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

AN-1

Application Note

Using Salvo Freeware Libraries with the HI-TECH PICC Compiler

Note This Application Note has been superceded by *AN-26 Building a Salvo Application with HI-TECH's PICC and PICC-18 C Compilers and Microchip's MPLAB IDE v6*.

Introduction

Salvo™, The RTOS that runs in tiny places™, is a full-featured cooperative multitasking RTOS for severely memory-limited applications. HI-TECH Software (http://www.htsoft.com/) produces ANSI-compliant C compilers (PICC, PICC Lite and PICC-18) that are certified by Pumpkin for use with Salvo on many of Microchip's (http://www.microchip.com/) PICmicro microcontrollers.

This Application Note explains how to use Salvo freeware libraries with the PICC compilers to create your own multitasking applications on PICmicro devices. Integration with Microchip's MPLAB IDE as well as HI-TECH's HPDPIC IDE is covered. Both HI-TECH PICC and PICC Lite[1] compilers are covered.

Note If you have the full version of Salvo installed on your development machine you may also find *AN-4 Building a Salvo Application with HI-TECH PICC and Microchip MPLAB* useful.

Library Features

Each Salvo freeware library for PICmicro devices is compiled to support all of Salvo's *default* functionality, with allowances made for the architecture or limited resources of the target processor. In general, each library will support several tasks and events, and will include all of Salvo's user services. Events of every type (binary semaphores, messages, etc.) are usually supported. Specifics on

created by Andrew E. Kalman on May 4, 2000 updated on Jul 23, 2003
All trademarks mentioned herein are properties of their respective companies.

each library can be found in the *Libraries* chapter of the *Salvo User Manual*.

Selecting the Appropriate Library

You must use the Salvo freeware library appropriate for your target processor. Libraries are provided for the entire PIC12 (baseline), PIC16 (midrange), PIC17 (high-end) and PIC18 line of PICmicro devices. The freeware libraries are located in `\salvo\lib\htpicc\sfp*.lib`. The nomenclature is similar to that of the HI-TECH PICC libraries and is covered in-depth in the *Libraries* chapter of the *Salvo User Manual*.

Configuring Included Files Properly

In addition to the Salvo header file `salvo.h`, every Salvo application needs its own `salvocfg.h` file. Normally, this is located in the same place as the application's `main.c`. Each source file in a Salvo application must include the main header file via an `#include <salvo.h>` preprocessor directive. This will automatically include the project's `salvocfg.h`, too.

When using freeware libraries with the PICC compiler, only a few configuration options are required in `salvocfg.h`. They are `OSUSE_LIBRARY`, `OSLIBRARY_TYPE`, `OSLIBRARY_CONFIG` and `OSLIBRARY_VARIANT`.

Compiling in Microchip MPLAB

Note In the example below, an application will be created using MPLAB, Microchip's integrated development environment (IDE). Settings for other environments may be similar.

A working MPLAB project for this example is contained in `\salvo\ex\ex1\sysa\ex1.pjt`.

Let's compile the simple multitasking program provided in `\salvo\ex\ex1\main.c` with the freeware libraries using the steps outlined below. This application runs three tasks and uses one event (a binary semaphore) to communicate between two tasks. The system timer, called via an interrupt, is used to provide delay services. The target processor is a Microchip PIC16C77 PICmicro MCU.

Configuring the Compiler

If you have not already done so, install the HI-TECH PICC compiler. The install directory is normally `c:\ht-pic`. See the PICC documentation for more information. Remember to add the following two lines to your `autoexec.bat` file:

```
SET HTC_ERR_FORMAT=Error[ ] file %%f  %%l : %%s
SET HTC_WARN_FORMAT=Warning[ ] file %%f  %%l : %%s
```

These lines are required to support double-clicking on errors in the Build Results window and have MPLAB automatically open the offending source file to the line where the error occurs.

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

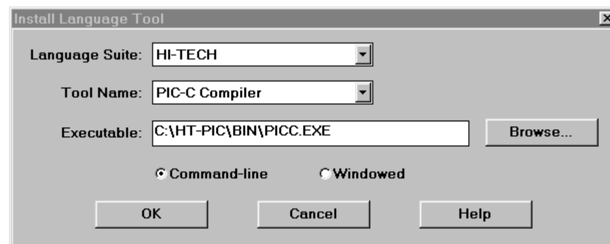

Figure 1: Installing the PICC Language Tool

Ensure that the Command-line radio button is selected. Repeat for the PICC Assembler (`picc.exe`) and PICC Linker (`picc.exe`) under Tool Name. Click OK to continue.

Creating and Configuring a New project

Create a new MPLAB project under Project → New Project. Navigate to the `\salvo\ex\ex1\sysa` directory and create an MPLAB project named `ex1.pjt`:

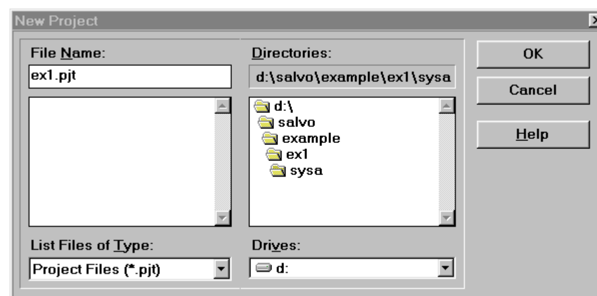

Figure 2: Creating the New Project

Click OK to continue.

In the Edit Project window, select the appropriate Development Mode and Language Tool Suite: HI-TECH. To aid PICC 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:

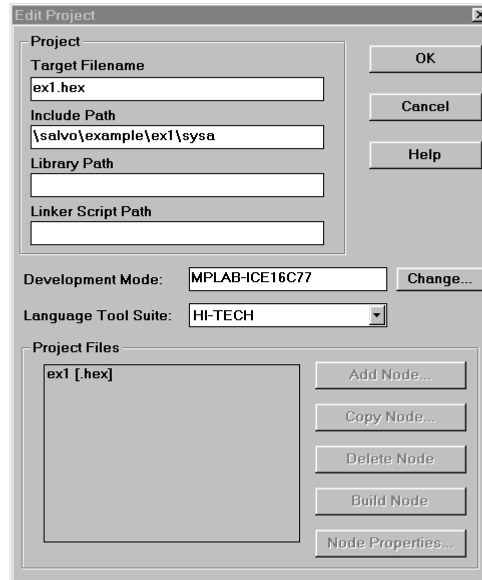

Figure 3: Setting the Include Path, Development Mode and Language Tool Suite

Click on `ex1[.hex]` and click on Node Properties. Select Language Tool: PIC-C Linker. Select the following options in the Node Properties window by clicking the corresponding On box:

- Generate Debug Info
- Map File (specify `ex1.map` under Data)

In the Additional Command Line Options text box, type `-fakelocal`:

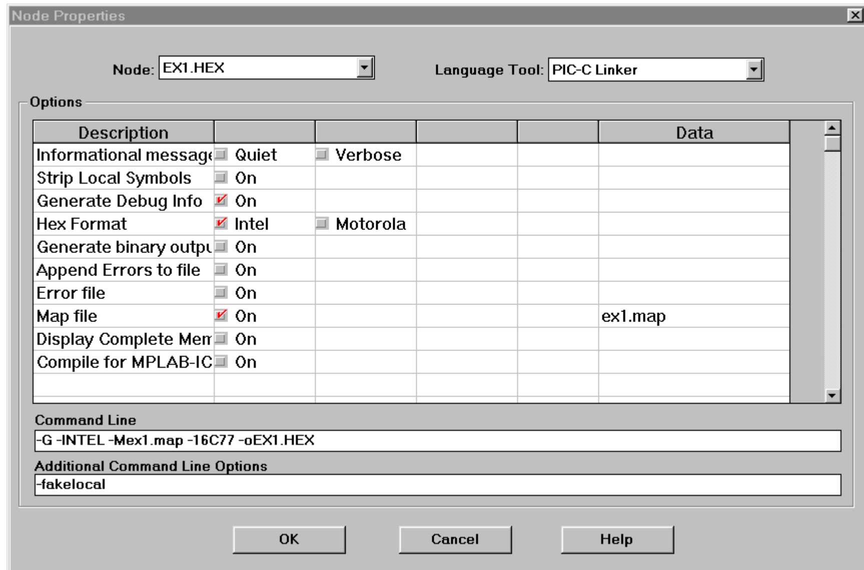

Figure 4: Setting Linker Options

Click OK to continue.

Adding the Source File

Click on `ex1[.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:

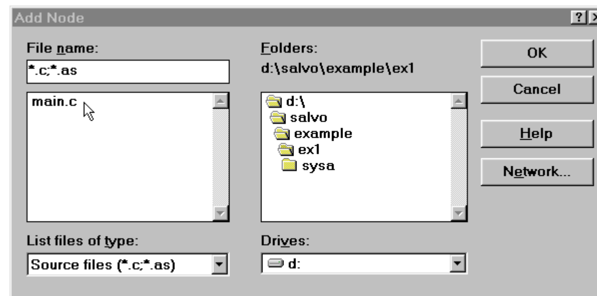

Figure 5: Selecting main.c Source File

Click on `main[.c]` in the Project Files pane of the Edit Project window and click on Node Properties. Select the following options by clicking the On box:

- Generate Debug Info
- Local Optimizations
- Global Optimizations (specify `5` under Data)
- Assembler List file

In the Additional Command Line Options text box, type `-fakelocal -I\salvo\inc`[2] as shown in Figure 6:

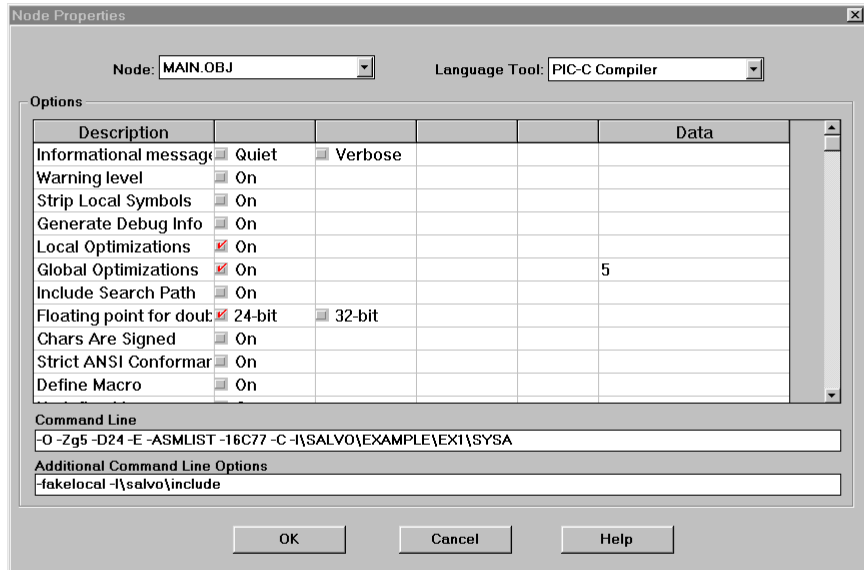

Figure 6: Setting Compiler Options

The additional search path enables the compiler to find the main Salvo header file, `\salvo\inc\salvo.h`. 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 the freeware library to your project (see below).

Adding Salvo's mem.c

Salvo *library builds* require Salvo's `mem.c` source file as part of each project. Click on `ex1[.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:

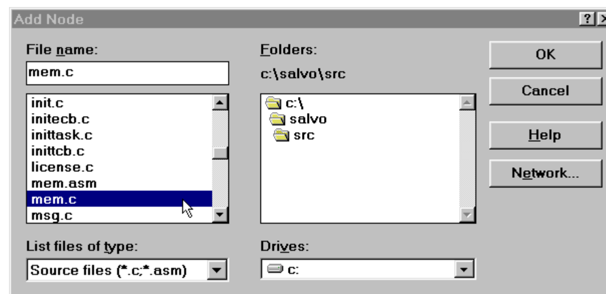

Figure 7: Adding Salvo's mem.c Source File

Adding the Freeware Library

The Salvo services called from `main.c` are contained in the freeware libraries. For the PIC16C77, the correct library (see *Selecting the Appropriate Library*, above) is `sfp42cab.lib`. In the Project Files pane of the Edit Project window, click on `ex1[.hex]`, click on Add Node, in the Add Node window select List files of type: Libraries (*.lib), navigate to `\salvo\lib\htpicc`, click on `sfp42Cab.lib` and click OK.

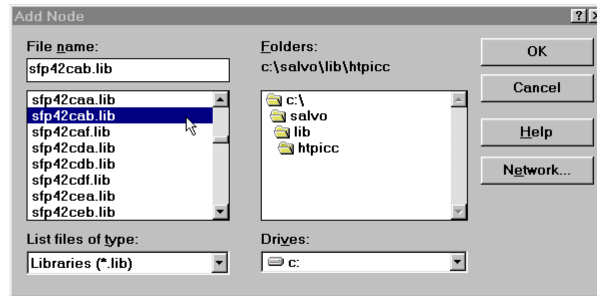

Figure 8: Adding the Freeware Library to the Project

The salvocfg.h Header File

The details on setting the required configuration options in `salvocfg.h` for use with freeware libraries are covered in depth in the *Libraries* chapter in the *Salvo User Manual*. For our example, the required settings are:

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

These configuration options are placed in the `\salvo\ex\ex1\sysa\salvocfg.h` header file. Since this directory is part of the search path (see Include Path in Figure 9), these options will be used to make this project.

The project is now complete, and the Edit Project window looks like this:

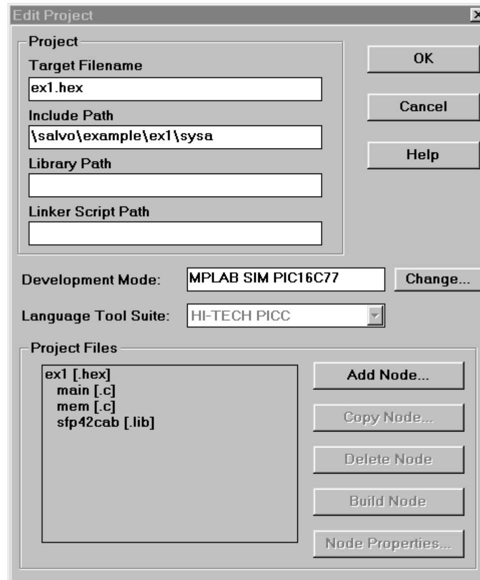

Figure 9: Complete Project Window

Note The Salvo freeware library *must always be at the end* of the list of Project Files in the Edit Project window for a successful compile. Nodes are always added to the end of this list. When adding additional `.c` source file nodes to a project, delete the library node first, and add it back afterwards so that it remains at the end of the list.

Click OK and select Project → Save Project to save the project.

Building the Project

You can now make the project (via Project → Make Project F10) and it should compile and link successfully. The Build Results window will look like this:

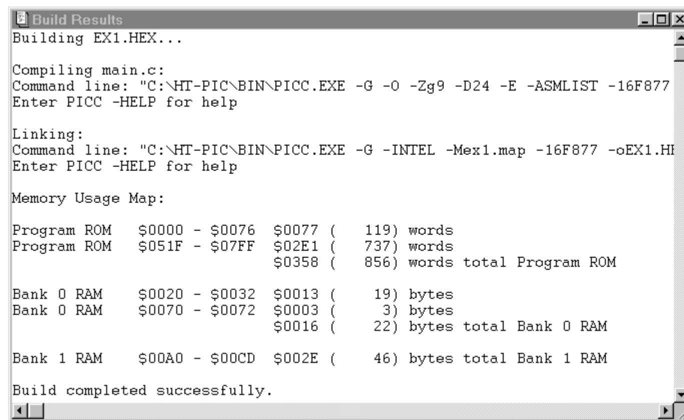

Figure 10: Results of a Successful Build

You can scroll back up through this window to see the results of compiling each source (`.c`) file and linking them together with the freeware library to form an executable Salvo application.

For more detailed information on the memory usage and call tree of this application, open `\salvo\ex\ex1\sysa\ex1.map`.

Compiling in HI-TECH HPDPIC

Note Rather than repeat the above instructions, this section addresses non-obvious issues when configuring an HPDPIC project to build the same application.

A working HPDPIC project for this example is contained in `\salvo\ex\ex1\sysa\ex1.prj`.

Configuring the Compiler

If you have not already done so, install the HI-TECH PICC compiler. The install directory is normally `c:\ht-pic`. See the PICC documentation for more information. Launch HPDPIC.

Verifying the Target Processor

Using Make → Load Project, open `ex1.prj` located in the `\salvo\ex\ex1\sysa` directory. Verify that the correct processor is selected under Options → Select processor … :

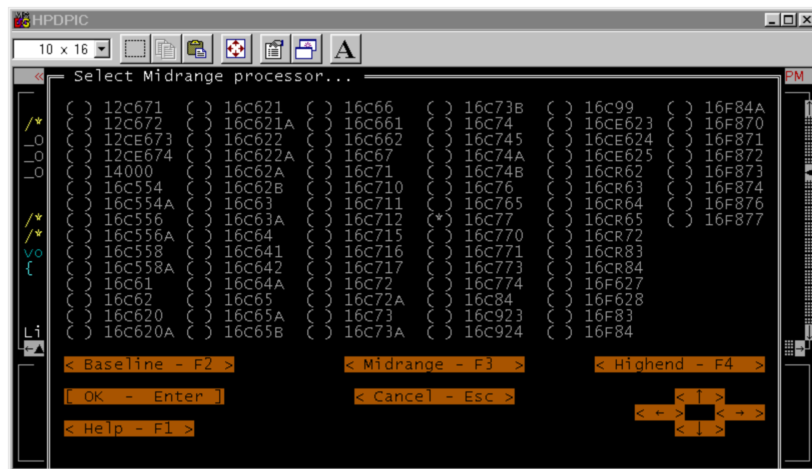

Figure 11: Selecting the Processor

Adding the Freeware Library

The freeware library `sfp42Cab.lib` should be listed first in the Library file list …:

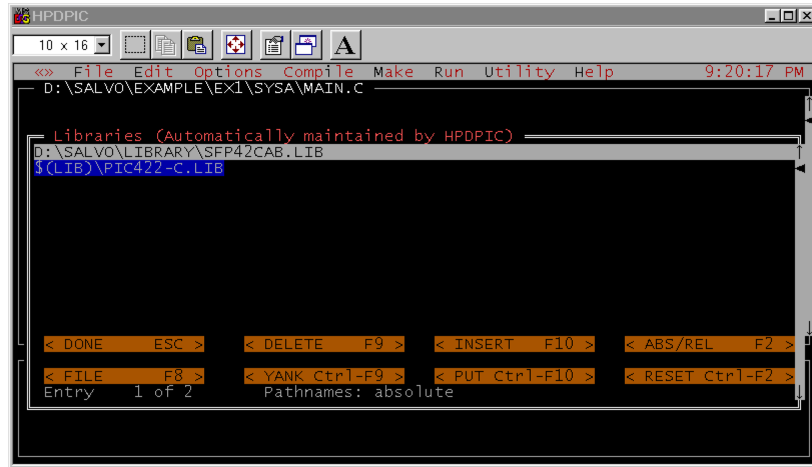

Figure 12: Linking to the Freeware Library

Note The Salvo freeware library's numeric suffix (`42C`, in this example) matches those of the PICC libraries – this is intentional. If the numbers don't match, you may have serious run-time problems.

Setting the Include Paths

For a successful make, the compiler must be able to find the `salvocfg.h` and `salvo.h` files. Since HPDPIC automatically searches the current directory, only a path to `\salvo\inc` is required to find `salvo.h`. Add it via Make → CPP include paths …:

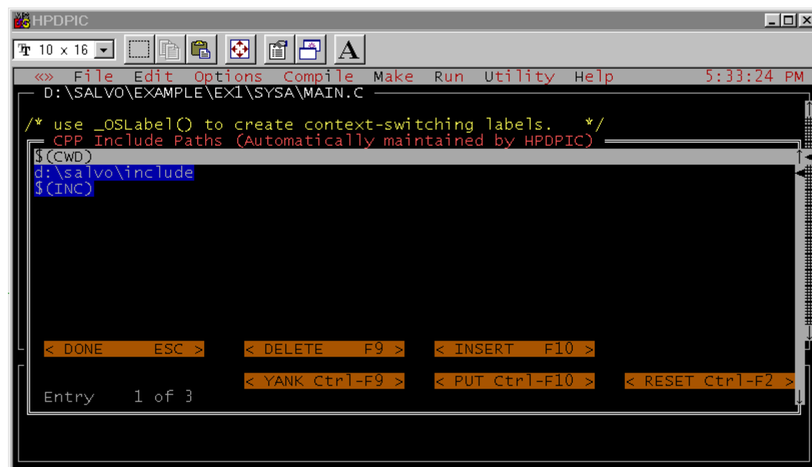

Figure 13: Setting the Include Paths

See *The salvocfg.h Header File* (above) for the proper configuration options settings for `salvocfg.h`.

Setting the Optimizations

Be sure to set the compiler optimizations:

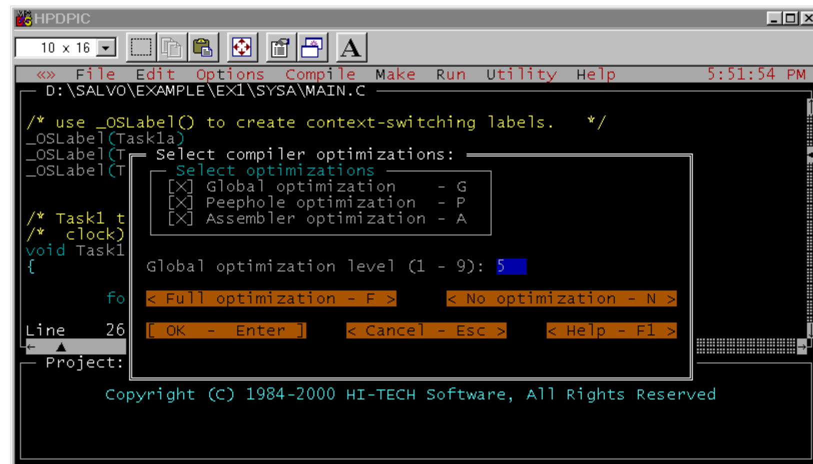

Figure 14: Setting the Optimizations

Building the Project

You can now make the project (via Make F5) and it should compile and link successfully. The size (ROM and RAM) of the final executable should be the same as the one generated in *Building the Project* (above, in *Compiling in Microchip MPLAB*).

After A Successful Build

Once you have built (i.e. compiled and linked) your project successfully, you can test your Salvo PICmicro application by running it in the MPLAB simulator, downloading it to a target system via an emulator (e.g. PICMASTER or MPLAB-ICE), or by programming the PICmicro device (e.g. via a PICSTART PLUS).

Reducing RAM Usage

Each freeware library is compiled with default values for the number of objects (tasks, events, etc.). By setting configuration parameters in `salvocfg.h` and compiling `mem.c` separately it's possible to increase or decrease the RAM allocated to Salvo, and hence the number of objects in your application.

If the number of objects in your application is smaller than what the freeware library is compiled for, you can reduce Salvo's RAM usage. First, add the appropriate configuration options to `salvocfg.h`. Then compile `\salvo\source\mem.c` and link the resulting object module (`mem.o`) to your application. In this case, `mem.o` must be linked *before* the library. These steps are outlined below.

The example application uses three tasks and one event (a binary semaphore). The default settings for the Salvo freeware library `sfp42Cab.lib` support a larger number[3] of tasks and events, and also allocate RAM for event flag and message queue control blocks, which go unused. By adding four configuration options to our `salvocfg.h`, as shown below:

```
#define OSUSE_LIBRARY           TRUE
#define OSLIBRARY_TYPE          OSL
#define OSLIBRARY_CONFIG        OSA
#define OSLIBRARY_VARIANT       OSB
#define OSEVENT_FLAGS           0
#define OSEVENTS                1
#define OSMESSAGE_QUEUES        0
#define OSTASKS                 3
```

and then rebuilding the project with `mem.c` as one of the nodes, we can reduce Salvo's RAM requirements to the minimum required by this application.

Figure 15 shows the MPLAB Edit Project window for the application mentioned above, but compiled for minimal RAM usage. The only difference is the additional `mem.c` node.

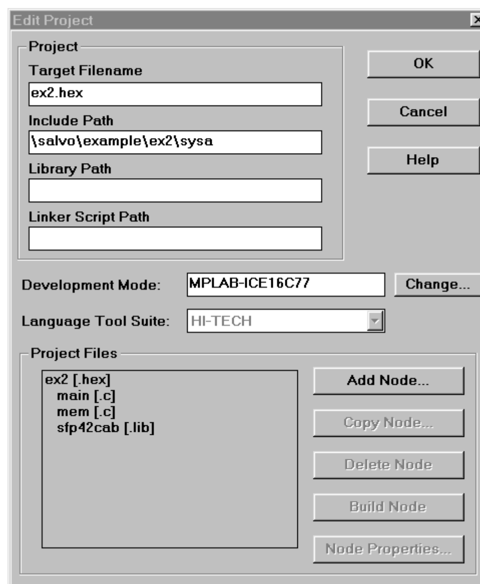

Figure 15: MPLAB Project with Multiple Source Files

Compare the build results shown in Figure 16 to those in Figure 10. Note the substantial reduction (18 bytes) in Bank 1 RAM utilization.

```
Build Results
Building EX2.HEX...

Compiling main.c:
Command line: "C:\HT-PIC\BIN\PICC.EXE -G -O -Zg9 -D24 -E -ASMLIST -16F877
Enter PICC -HELP for help

Compiling mem.c:
Command line: "C:\HT-PIC\BIN\PICC.EXE -G -O -Zg5 -D24 -E -ASMLIST -16F877
Enter PICC -HELP for help

Linking:
Command line: "C:\HT-PIC\BIN\PICC.EXE -G -INTEL -Mex2.map -16F877 -oEX2.HE
Enter PICC -HELP for help

Memory Usage Map:

Program ROM   $0000 - $0076  $0077 (   119) words
Program ROM   $051F - $07FF  $02E1 (   737) words
                             $0358 (   856) words total Program ROM

Bank 0 RAM    $0020 - $0032  $0013 (    19) bytes
Bank 0 RAM    $0070 - $0072  $0003 (     3) bytes
                             $0016 (    22) bytes total Bank 0 RAM

Bank 1 RAM    $00A0 - $00BB  $001C (    28) bytes total Bank 1 RAM

Build completed successfully.
```

Figure 16: Build Results after Minimizing Salvo RAM Usage

Figure 17 shows the Source file list … window for the same project in HPDPIC:

```
HPDPIC
10 x 16
«» File  Edit  Options  Compile  Make  Run  Utility  Help        9:17:57 PM
D:\SALVO\EXAMPLE\EX1\SYSA\MAIN.C

Source Files
D:\SALVO\EXAMPLE\EX1\MAIN.C
D:\SALVO\SOURCE\MEM.C

< DONE     ESC >  < DELETE     F9 >  < INSERT    F10 >  < ABS/REL     F2 >
< FILE      F8 >  < YANK Ctrl-F9 >   < PUT Ctrl-F10 >
Entry    1 of 2      Pathnames: absolute
```

Figure 17: HPDPIC Project with Multiple Source Files

These MPLAB (`.pjt`) and HPDPIC (`.prj`) projects and the `salvocfg.h` header file for reduced RAM utilization are located in `\salvo\ex\ex2\sysa`.

Calling Event Services from Within Interrupts

In the example above, the binary semaphore is signaled from within `Task3()`, i.e. from the background level. Hence the `b` library variant. It is possible to call event services from the

foreground (interrupt) level, or even from both the foreground and the background, by choosing an alternate library variant.

By referring to the *Library* chapter of the *Salvo User Manual* we find that the `f` library variants allow you to call event-reading and –signaling services from the foreground. Similarly, the `a` variants allow you to call the applicable services from anywhere in your code.

The interrupt_level Pragma

If you plan on calling any of Salvo's services from the foreground and background, you should add the following line immediately prior to the interrupt handler:

```
#pragma interrupt_level 0
```
[4]

PICC requires this in order to manage the parameter stack(s) for functions located on multiple call graphs.

Note This pragma has no effect if there aren't any functions located on multiple call graphs. Therefore it's OK to add it to any application compiled with PICC.

Example: Foreground Signaling of One Event Type

If we move the call to `OSSignalBinSem()` from `Task3()` to the interrupt handler `interrupt()` without changing the library variant, you'll find that the application crashes from a stack overflow almost immediately. This is because the default interrupt control[5] in `OSSignalBinSem()` is incompatible with being placed inside an interrupt. To circumvent this, you must change `OSLIBRARY_VARIANT` to `OSF` and link to the freeware library `sfp42Caf.lib` (note the `f` for *foreground* in the variant field) in order to properly support event service calls in the foreground. The program now works properly, albeit with behavior that's different from before, as expected.

Example: Foreground and Background Signaling of One Event Type

If we call `OSSignalBinSem()` from `Task3()` and from within `interrupt()`, the compiler issues an error message:

```
Error[ ] file : function _OSSignalBinSem appears
in multiple call graphs: rooted at intlevel0 and
_main Exit status = 1
```

This occurs because PICC does not pass function parameters on the stack. To resolve this, add the `interrupt_level 0` pragma to your interrupt handler (see above) and use the `a`-variant library after setting `OSLIBRARY_TYPE` to `OSA`. Also, in `Task3()` you will need to call `OSEnterCritical()` immediately before `OSSignalBinSem()` and `OSExitCritical()` immediately after it.[6] The program now works, with apparently the same behavior as when `OSSignalBinSem()` is called only from within `interrupt()`. Can you tell why?

Example: Mixed Signaling of Multiple Event Types

The library variants affect all event services equally – that is, an `f`-variant library expects all applicable event services to be called from the foreground, i.e. from within interrupts. If you wish to call some services from the background, and others from the foreground, you'll have to use the `a`-variant library, as explained above.

A complication arises when you need an `a`-variant library for a particular event type, and you also are using additional event types. In this case, each instance of an applicable event service in use *must be called from the foreground*. If it's not called from the foreground, the compiler issue this error message:

```
Error[ ] file : function _OSSignalBinSem is not
called from specified interrupt level
Exit status = 1
```

However, it need not be called from the background. If you have the "opposite" situation, e.g. you are using an `a`-variant library for one type of event and you need to call an event service for a different event type only from the background, one solution is to place the required foreground call inside an interrupt handler, with a conditional that prevents it from ever happening, e.g.:

```
#pragma interrupt_level 0
void interrupt IntVector( void )
{
  /* real code is here …          */
  …
  /* dummy to satisfy call graph. */
  if ( 0 ) OSSignalBinSem(OSECBP(1));
}
```

This creates a call graph acceptable to PICC and allows a successful compile and execution.

PICC vs. PICC Lite Issues

You may encounter the following error when making a PICC project with HTLPIC (PICC Lite's version of HPDPIC):

```
C:\PICCLITE\LIB\PIC400-L.lib::Can't open
  (error): : No such file or directory
```

This occurs because PICC Lite is supplied without this particular library. The solution is to delete the library from the Library file list … in HPDPIC and re-make the project.

Problems Encountered after Updating a Project File

You may be faced with the following dialog window if you load a project that was created with an earlier or later version of HPDPIC or HTLPIC:

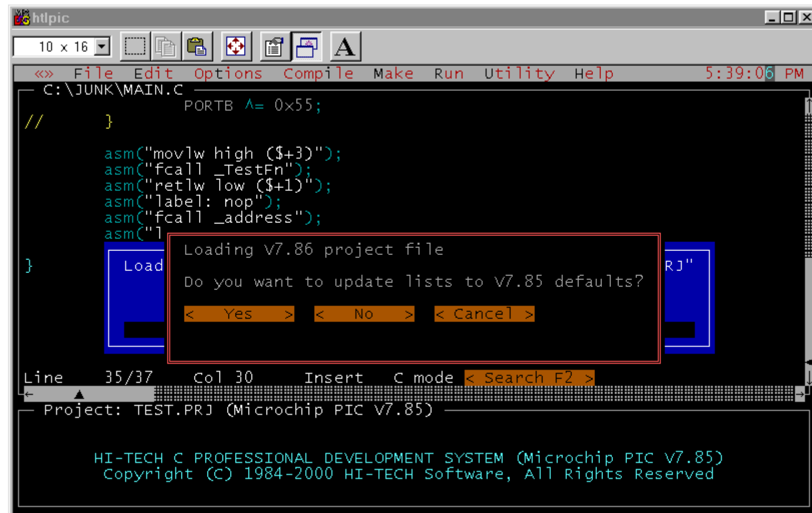

Figure 18: Updating Lists

If you choose Yes the project file will be updated but the settings for the Libraries list and the CPP include paths will be lost. You will need to re-enter them in order to make the project successfully. You can usually choose No without any ill effects.

Making Your Own Application

Creating your own application with the freeware libraries is nearly as easy as building the example projects above. With your own source files calling Salvo services, just remember to:

- Select the correct freeware library based on your processor and which Salvo features you plan to use,
- Set the appropriate configuration options in your project's own `salvocfg.h`,
- Create and use your own MPLAB or HPDPIC project to build your application, and
- Set the project settings and Node Properties (MPLAB), or the Source Files list, the Libraries list and the CPP Include Paths (HPDPIC) properly to access the appropriate Salvo directories.
- If your application doesn't use many tasks or events, consider reducing Salvo's RAM usage by modifying `salvocfg.h` as described above.

---

[1] A freeware version supporting only the PIC16C84, PIC16F84 and PIC16F84A.

[2] Or the appropriate path to the \salvo\inc directory on your system. PICC does not support multiple, semicolon-delimited include paths in the Include Path field of MPLAB's Edit Project window.

[3] See Libraries chapter in Salvo User Manual for specifics.

[4] Salvo always uses level 0.

[5] `OSSignalBinSem()`, like many other user services, disables interrupts on entry and (blindly) re-enables them on exit. The re-enabling of interrupts, if placed inside a PICmicro interrupt routine, causes problems. `OSSignalBinSem()` in the `f-` and `a`-variant libraries control interrupts differently.

[6] See the Reference chapter of the Salvo User Manual for more information on `OSCALL_OSSIGNALEVENT` and other configuration parameters.