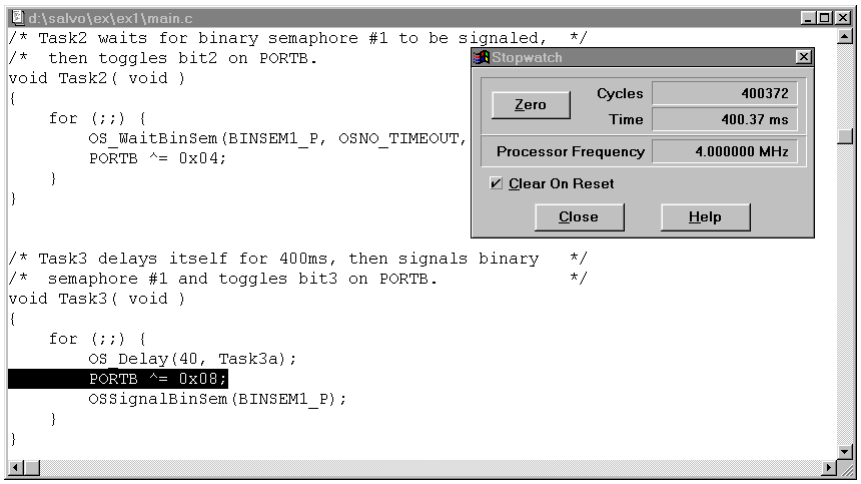

Figure 15: Measuring 400ms of Task Delay in the Simulator via a Breakpoint

Note The extra 0.37 milliseconds shown in the Stopwatch window of Figure 15 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 Debug → Run → Reset F6, Debug → Clear All Points → Yes, and set a breakpoint at the first call to `OSCreateTask()` in `main.c`. Select Debug → Run → Run F9. Execution will stop in `main.c` at the call to `OSCreateTask()`. Now[7] choose Debug → Run → Step F7. The `\salvo\src\inittask.c` file window will open, and you can step through and observe the operation of `OSCreateTask()`.

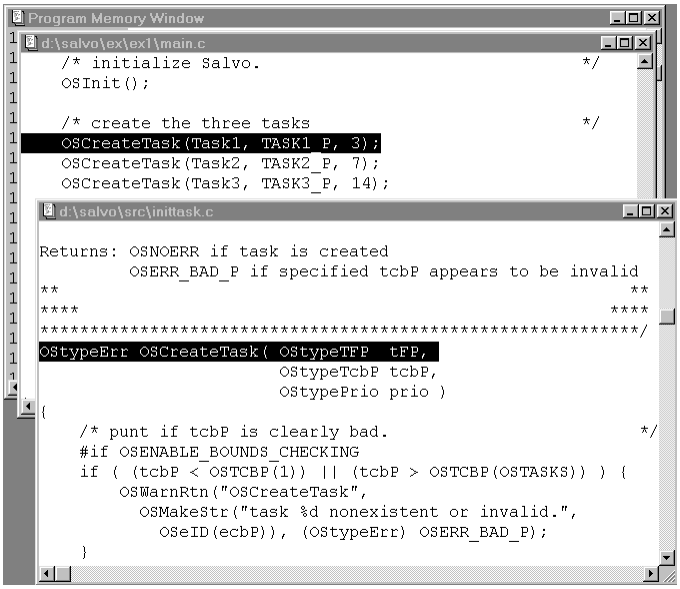

Figure 16: Stepping Through Salvo Source Code

Troubleshooting

"No Build Tool Installed …"

If you receive this error:

```
Building MYEX1.HEX...
Compiling MAIN.C:
No build tool installed for this file.

Build failed.
```

Listing 4: Failure to Build Project

when trying to build your project, it may be due to an improper Language Tool selection in the Node Properties of the file being compiled. Ensure that you have selected Language Tool: MPLAB-C18 for each `*.c` file in your project.

MP2COD Warnings with Library Builds

When building your project using Salvo libraries, you may encounter a warning like this:

```
MP2COD v2.20.00, COFF to COD File Converter

Copyright (c) 2000 Microchip Technology Inc.


Warning - Could not open source file
'c:\build\salvo\src\binsem.c'.

This file will not be present in the list file.

…

Warning - Could not open source file
'c:\build\salvo\src\timer.c'.

This file will not be present in the list file.


Errors    : 0

Warnings  : 10
```

Listing 5: "This file will not be present …" Warnings

These warnings can be avoided by ensuring that you are running the latest version of `MP2COD.EXE`.

No Debug Information Available

If the pop-up window shown in Figure 17 appears after you select Debug → Run → Run, then Debug → Run → Halt an application from within MPLAB, it's probably because you've selected No List File: On in the Node Properties of the target (`[.hex]`) file. Be sure to leave this option unchecked.

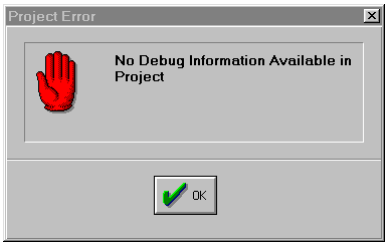

Figure 17: Result of Suppressing List File

Seemingly Random Build Errors

If you encounter either

```
"Unable to spawn process. WinExec returned error code
0."
```

or

```
" MPLAB is unable to find output file "filename". This
may be due to a compile, assemble, or link process
failure."
```

errors this is probably due to the manner in which you've specified your project's include paths. To confirm this, select Window → Project and examine the dependency lists for all of the nodes in your project. A typical node in a Salvo project will have a few (e.g. 1-5) header file dependencies. If you encounter one or more nodes with many more unneeded dependencies, then these build errors are likely to occur.

The solution is to specify *only* your working directory in your project's Include Path (as shown in Figure 3), and manually add any other required include paths by using MPLAB-C18's `-i`*pathname* command-line parameter.

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.pjt`. 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.pjt`. 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.pjt`. These projects also define the `MAKE_WITH_SOURCE` symbol.

---

1 They are normally located in the MPLAB installation directory.

2 Salvo supports small and large memory models for use with MPLAB-C18. All nodes must have the same memory model selected, and the memory model must match that of the Salvo library used (if any).

3 See `\salvo\ex\ex1\main.c` for examples of where `SYSE` and other test-system-specific symbols are used.

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

5 You can Ctrl-select multiple files at once.

[6] The build results are shown in this format to avoid the window clipping that occurs in the normal MPLAB Build Results window.

[7] Ensure that a C source window (in this case, `main.c`) is the foremost window when stepping through C source code. If the Program Memory window is foremost, stepping will occur in assembly language instead.