PUMPKIN™
REAL-TIME SOFTWARE

AN-4
Application Note

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

Building a Salvo Application with HI-TECH PICC and Microchip MPLAB v5

Note This Application Note has been superceded by *AN-26 Building a Salvo Application with HI-TECH's PICC and PICC-18 C Compilers and Microchip's MPLAB IDE v6*.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

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

In this Application Note we will build the tutorial program located in `\salvo\tut\tu6\sysa` for a PIC16C77. We'll do this in two ways – using the Salvo source files, and using the Salvo standard libraries.

Note This Application Note assumes that you have the full version of Salvo installed on your development machine. If you are working with the freeware libraries and do not have the Salvo source code, please refer to *AN-1 Using Salvo Freeware Libraries with the HI-TECH PICC Compiler*.

created by Andrew E. Kalman on May 13, 2000 updated on Jul 23, 2003

Building with Salvo Source Files

The Application

The Salvo application in `\salvo\tut\tu6\main.c` has two tasks that pass information to a third via a message. The `\salvo\tut\tu6\sysa\salvocfg.h` configuration file contains:

```
#define OSBYTES_OF_DELAYS      1
#define OSCOMPILER             OSHT_PICC
#define OSENABLE_MESSAGES      TRUE
#define OSEVENTS               1
#define OSLOC_ALL              bank1
#define OSTARGET               OSPIC16
#define OSTASKS                3
```

Therefore this application will compile with support for messages and 8-bit task delays, one event control block (for the message) and three task control blocks. The HI-TECH PICC compiler is specified, along with the PIC16 (midrange) family of Microchip PICmicro devices.

More information on this particular application can be found in *Tutorial* chapter of the *Salvo User Manual*.

Creating the Project

Note The project described below is contained in the MPLAB project file `\salvo\tut\tu6\sysa\tu6.pjt`. You may use it to verify the actions listed below. Alternately, you can create your own project under a different name.

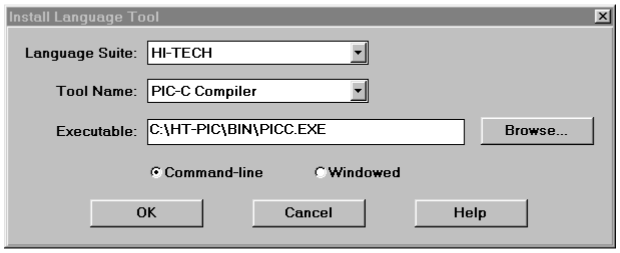
1. If you have not already done so, install the HI-TECH PICC compiler. The install directory is normally `c:\ht-pic`. See the PICC documentation for more information. Remember to add the following two lines to your `autoexec.bat` file:

```
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.

2. 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:


Install Language Tool

Language Suite: HI-TECH

Tool Name: PIC-C Compiler

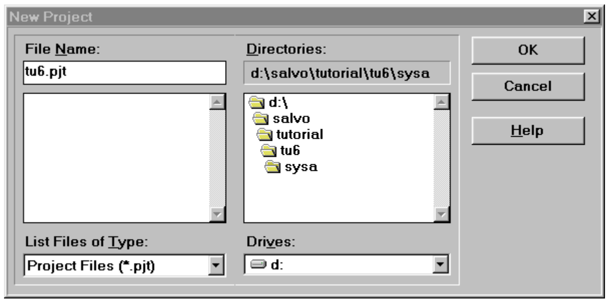
Executable: C:\HT-PIC\BIN\PICC.EXE

(● Command-line) ○ Windowed

Figure 1: Installing the PICC Language Tool

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

3. Under Project → New Project create a new project called `main.pjt` in the `\salvo\tut\tu6\sysa` directory:


New Project

File Name: tu6.pjt

Directories: d:\salvo\tutorial\tu6\sysa

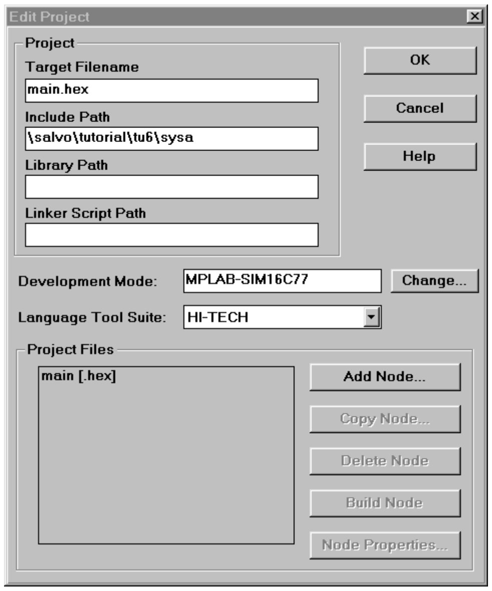
d:\
salvo
tutorial
tu6
sysa

List Files of Type: Project Files (*.pjt)

Drives: d:

Figure 2: Creating the New Project

Click OK to continue. In the Edit Project window, select Development Mode: MPLAB-SIM16C77[1] to configure the project for the PIC16C77 PICmicro device and Language Tool Suite: HI-TECH to use the PICC compiler. Set the Include Path to `\salvo\tut\tu6\sysa`:



Edit Project

Project

Target Filename: main.hex

Include Path: \salvo\tutorial\tu6\sysa

Library Path:

Linker Script Path:

Development Mode: MPLAB-SIM16C77

Language Tool Suite: HI-TECH

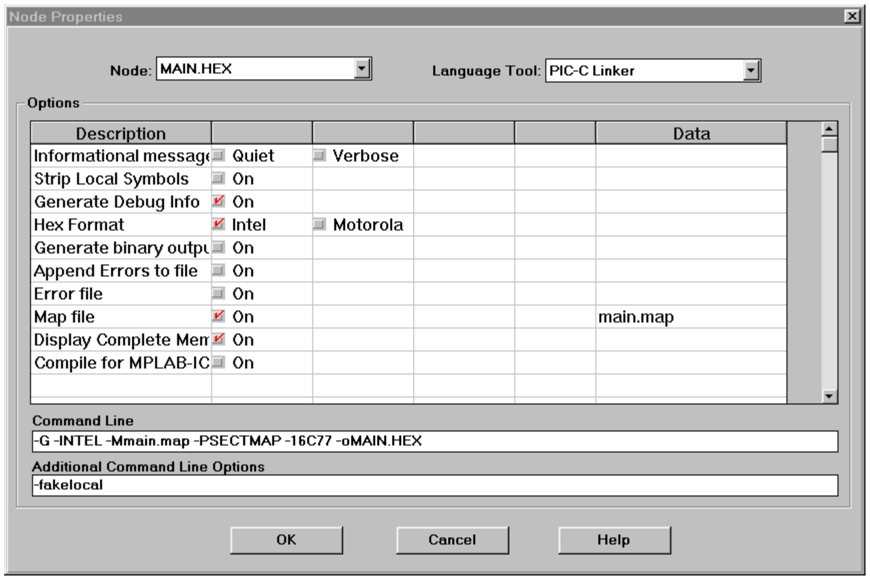
Project Files: main [.hex]

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

4. In the Edit Project window select `main [.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 `main.map` under Data)
- Display Complete Memory Usage

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



Node Properties

Node: MAIN.HEX   Language Tool: PIC-C Linker

Options

Description					Data
Informational message	☐ Quiet	☐ Verbose			
Strip Local Symbols	☐ On				
Generate Debug Info	✓ On				
Hex Format	✓ Intel	☐ Motorola			
Generate binary output	☐ On				
Append Errors to file	☐ On				
Error file	☐ On				
Map file	✓ On				main.map
Display Complete Mem	✓ On				
Compile for MPLAB-IC	☐ On				

Command Line

```
-G -INTEL -Mmain.map -PSECTMAP -16C77 -oMAIN.HEX
```

Additional Command Line Options

```
-fakelocal
```

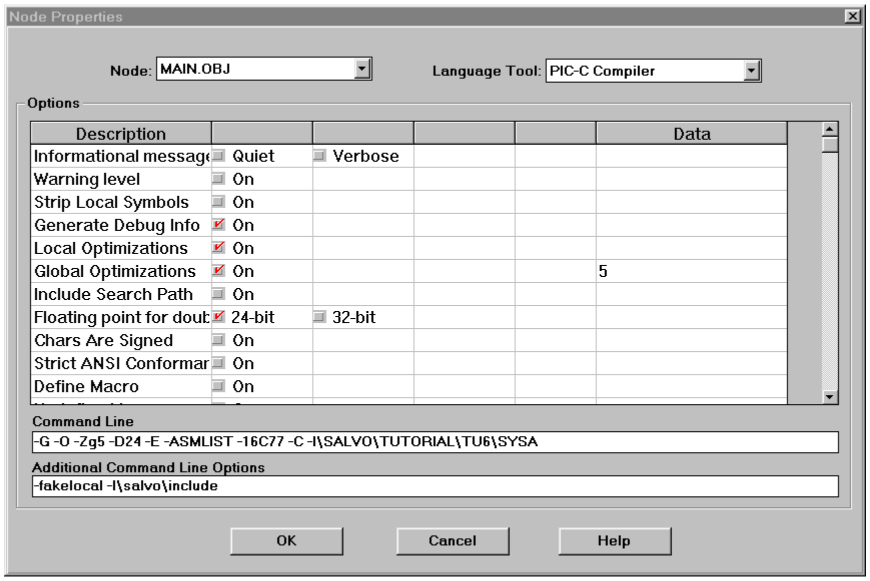
Figure 4: Setting Linker Options

Click OK to continue.

5. Click on `main[.hex]` in the Edit Project window and click on Add Node. Choose the `main.c` file in the `\salvo\tut\tu6` directory and click OK. Click on `main[.c]` in 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]



Description					Data
Informational message	☐ Quiet	☐ Verbose			
Warning level	☐ On				
Strip Local Symbols	☐ On				
Generate Debug Info	✓ On				
Local Optimizations	✓ On				
Global Optimizations	✓ On				5
Include Search Path	☐ On				
Floating point for doub	✓ 24-bit	☐ 32-bit			
Chars Are Signed	☐ On				
Strict ANSI Conformar	☐ On				
Define Macro	☐ On				

Command Line
`-G -O -Zg5 -D24 -E -ASMLIST -16C77 -C -I\SALVO\TUTORIAL\TU6\SYSA`

Additional Command Line Options
`-fakelocal -I\salvo\include`

Figure 5: Setting Compiler Options

Click OK to continue.

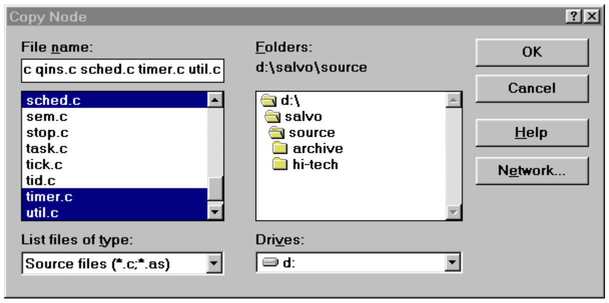
6. The application in `\salvo\tut\tu6\main.c` contains calls to the following Salvo user services:

```
OS_Delay()                    OSEnableInts()
OS_WaitMsg()                  OSInit()
OS_Yield()                    OSSignalMsg()
OSCreateMsg()                 OSSched()
OSCreateTask()                OSTimer()
```

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:

```
delay.c                     mem.c
event.c                     msg.c
idle.c                      qins.c
init.c                      sched.c
initecb.c                   timer.c
inittask.c                  util.c
inittcb.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\source` directory. Add each one of the files listed above,[3] and click OK.


Copy Node

File name: c qins.c sched.c timer.c util.c

sched.c, sem.c, stop.c, task.c, tick.c, tid.c, timer.c, util.c

Folders: d:\salvo\source

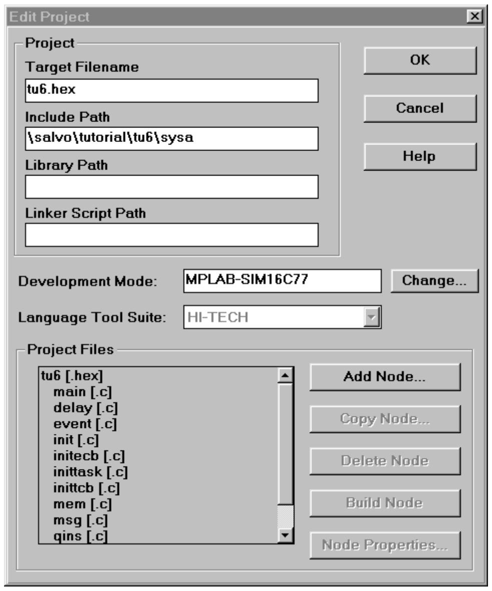
d:\, salvo, source, archive, hi-tech

List files of type: Source files (*.c;*.as)

Drives: d:

Figure 6: Adding Salvo Source Files to Project

Your Edit Project window should now look like this:


Edit Project

Project

Target Filename: tu6.hex

Include Path: \salvo\tutorial\tu6\sysa

Library Path:

Linker Script Path:

Development Mode: MPLAB-SIM16C77

Language Tool Suite: HI-TECH

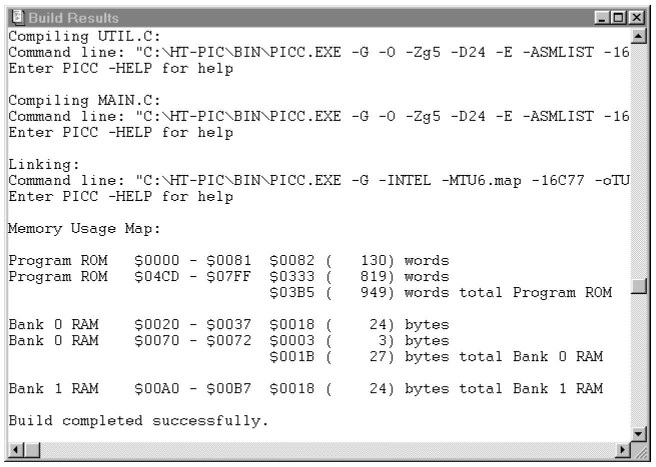
Project Files

tu6 [.hex]
main [.c]
delay [.c]
event [.c]
init [.c]
initecb [.c]
inittask [.c]
inittcb [.c]
mem [.c]
msg [.c]
qins [.c]

Figure 7: Complete Project Window

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

7. Now you can build the project via Project → Make Project F10. The results are shown the Build Results window:

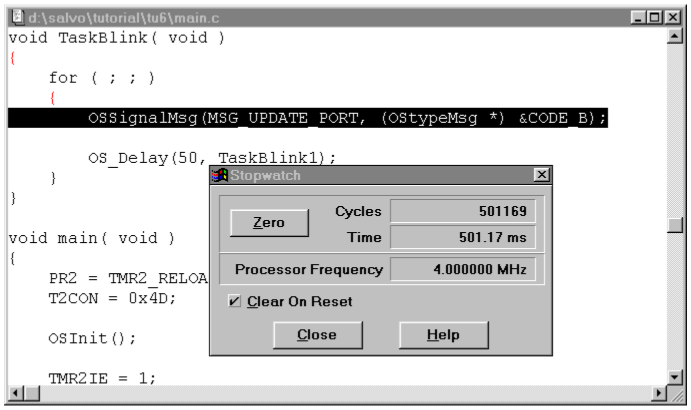


```
Compiling UTIL.C:
Command line: "C:\HT-PIC\BIN\PICC.EXE -G -O -Zg5 -D24 -E -ASMLIST -16
Enter PICC -HELP for help

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

Linking:
Command line: "C:\HT-PIC\BIN\PICC.EXE -G -INTEL -MTU6.map -16C77 -oTU
Enter PICC -HELP for help

Memory Usage Map:

Program ROM   $0000 - $0081  $0082 (   130) words
Program ROM   $04CD - $07FF  $0333 (   819) words
                             $03B5 (   949) words total Program ROM

Bank 0 RAM    $0020 - $0037  $0018 (    24) bytes
Bank 0 RAM    $0070 - $0072  $0003 (     3) bytes
                             $001B (    27) bytes total Bank 0 RAM

Bank 1 RAM    $00A0 - $00B7  $0018 (    24) bytes total Bank 1 RAM

Build completed successfully.
```

Figure 8: Results of a Successful Build

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\tut\tu6\main.c` in the Open Existing File window, click OK, set a breakpoint on the `OSSignalMsg()` line of `TaskBlink()`, and select Debug → Run → Run F9. Program execution will stop in `TaskBlink()`. 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 half a second (50 times 10ms, the `TMR2`-driven system tick rate in this example).



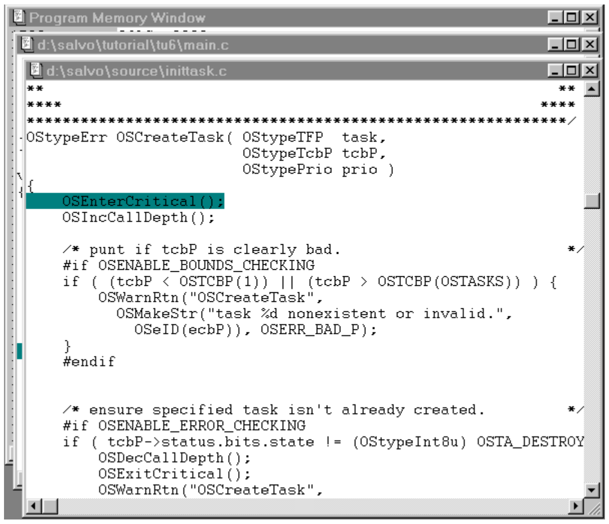
```
void TaskBlink( void )
{
    for ( ; ; )
    {
        OSSignalMsg(MSG_UPDATE_PORT, (OStypeMsg *) &CODE_B);

        OS_Delay(50, TaskBlink1);
    }
}

void main( void )
{
    PR2 = TMR2_RELOA
    T2CON = 0x4D;

    OSInit();

    TMR2IE = 1;
```

Cycles 501169
Time 501.17 ms
Processor Frequency 4.000000 MHz
[✓] Clear On Reset

Figure 9: Measuring 500ms of Task Delay in the Simulator via a Breakpoint

Note The extra 1.17 milliseconds shown in the Stopwatch window of Figure 9 below 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.

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[4] 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()`.

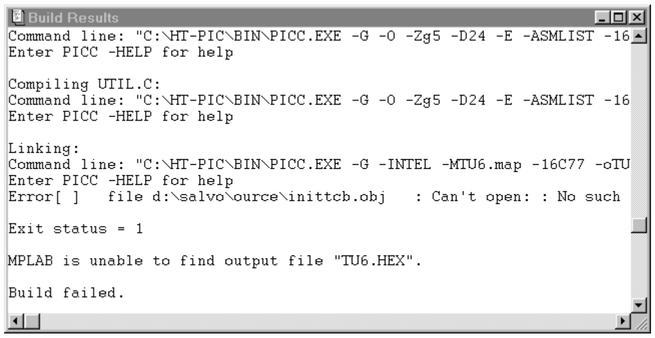


```
**                                                          **
****                                                      ****
*************************************************************/
OStypeErr OSCreateTask( OStypeTFP  task,
                        OStypeTcbP tcbP,
                        OStypePrio prio )
{
    OSEnterCritical();
    OSIncCallDepth();

    /* punt if tcbP is clearly bad.                             */
    #if OSENABLE_BOUNDS_CHECKING
    if ( (tcbP < OSTCBP(1)) || (tcbP > OSTCBP(OSTASKS)) ) {
        OSWarnRtn("OSCreateTask",
          OSMakeStr("task %d nonexistent or invalid.",
            OSeID(ecbP)), OSERR_BAD_P);
    }
    #endif


    /* ensure specified task isn't already created.             */
    #if OSENABLE_ERROR_CHECKING
    if ( tcbP->status.bits.state != (OStypeInt8u) OSTA_DESTROY
        OSDecCallDepth();
        OSExitCritical();
        OSWarnRtn("OSCreateTask",
```

Figure 10: Stepping Through Salvo Source Code

Troubleshooting

Occasionally you may get an error when `picc.exe` links the project. This error manifests itself in an incorrectly formatted command line containing the full path name of the one Salvo object module (often the last) being linked to form your application. This is shown in Figure 11 – note the incorrect "`d:\salvo\ource\inittcb.obj`".



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

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

Linking:
Command line: "C:\HT-PIC\BIN\PICC.EXE -G -INTEL -MTU6.map -16C77 -oTU
Enter PICC -HELP for help
Error[ ]   file d:\salvo\ource\inittcb.obj   : Can't open: : No such

Exit status = 1

MPLAB is unable to find output file "TU6.HEX".

Build failed.
```

Figure 11: Link Error Involving Path Name

Select Project → Edit Project, select the first in the list (`main.c` in this example), select Delete Node. Add the node back to the project using Copy Node as outlined above – this will place it at the end of the list. Re-make the Project.

Alternatively, exiting and re-starting MPLAB may fix this error.

Building with Salvo Libraries

You can also build applications using the Salvo standard libraries. The procedure is very similar to the one outlined above, with the following exceptions:

- You don't need to add any of Salvo's source files as nodes in your project,
- You must link to the standard library that's appropriate for your target processor and the Salvo functionality you desire, and
- Your project's `salvocfg.h` contains just a few configuration options to specify which type of library you're using.

AN-1 Using Salvo Freeware Libraries with the HI-TECH PICC Compiler describes in-depth how to build an application using the freeware libraries. Building applications with the standard libraries is substantially similar. When using the standard libraries, be sure to select the correct (standard) library when adding it to your project's nodes. More information on the standard libraries and how to identify them is found in the *Libraries* chapter of the *Salvo User Manual*.

A complete project using the standard libraries is contained in the MPLAB project file `\salvo\tut\tu6\sysa\tu6le.pjt`.

A complete project using the freeware libraries is contained in the MPLAB project file `\salvo\tut\tu6\sysa\tu6lite.pjt`.

1 The Development Mode window allows you to select both the development mode (editor only, simulator, in-circuit emulator, etc.) and the particular target PICmicro device.

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 You can Ctrl-select multiple files at once.

[4] 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.