PUMPKIN™
REAL-TIME SOFTWARE

**AN-14**
**Application Note**

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

# *Building a Salvo Application with IAR's PIC18 C Compiler and Embedded Workbench IDE*

## Introduction

This Application Note explains how to use IAR's (http://www.iar.com/) PIC18 C compiler and Embedded Workbench IDE to create a multitasking Salvo application for Microchip's (http://www.microchip.com/) PIC18 PICmicro® microcontrollers.

We will show you how to build the Salvo application contained in `\salvo\ex\ex1\main.c` for a PIC18C452 using IAR Workbench for PIC18 v2.10A/WIN. 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 the IAR Embedded Workbench for the PIC18.

## Related Documents

The following Salvo documents should be used in conjunction with this manual when building Salvo applications with IAR's PIC18 C compiler:

*Salvo User Manual*
*Salvo Compiler Reference Manual RM-IAR18*

## Creating and Configuring a New Project

Create a new Embedded Workbench project under File → New → Project → OK. Select PIC18 as the Target CPU Family,


*All trademarks mentioned herein are properties of their respective companies.*

navigate to your working directory (in this case we've chosen `c:\temp`) and create a project named `myex1.pew`:

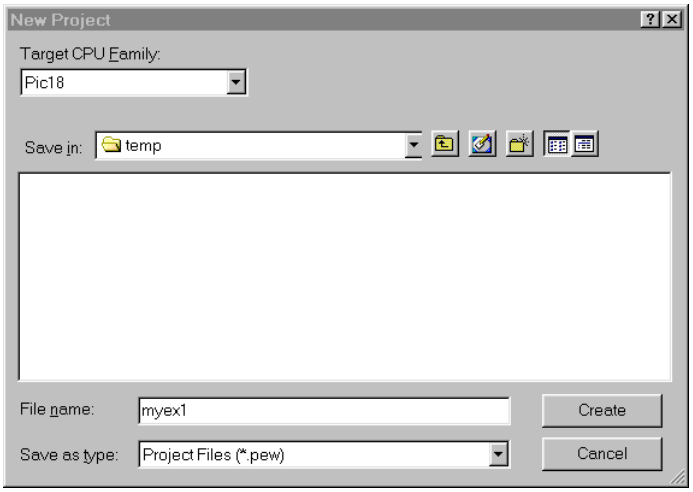


**Figure 1: Creating the New Project**

Click Create to continue. Choose File → Save to save the project.

In order to manage your project effectively, we recommend that you create a set of groups for your project. They are:

Listings
Salvo Configuration File
Salvo Help Files
Salvo Libraries
Salvo Sources
Sources

For each group, choose Project → New Group, add in the Group Name and select OK.

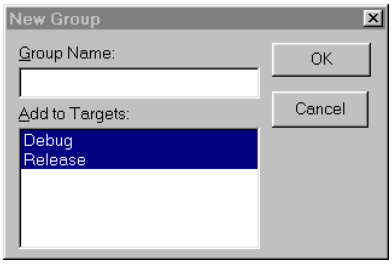


**Figure 2: Creating a Group**

When finished, your project window should look like this:

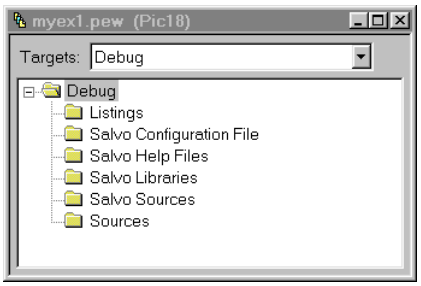


**Figure 3: Project Window with Groups**

Now we'll select the project's options for your particular PIC18 microcontroller. Select Project → Options → General → Target and select Code model → Stack[1] and the appropriate Processor variant:

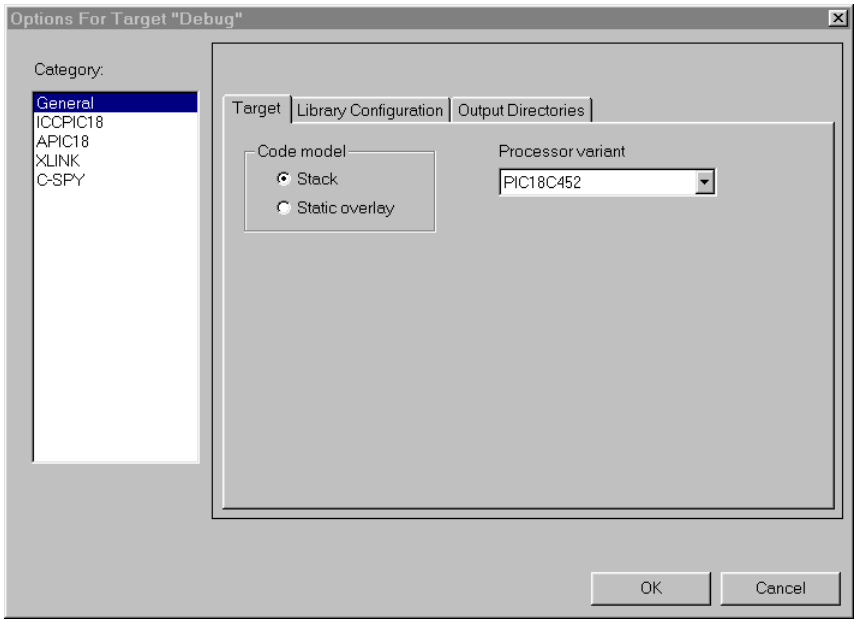


**Figure 4: Setting the Code Model and Processor Variant**

Next, select Project → Options → ICCPIC18 → Preprocessor and add the include paths `$PROJ_DIR$\` and `c:\salvo\inc\` under Include paths.[2] Also, define any symbols you may need for your project under Defined symbols:[3, 4]

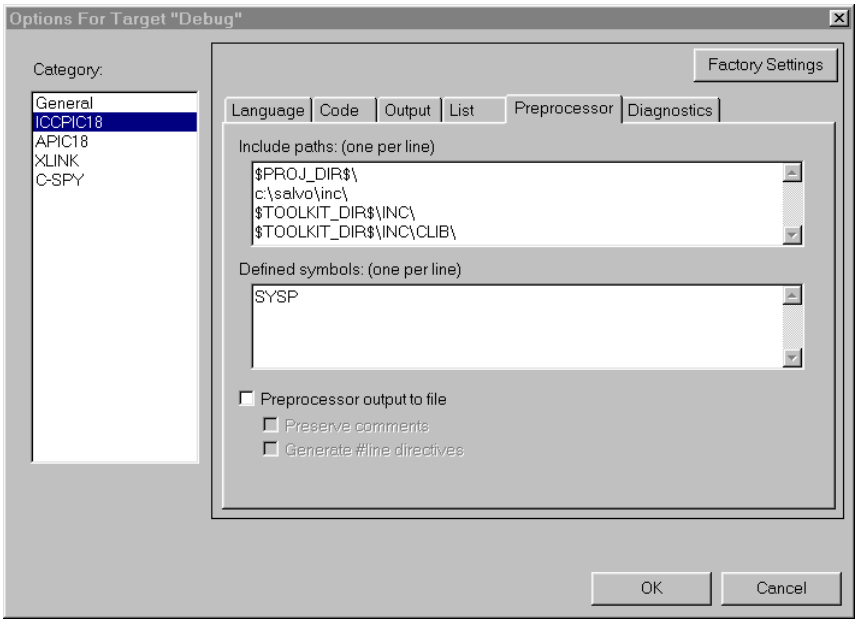


**Figure 5: ICCPIC18 Settings – Project Include Paths**

Next, select Project → Options → XLINK → List → Generate Linker listing. This will create a useful `.map` file with the application's ROM and RAM requirements, etc. Under Project → Options → XLINK → Include, you can use the default XCL file name.

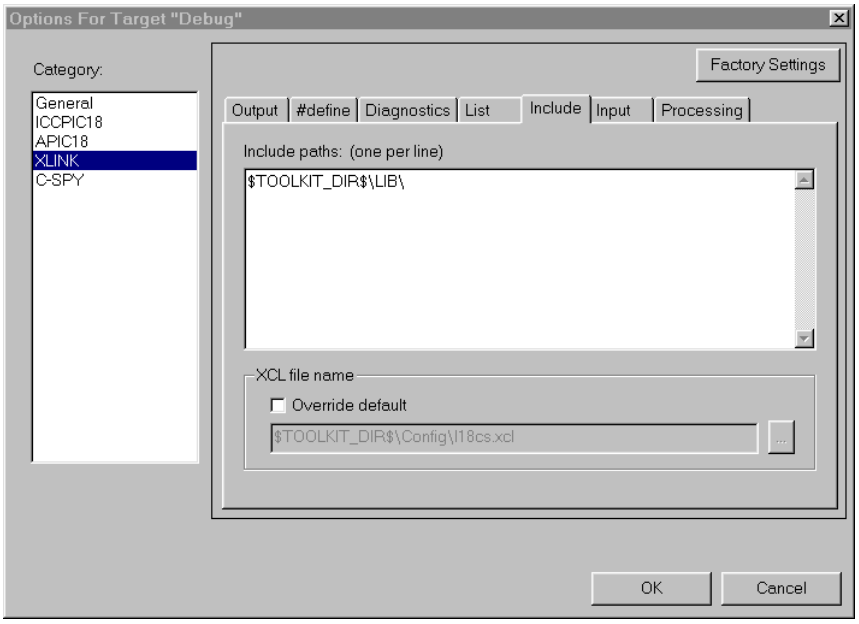


**Figure 6: XLINK Settings – Project XCL File Name**

Lastly, under Project → Options → C-SPY → C-SPY Settings, select the Driver (ICE2000 Emulator or Simulator) and select Device description file → Use device description

description file and select the appropriate description file for your PIC18:

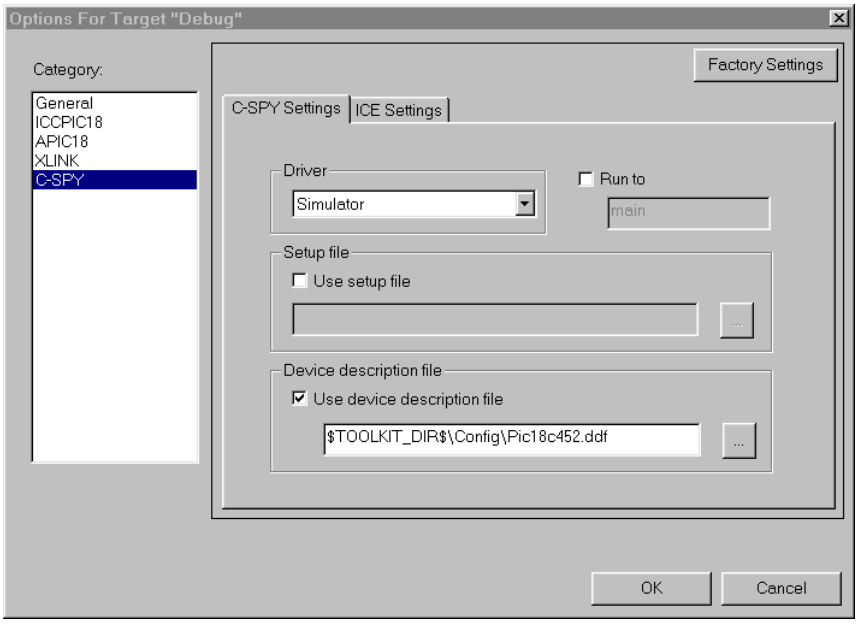


**Figure 7: C-SPY Settings – Project Chip Description File**

Select OK to finish configuring your project.

## Adding your Source File(s) to the Project

Now it's time to add files to your project. Choose Project → Files, C/C++ Source Files (*.c,*.cpp,*.cc) under Files of type, Sources under Add to Group, navigate to your project's directory, select your `main.c` and Add. Your Project Files window should look like this:

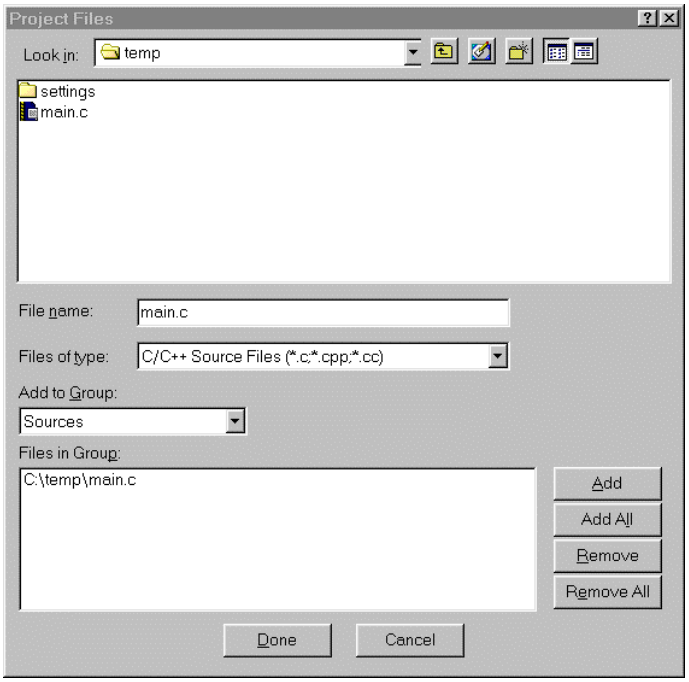


**Figure 8: Project Files Window**

When finished, select Done, and your project window should look like this:

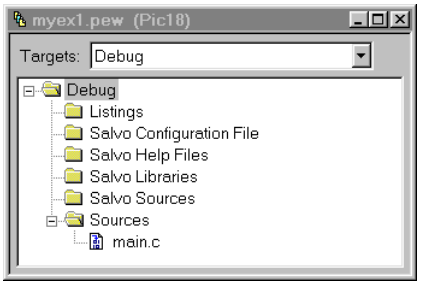


**Figure 9: Project Window with Project-Specific Source Files**

## 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 a Library

For a *library build*, a fully-featured Salvo freeware library for the PIC18 is `sfiar18-slna.r49`.[5] Select Project → Files, Library/Object Files (*.r*) under Files of type, Salvo Libraries

under Add to Group, navigate to the `\salvo\lib\iar18` directory, select `sfiar18-slna.r49` and Add:

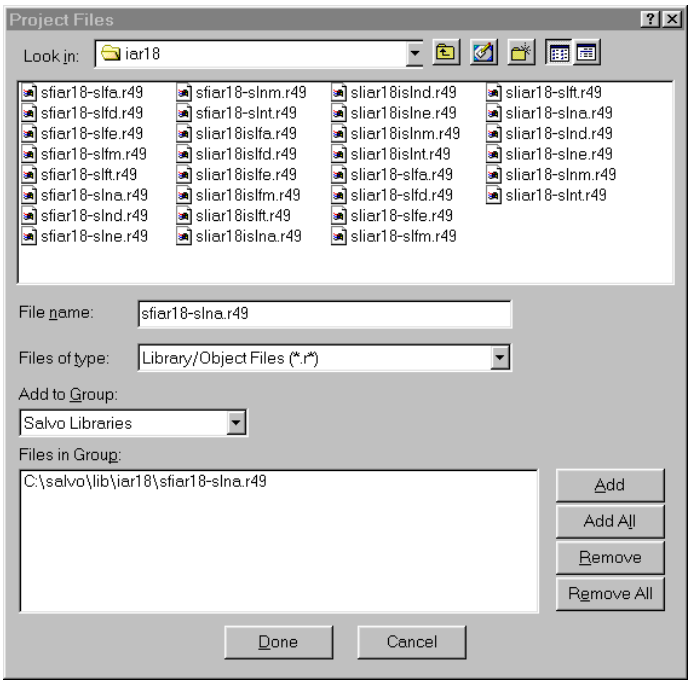


**Figure 10: Adding the Library to the Project**

Select Done when you are finished. You can find more information on Salvo libraries in the *Salvo User Manual* and in the *Salvo Compiler Reference Manual RM-IAR18*.

## Adding Salvo's mem.c

Salvo library builds also require Salvo's `mem.c` source file as part of each project. Choose Project → Files, C/C++ Source Files (*.c,*.cpp,*.cc) under Files of type, select Salvo Sources under Add to Group, navigate to `\salvo\src`, select `mem.c` and Add. Your Project Files window should look like this:

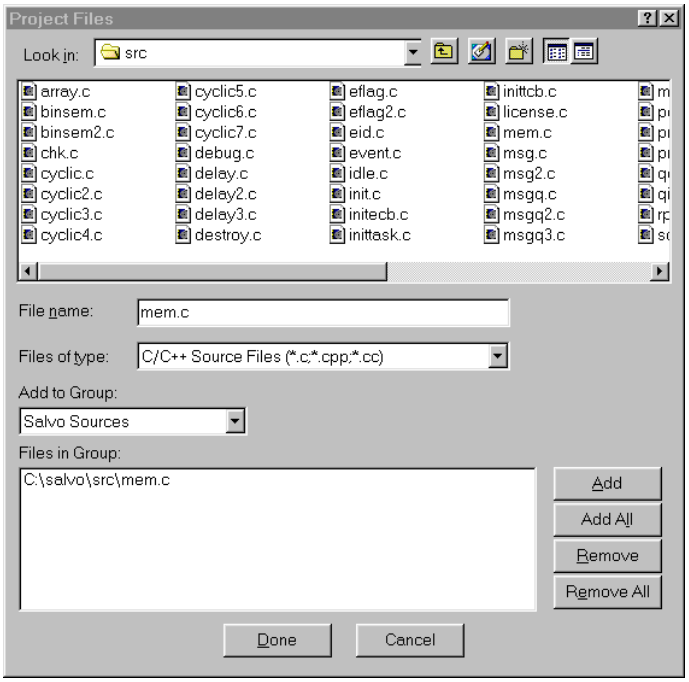


**Figure 11: Project Files Window**

When finished, select Done.

## The salvocfg.h Header File

You will also need a `salvocfg.h` file for this project. To use the library selected in Figure 10, your `salvocfg.h` should contain only:

```
#define OSUSE_LIBRARY          TRUE
#define OSLIBRARY_GLOBALS      OSN
#define OSLIBRARY_CONFIG       OSA
```

**Listing 1: salvocfg.h for a Library Build**

Create this file and save it in your project directory, e.g. `c:\temp\salvocfg.h`.

Select Project → Files, All Files (*.*) under Files of type, Salvo Configuration File under Add to Group, navigate to your project's directory, select `salvocfg.h` and Add:

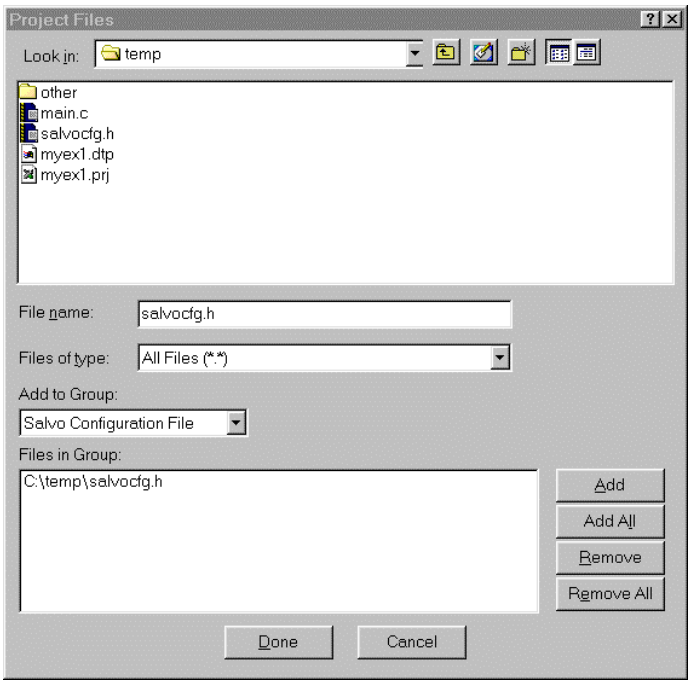


**Figure 12: Adding the Configuration File to the Project**

Your project window should now look like this:

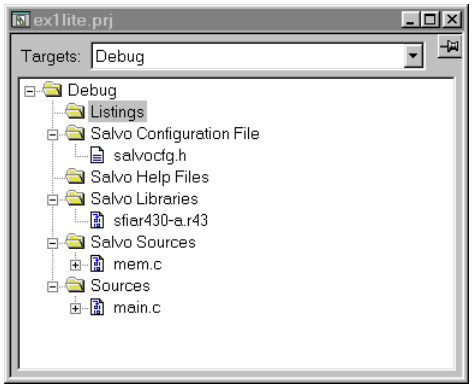


**Figure 13: Project Window for a Library Build**

**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 editing, etc.

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
idle.c                  sched.c
init.c                  timer.c
inittask.c
```

To add these files to your project, select Project → Files, All Files (*.*) under Files of type, Salvo Sources under Add to Group:, navigate to the \salvo\src directory, select[6] the files listed above and Add:

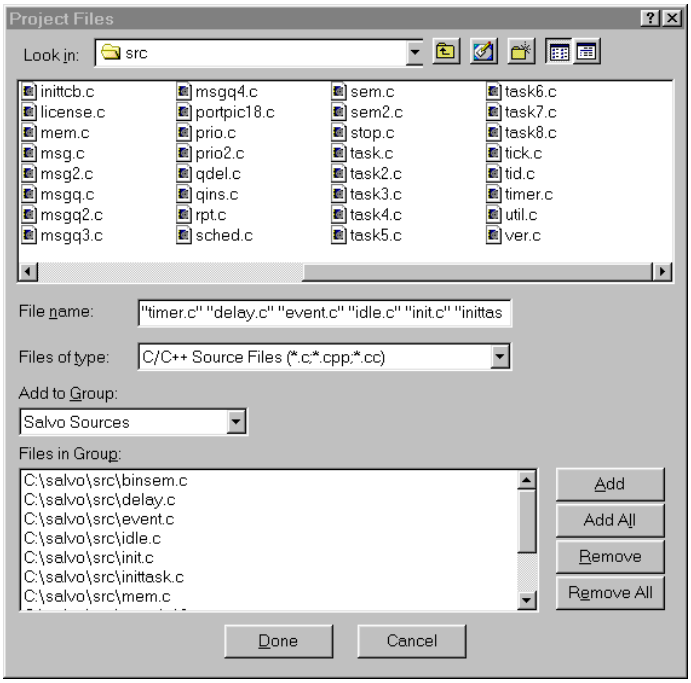


**Figure 14: Adding Salvo Source Files to the Project**

Select Done when finished. Your project window should now look like this:

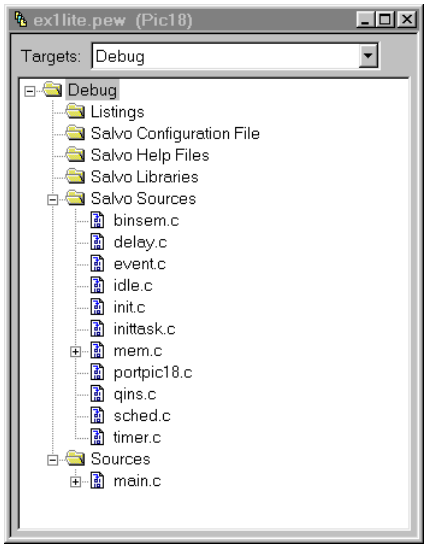


**Figure 15: Project Window for a Source Code Build**

## 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`.

Select Project > Files, All Files (*.*) under Files of type, Salvo Configuration File under Add to Group, navigate to your project's directory, select `salvocfg.h` and Add:

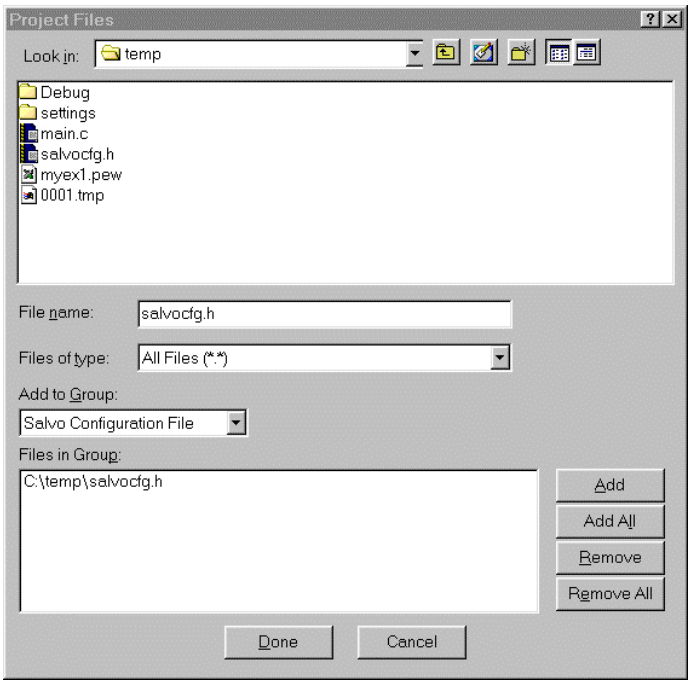


**Figure 16: Adding the Configuration File to the Project**

Your project window should now look like this:

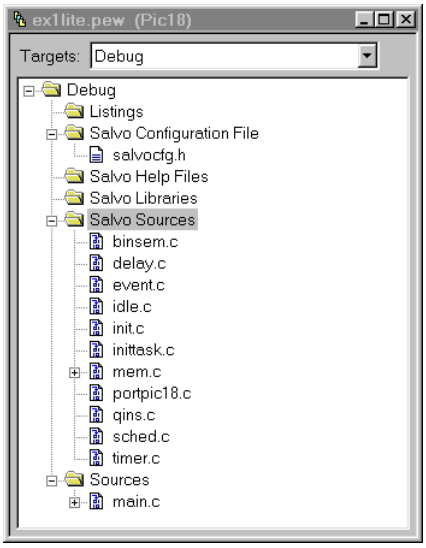


**Figure 17: Edit Project Window for a Library Build**

**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 editing, etc.

## Building the Project

For a successful compile, your project must also include a header file (e.g. `#include <io18c452.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 → Make or Project → Build All. The build results can be seen in the map file located in the project's `Debug\List` subdirectory:[7]

```
################################################################################
#                                                                              #
#      IAR Universal Linker V4.53I/386                                         #
#                                                                              #
#           Link time     =  07/Aug/2002  21:47:38                            #
#           Target CPU    =  PIC18                                            #
#           List file     =  C:\temp\Debug\List\myex1.map                     #
#           Output file 1 =  C:\temp\Debug\Exe\myex1.d49                      #
#                            Format: debug                                     #
#                            UBROF version 9.1.0                               #
#                            Using library modules for C-SPY (-rt)             #
#           Command line  =  C:\temp\Debug\Obj\binsem.r49                      #
#                            C:\temp\Debug\Obj\delay.r49                       #
#                            C:\temp\Debug\Obj\event.r49                       #
#                            C:\temp\Debug\Obj\idle.r49                        #
#                            C:\temp\Debug\Obj\init.r49                        #
#                            C:\temp\Debug\Obj\inittask.r49                    #
#                            C:\temp\Debug\Obj\mem.r49                         #
#                            C:\temp\Debug\Obj\portpic18.r49                   #
#                            C:\temp\Debug\Obj\qins.r49                        #
#                            C:\temp\Debug\Obj\sched.r49                       #
#                            C:\temp\Debug\Obj\timer.r49                       #
#                            C:\temp\Debug\Obj\main.r49                        #
#                            -e_medium_read=_formatted_read                    #
#                            -e_Scanf_1=_Scanf                                 #
#                            -e_small_write=_formatted_write                   #
#                            -e_Printf_1=_Printf -o                            #
#                            C:\temp\Debug\Exe\myex1.d49 -rt -l                 #
#                            C:\temp\Debug\List\myex1.map -xms                  #
#                            -IC:\IAR\EW33\PIC18\LIB\ -f                       #
#                            C:\IAR\EW33\PIC18\Config\l18cs.xcl (-cpic18        #
#                            -D_..X_STACK_SIZE=130 -Z(CODE)INTVEC=0000-00100    #
#                            -Z(CODE)ICODE,RCODE,BANK_ID,BANK_ID_END,BANK_ZD,B #
#                            ANK_ZD_END=4-1FFFFF                               #
#                            -P(CODE)CODE=4-1FFFFF                             #
#                            -P(CODE)BANKN_ID,BANK0_ID,BANK1_ID,BANK2_ID,BANK3 #
#                            _ID,BANK4_ID,BANK5_ID,BANK6_ID,BANK7_ID,BANK8_ID, #
#                            BANK9_ID,BANK10_ID,BANK11_ID,BANK12_ID,BANK13_ID, #
#                            BANK14_ID,BANK15_ID,CONST=4-1FFFFF                #
#                            -Z(CODE)CSTACK=1000000-100001F                    #
#                            -Z(CODE)CHECKSUM=1000100-100010F                  #
#                            -Z(XDATA)EEPROM_I,EEPROM_Z,EEPROM_N=[0-_..X_EEPRO #
#                            M_END]/0100                                       #
#                            -P(CODE)EXTMEM_I,EXTMEM_Z,EXTMEM_N=_..X_EXTMEM_ST #
#                            ART-_..X_EXTMEM_END                               #
#                            -Z(DATA)WRKSEG,BANKN_I,BANKN_Z,BANKN_N=0-07F      #
#                            -Z(DATA)OVERLAY0,BANK0_I,BANK0_Z,BANK0_N=0-0FF    #
#                            -Z(DATA)OVERLAY1,BANK1_I,BANK1_Z,BANK1_N=100-1FF  #
#                            -Z(DATA)OVERLAY2,BANK2_I,BANK2_Z,BANK2_N=200-2FF  #
#                            -Z(DATA)OVERLAY3,BANK3_I,BANK3_Z,BANK3_N=300-3FF  #
#                            -Z(DATA)OVERLAY4,BANK4_I,BANK4_Z,BANK4_N=400-4FF  #
#                            -Z(DATA)OVERLAY5,BANK5_I,BANK5_Z,BANK5_N=500-5FF  #
#                            -Z(DATA)OVERLAY6,BANK6_I,BANK6_Z,BANK6_N=600-6FF  #
#                            -Z(DATA)OVERLAY7,BANK7_I,BANK7_Z,BANK7_N=700-7FF  #
#                            -Z(DATA)OVERLAY8,BANK8_I,BANK8_Z,BANK8_N=800-8FF  #
#                            -Z(DATA)OVERLAY9,BANK9_I,BANK9_Z,BANK9_N=900-9FF  #
#                            -Z(DATA)OVERLAY10,BANK10_I,BANK10_Z,BANK10_N=0A00 #
#                            -0AFF                                             #
#                            -Z(DATA)OVERLAY11,BANK11_I,BANK11_Z,BANK11_N=0B00 #
#                            -0BFF                                             #
#                            -Z(DATA)OVERLAY12,BANK12_I,BANK12_Z,BANK12_N=0C00 #
#                            -0CFF                                             #
#                            -Z(DATA)OVERLAY13,BANK13_I,BANK13_Z,BANK13_N=0D00 #
#                            -0DFF                                             #
#                            -Z(DATA)OVERLAY14,BANK14_I,BANK14_Z,BANK14_N=0E00 #
#                            -0EFF                                             #
#                            -Z(DATA)OVERLAY15,BANK15_I,BANK15_Z,BANK15_N=0F00 #
#                            -0F7F                                             #
#                            -Z(DATA)STACK+_..X_STACK_SIZE=0-0F7F              #
#                            -P(DATA)OVERLAY,BANK_I,BANK_Z,BANK_N=[0-0F7F]/010 #
#                            0                                                 #
#                            clib/cl18s.r49 -Ointel-extended=.hex)             #
#                            -D_..X_EEPROM_END=0 -D_..X_EXTMEM_START=0         #
#                            -D_..X_EXTMEM_END=0                               #
#                                                                              #
#                       Copyright 1987-2002 IAR Systems. All rights reserved. #
################################################################################
```

```
[SNIP]

        ****************************************
              *                                  *
              *      SEGMENTS IN ADDRESS ORDER   *
              *                                  *
              ****************************************


SEGMENT             SPACE   START ADDRESS    END ADDRESS      SIZE  TYPE  ALIGN
=======             =====   =============    ===========      ====  ====  =====
INTVEC              CODE         00000000 - 0000000B            C   com    1
__aseg (ABS)        CODE              00000000                      rel    0
                    CODE              00000000

[SNIP]

                    CODE              00000000
ICODE               CODE         0000000C - 00000099           8E   rel    1
RCODE               CODE         0000009A - 00000171           D8   rel    1
BANK_ID             CODE         00000172 - 00000175            4   rel    0
BANK_ID_END         CODE         00000176 - 00000178            3   rel    0
BANK_ZD             CODE         00000179 - 00000190           18   rel    0
BANK_ZD_END         CODE         00000191 - 00000193            3   rel    0
<CODE> 1            CODE         00000194 - 000011E5         1052   rel    1
WRKSEG              DATA         00000000 - 0000000C            D   rel    0
BANKN_I             DATA              0000000D                      dse    0
BANKN_Z             DATA         0000000D - 0000000E            2   rel    0
STACK               DATA         0000000F - 0000013E          130   dse    0
<BANK_Z,BANK_I> 1   DATA         0000013F - 00000172           34   rel    0
BANKN_A (ABS)       DATA         00000F80 - 00000F80            1   rel    0
                    DATA         00000F81 - 00000F81            1

[SNIP]

                    DATA         00000FFF - 00000FFF            1

              ****************************************
              *                                  *
              *        END OF CROSS REFERENCE    *
              *                                  *
              ****************************************

 4 582 bytes of CODE memory
   371 bytes of DATA memory
    97 bytes of NEARDATA memory

Errors: none
Warnings: none
```

**Listing 3: Source Code Build Results (Abbreviated)**

**Note** The Embedded Workbench for PIC18 projects supplied in the Salvo for PICmicro® MCUs distributions contain additional help files in each project's Salvo Help Files group.

## Testing the Application

You can test and debug this application using the C-SPY debugger and either the simulator or the Flash Emulation Tool. To launch C-SPY, choose Project → Debugger.

You can use all of C-SPY's supported features when debugging and testing Salvo applications. This includes breakpoints, profiling, intelligent watch window, code coverage, etc.

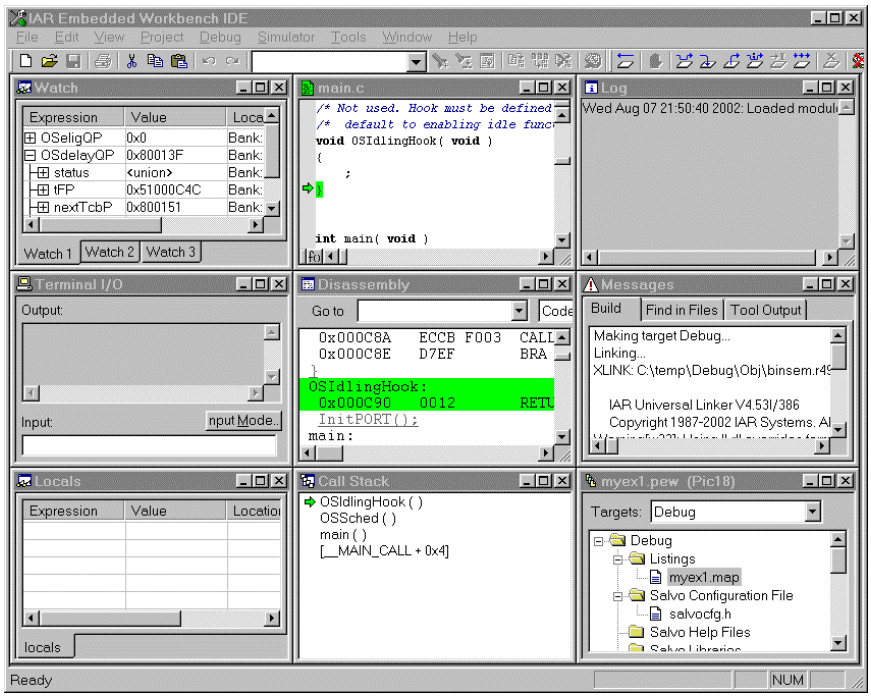


**Figure 18: Testing a Salvo Application in C-SPY**

**Note** C-SPY supports debugging at the source code level. Only applications built from the Salvo source code or a Salvo Pro library enable you to step through Salvo services (e.g. `OSCreateBinSem()`) at the source code level. Regardless of how you build your Salvo application, you can always step through your own C and assembly code in C-SPY.

# Troubleshooting

## PIC18 C-compiler Error: Cannot open include file(s)

If you fail to add `\salvo\inc` to the project's include paths (see Figure 5) the compiler will generate an error like this one:

```
Fatal Error[Pe005]: could not open source file
"salvo.h"
```

**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 5) the compiler will generate an error like this one:

```
Fatal Error[Pe005]: could not open source file
"salvocfg.h"
```

**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`.

## Example Projects

Example projects for IAR's PIC18 C compiler are found in the `\salvo\tut\tu1-6\sysp` directories. The include path for each of these projects includes `salvo\tut\tu1\sysp`, and each project defines the `SYSP` symbol.

Complete projects using Salvo freeware libraries are contained in the project files `\salvo\tut\tu1-6\sysp\tu1-6lite.pew`. These projects also define the `MAKE_WITH_FREE_LIB` symbol.

Complete projects using Salvo standard libraries are contained in the project files `\salvo\tut\tu1-6\sysp\tu1-6le.pew`. These projects also define the `MAKE_WITH_STD_LIB` symbol.

Complete projects using Salvo standard libraries with debugging information are contained in the project files `\salvo\tut\tu1-6\sysp\tu1-6prolib.pew`. These projects also define the `MAKE_WITH_STD_LIB` symbol.

Complete projects using Salvo source code are contained in the project files `\salvo\tut\tu1-6\sysp\tu1-6pro.pew`. These projects also define the `MAKE_WITH_SOURCE` symbol.

---

1 Currently Salvo only supports the stack code model. Choosing the Static overlay model will result in a non-working Salvo application.

2 Relative pathnames are also supported.

3 This Salvo project supports a wide variety of targets and compilers. For use with IAR's PIC18 compiler, it requires the `SYSP` defined symbol, as well as the symbols `MAKE_WITH_FREE_LIB` or `MAKE_WITH_STD_LIB` for library builds. When you write your own projects, you may not require any symbols.

4 We recommend using the Embedded Workbench's argument variables like $PROJ_DIR$ and $TOOLKIT_DIR$ whenever possible.

[5] This Salvo Lite library contains all of Salvo's basic functionality. The corresponding Salvo LE and Pro libraries are sliar18-slna.r49 and sliar18islna.r49, respectively.

[6] You can Ctrl-select multiple files at once.

[7] We recommend that you add the project's map file to your project's Listings group.