PUMPKIN™
REAL-TIME SOFTWARE

AN-23
Application Note

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

Building a Salvo Application with Rowley Associates' CrossStudio for MSP430

Introduction

This Application Note explains how to use Rowley Associates' (http://www.rowley.co.uk/) CrossStudio for MSP430 to create a multitasking Salvo application for Texas Instruments' (http://www.ti.com/) MSP430 ultra-low-power microcontrollers.

We will show you how to build the Salvo application contained in `\salvo\ex\ex1\main.c` for an MSP430F149 using CS430. 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 CrossStudio for MSP430. Familiarize yourself with the CS430 IDE.

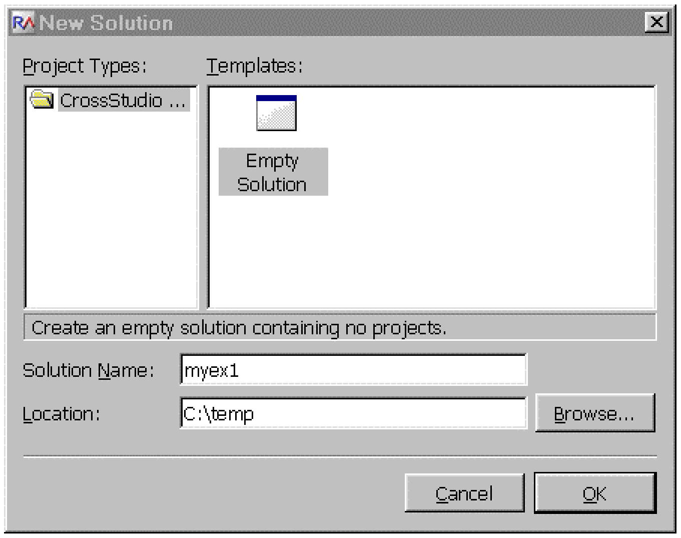
Related Documents

The following Salvo documents should be used in conjunction with this manual when building Salvo applications with Rowley Associates' CrossStudio for MSP430:

Salvo User Manual
Salvo Compiler Reference Manual RM-CS430

Creating and Configuring a New Project

Create a new CS430 solution under File → New → New Blank Solution. Navigate to your working directory (in this case we've chosen `c:\temp`) and create a solution named `myex1`:


New Solution

Project Types: CrossStudio ...

Templates: Empty Solution

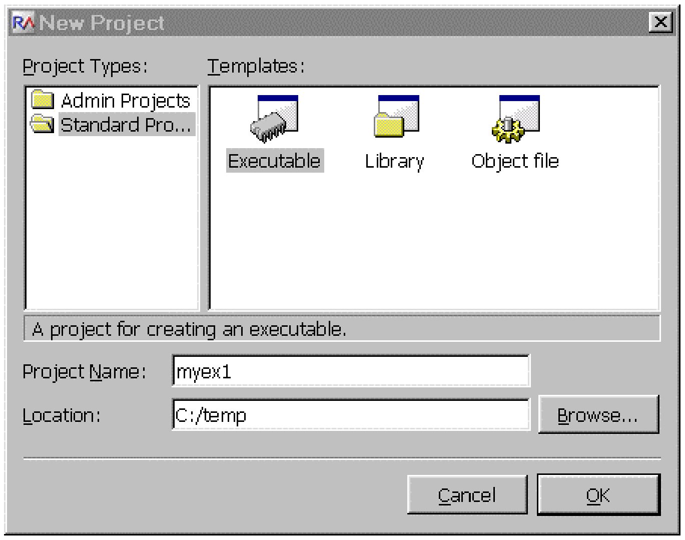
Create an empty solution containing no projects.

Solution Name: myex1

Location: C:\temp

Figure 1: Creating the New Solution

Click OK to continue. Now create a new CS430 project under File → New → New Project. Again, navigate to your working directory and create a standard project named `myex1` using the Executable template:


New Project

Project Types: Admin Projects, Standard Pro...

Templates: Executable, Library, Object file

A project for creating an executable.

Project Name: myex1

Location: C:/temp

Figure 2: Creating the New Project

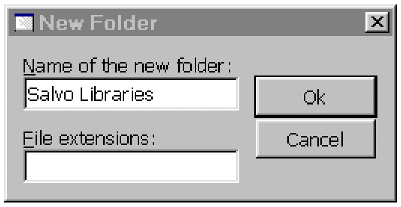
Click OK to continue. CS430 will automatically save the project whenever you close it.

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

Listings
Salvo Configuration File
Salvo Help Files
Salvo Libraries

Salvo Source Files
Source Files

For each[1] folder, left-click on the project in the Project Explorer window, right-click, choose New Folder…, enter the desired name under Name of the new folder and click Ok.

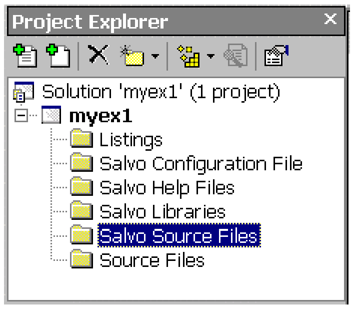

New Folder

Name of the new folder: Salvo Libraries

File extensions:

Figure 3: Creating a Folder

When finished, your Project Explorer[2] window should look like this:


Project Explorer

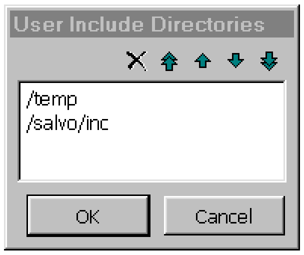
Solution 'myex1' (1 project)
myex1
Listings
Salvo Configuration File
Salvo Help Files
Salvo Libraries
Salvo Source Files
Source Files

Figure 4: Project Explorer Window with Folders

Now let's setup the project's options for Salvo's pathnames, etc.

Note We recommend that you do set of these options *at the solution level* unless it's imperative that they be set at the project level.

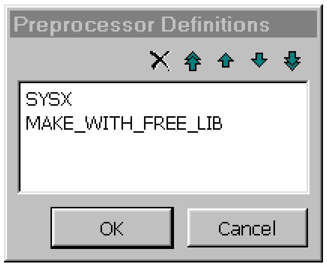
Select the solution by left-clicking on the solution in the Project Explorer window. In the Properties window[3], under Preprocessor Options → User Include Directories, add[4] the project's own include path and `/salvo/inc`:[5]


User Include Directories

```
/temp
/salvo/inc
```

Figure 5: CS430 Options – User Include Directories

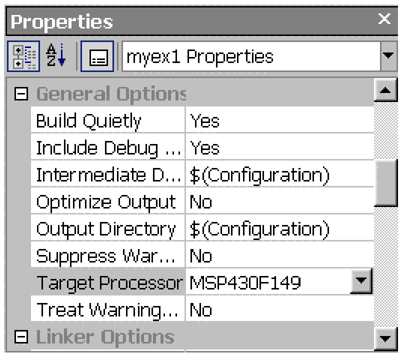
Next, define any symbols[6] you may need for your project under Preprocessor Options → Preprocessor Definitions:



```
SYSX
MAKE_WITH_FREE_LIB
```

Figure 6: CS430 Options – Preprocessor Definitions

Lastly, in the Properties window, select the MSP430F149 under General Options → Target Processor.

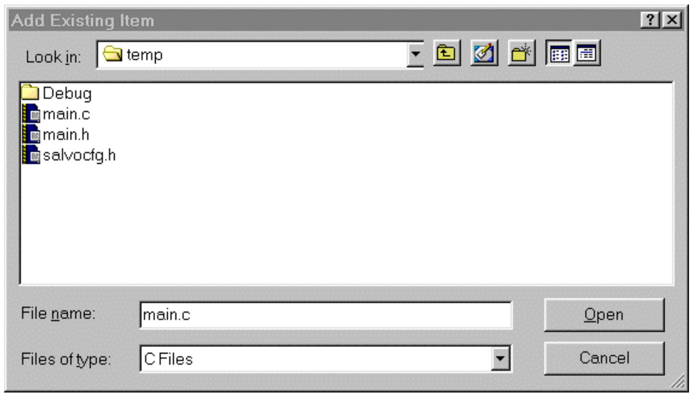


General Options	
Build Quietly	Yes
Include Debug ...	Yes
Intermediate D...	$(Configuration)
Optimize Output	No
Output Directory	$(Configuration)
Suppress War...	No
Target Processor	MSP430F149
Treat Warning...	No
Linker Options	

Figure 7: CS430 Options –Target Processor Selection

Adding your Source File(s) to the Project

Now it's time to add files to your project. In the Project Explorer window, select the Source Files folder, right-click to choose Add Existing File …, choose Files of type: C Files, navigate to your project's directory, select your `main.c` and click Open. Your Add Files… window should look like this:

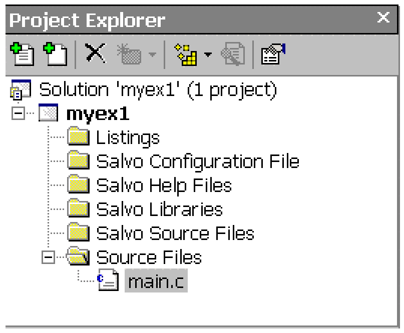


Debug
main.c
main.h
salvocfg.h

File name: main.c
Files of type: C Files

Figure 8: Project Files Window

When finished, your Project Manager window should look like this:



Solution 'myex1' (1 project)
- myex1
  - Listings
  - Salvo Configuration File
  - Salvo Help Files
  - Salvo Libraries
  - Salvo Source Files
  - Source Files
    - main.c

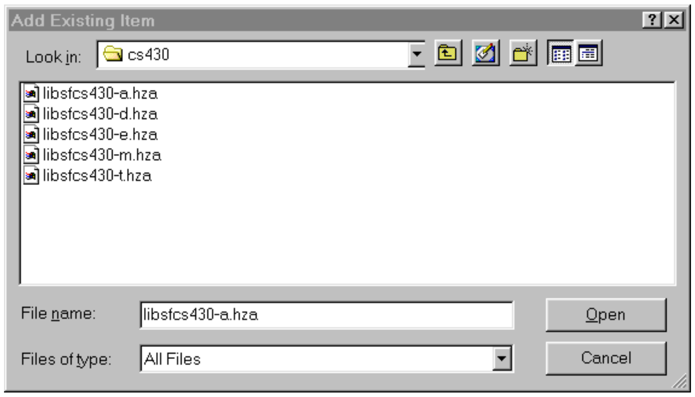
Figure 9: Project Manager Window with Project-Specific Source Files

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 MSP430 for use with CS430 is `libsfcs430-a.hza`.[7] Select the Salvo Libraries folder in the Project Explorer window, right-click, choose Add Existing File …, choose Files of type: All Files, navigate to the `\salvo\lib\cs430` directory and select `libsfcs430-a.hza`:



Look in: cs430

libsfcs430-a.hza
libsfcs430-d.hza
libsfcs430-e.hza
libsfcs430-m.hza
libsfcs430-t.hza

File name: libsfcs430-a.hza
Files of type: All Files

Figure 10: Adding the Library to the Project

Click Open 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-CS430*.

Adding Salvo's mem.c

Salvo library builds also require Salvo's `mem.c` source file as part of each project. Select the Salvo Source Files folder in the Project Explorer window, right-click, choose Add Existing File …, choose Files of type: All Files, navigate to the `\salvo\src` directory and select `mem.c`:

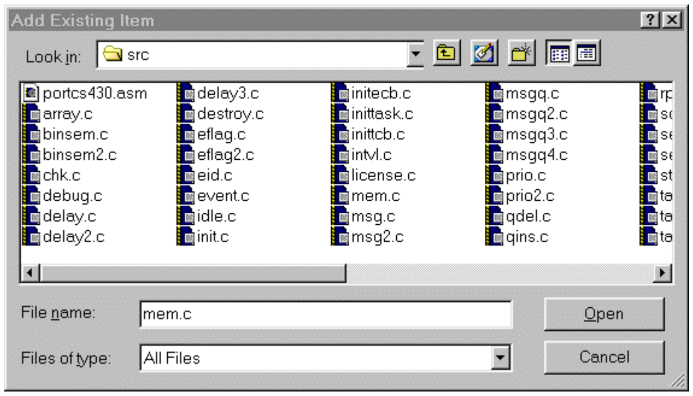


Figure 11: Adding mem.c to the Project

Click Open when you are finished. Your Project Explorer window will look like this:

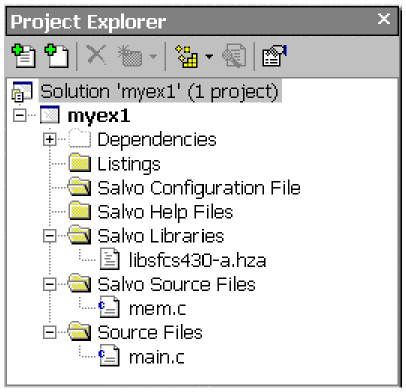


Figure 12: Project Manager Window for a Library Build

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_TYPE         OSF
#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`. Add it to the Salvo Configuration File folder in the Project Explorer.

Proceed to *Building the Project*, below.

Adding Salvo Source Files

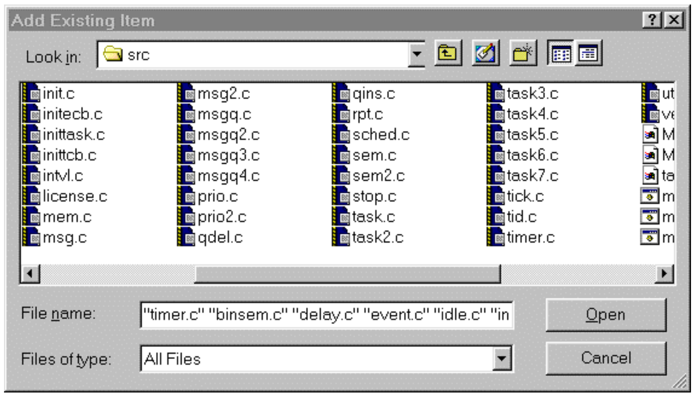
If you have a Salvo distribution that contains source files, you can do a *source code build* instead of a library build. The application in `\salvo\ex\ex1\main.c` contains calls to the following Salvo user services:

```
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                 portcs430.asm
event.c                 qins.c
idle.c                  sched.c
init.c                  timer.c
inittask.c
```

In the Project Manager window, select the Salvo Sources folder, right-click to choose Add Files…, choose Files of type: Source Files (*.c, *.s, *.h), navigate to the `\salvo\src` directory and select[8] the `*.c` files listed above. Your Add Files… window should look like this:



File name: "timer.c" "binsem.c" "delay.c" "event.c" "idle.c" "in

Files of type: All Files

Figure 13: Adding Salvo Source Files to the Project

Click Open when finished.

The salvocfg.h Header File

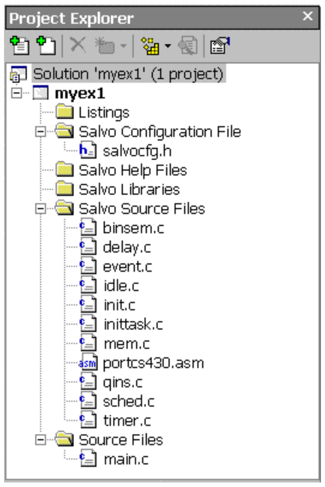
You will also need a `salvocfg.h` file for this project. Configuration files for source code builds are quite different from those for library builds (see Listing 1, above). For a source code build, the `salvocfg.h` for this project contains only:

```
#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`. Add it to the Salvo Configuration File folder in the Project Explorer.

Your Project Manager window should now look like this:



```
Project Explorer
Solution 'myex1' (1 project)
  myex1
    Listings
    Salvo Configuration File
      salvocfg.h
    Salvo Help Files
    Salvo Libraries
    Salvo Source Files
      binsem.c
      delay.c
      event.c
      idle.c
      init.c
      inittask.c
      mem.c
      portcs430.asm
      qins.c
      sched.c
      timer.c
    Source Files
      main.c
```

Figure 14: Complete Project Manager Window for a Source-Code 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 viewing, editing, etc.

Building the Project

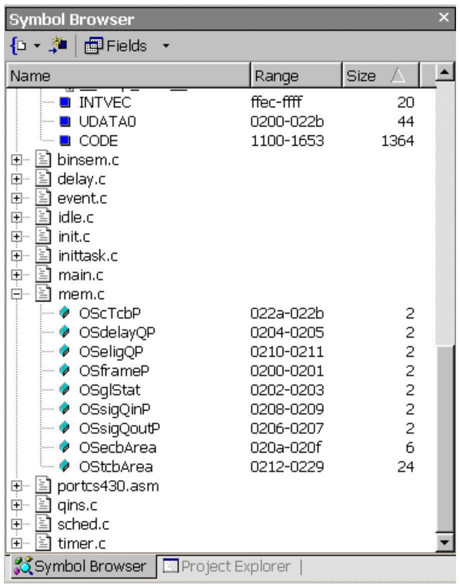
For a successful compile, your project must also include a header file (e.g. `#include <msp430x14x.h>`) for the particular chip you are using. Normally, this is included in each of your source files (e.g. `main.c`), or in a header file that's included in each of your source files (e.g. `main.h`).

With everything in place, you can now build the project using Project → Rebuild myex1 or Project → Rebuild Solution. The IDE's status window will reflect the hcc command lines:

```
Delete C:/temp/Debug/myex1.hzx
[SNIP]
Delete C:/temp/Debug/portcs430.hzo
Debug/main.hzo does not exist.
[SNIP]
Debug/portcs430.hzo does not exist.
Compiling main.c [Debug]
cd C:/temp
C:/PROGRAM FILES/ROWLEY ASSOCIATES LIMITED/CROSSWORKS MSP430
1.0.0/bin/hcc -DSYSX -DMAKE_WITH_SOURCE -D__CROSSWORKS
-D__TARGET_PROCESSOR=MSP430F149 -g -I/temp -I/salvo/inc
-JC:/PROGRAM FILES/ROWLEY ASSOCIATES LIMITED/CROSSWORKS MSP430
1.0.0/include main.c -o Debug/main.hzo -mmpy -
D__CROSSWORKS_MSP430
[SNIP]
Assembling portcs430.asm [Debug]
cd C:/temp
C:/PROGRAM FILES/ROWLEY ASSOCIATES LIMITED/CROSSWORKS MSP430
1.0.0/bin/has -DSYSX -DMAKE_WITH_SOURCE -D__CROSSWORKS
-D__TARGET_PROCESSOR=MSP430F149 -g -I/temp -I/salvo/inc
-JC:/PROGRAM FILES/ROWLEY ASSOCIATES LIMITED/CROSSWORKS MSP430
1.0.0/include ../salvo/src/portcs430.asm -o Debug/portcs430.hzo
-D__CROSSWORKS_MSP430
Linking myex1 [Debug]
cd C:/temp
C:/PROGRAM FILES/ROWLEY ASSOCIATES LIMITED/CROSSWORKS MSP430
1.0.0/bin/hld C:/PROGRAM FILES/ROWLEY ASSOCIATES
LIMITED/CROSSWORKS MSP430 1.0.0/lib/crt0.hzo Debug/main.hzo
Debug/mem.hzo Debug/timer.hzo Debug/binsem.hzo Debug/delay.hzo
Debug/event.hzo Debug/idle.hzo Debug/init.hzo
Debug/inittask.hzo Debug/qins.hzo Debug/sched.hzo
Debug/portcs430.hzo -g -LC:/PROGRAM FILES/ROWLEY ASSOCIATES
LIMITED/CROSSWORKS MSP430 1.0.0/lib -
D___vfprintf=___vfprintf_int -D___vfscanf=___vfscanf_int
-GCODE=L -GCONST=L -TINTVEC=ffe0 -GINTVEC=L
-TCODE,CONST=1100-ffff -T.abs=0 -G.abs=L -
TIDATA0,UDATA0=200-9ff -DRAM_Start_Address=512 -DRAM_Size=2048
-Fhzx -o Debug/myex1.hzx
Build Complete
```

Listing 3: Build Results for A Successful Source-Code Build

The Symbol Browser[9] contains address, symbol and other useful information:



Name	Range	Size
INTVEC	ffec-ffff	20
UDATA0	0200-022b	44
CODE	1100-1653	1364
binsem.c		
delay.c		
event.c		
idle.c		
init.c		
inittask.c		
main.c		
mem.c		
OScTcbP	022a-022b	2
OSdelayQP	0204-0205	2
OSeligQP	0210-0211	2
OSframeP	0200-0201	2
OSglStat	0202-0203	2
OSsigQinP	0208-0209	2
OSsigQoutP	0206-0207	2
OSecbArea	020a-020f	6
OStcbArea	0212-0229	24
portcs430.asm		
qins.c		
sched.c		
timer.c		

Figure 15: Symbol Browser Window

In Figure 15 you can see that the application requires 44 bytes of RAM (`UDATA0`) and 1,364 instructions in ROM (`CODE`). Additionally, you can see the locations and the sizes of Salvo's global variables (e.g. `OSeligQP`, the eligible queue pointer).

Note The CS430 projects supplied in the Salvo for TI's MSP430 distributions contain additional help files in each project's Salvo Help Files group.

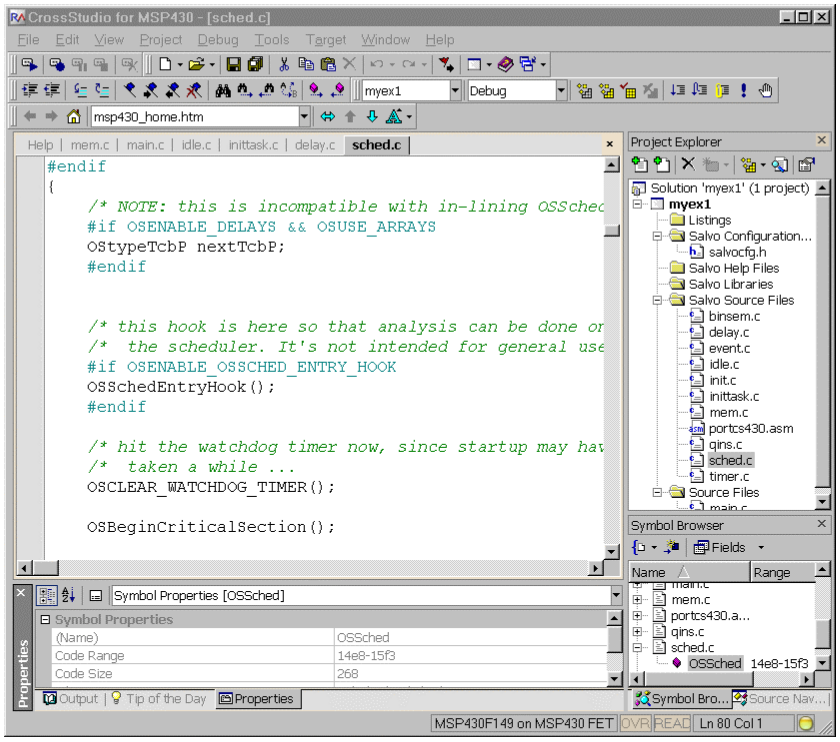
IDE Extras

Disassembly

One very useful feature of the CS430 IDE is the ability to view the assembly-language output of the compiler directly inside the IDE. To do this, the project must have been successfully built. Then, select (left-click) the C file of interest, then right-click and choose Disassemble. A window will open, displaying the C-language source code along with the MSP430 instructions.

Browsing

CS430's Symbol Browser enables you to go directly to the source where functions and variables are defined. This works with any source code (project- or Salvo-specific), and also with the `i`-option Salvo libraries that include debugging information.


CrossStudio for MSP430 - [sched.c]

```
#endif
{
    /* NOTE: this is incompatible with in-lining OSSchec
    #if OSENABLE_DELAYS && OSUSE_ARRAYS
    OStypeTcbP nextTcbP;
    #endif


    /* this hook is here so that analysis can be done or
    /*  the scheduler. It's not intended for general use
    #if OSENABLE_OSSCHED_ENTRY_HOOK
    OSSchedEntryHook();
    #endif

    /* hit the watchdog timer now, since startup may hav
    /*  taken a while ...
    OSCLEAR_WATCHDOG_TIMER();

    OSBeginCriticalSection();
```

Symbol Properties [OSSched]

Symbol Properties	
(Name)	OSSched
Code Range	14e8-15f3
Code Size	268

```
Solution 'myex1' (1 project)
  myex1
    Listings
    Salvo Configuration...
      salvocfg.h
    Salvo Help Files
    Salvo Libraries
    Salvo Source Files
      binsem.c
      delay.c
      event.c
      idle.c
      init.c
      inittask.c
      mem.c
      portcs430.asm
      qins.c
      sched.c
      timer.c
    Source Files
```

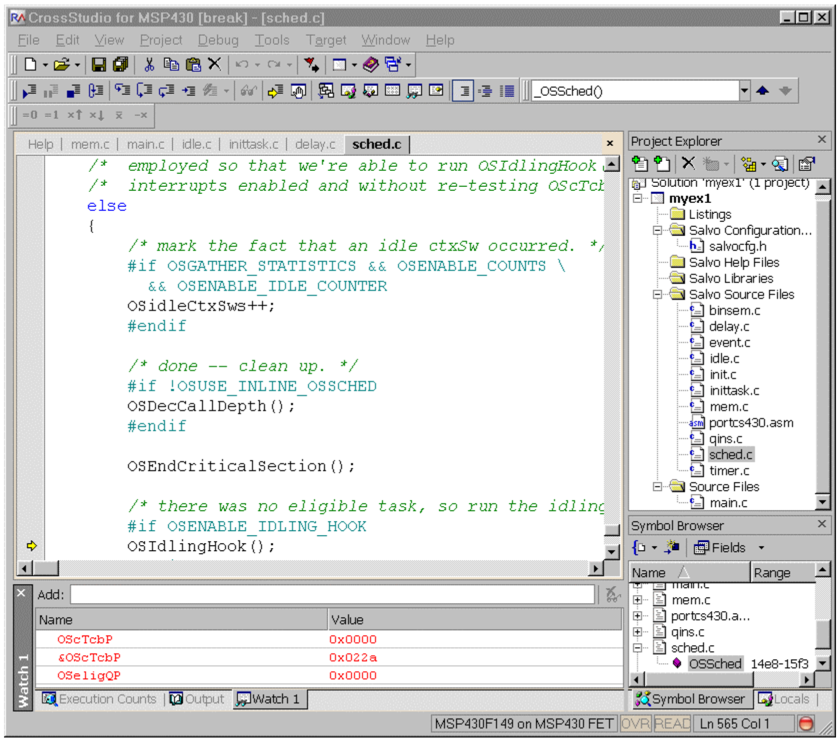
MSP430F149 on MSP430 FET | Ln 80 Col 1

Figure 16: Browsing at the Source-Code Level

Testing the Application

You can test and debug this application using the Flash Emulation Tool. Choose Debug → Start Debugging to begin a debugging session.

You can use all of the IDE's supported features when debugging and testing Salvo applications. This includes breakpoints, watch windows, etc.



```
    /*  employed so that we're able to run OSIdlingHook
    /*  interrupts enabled and without re-testing OScTcb
    else
    {
        /* mark the fact that an idle ctxSw occurred. */
        #if OSGATHER_STATISTICS && OSENABLE_COUNTS \
          && OSENABLE_IDLE_COUNTER
        OSidleCtxSws++;
        #endif

        /* done -- clean up. */
        #if !OSUSE_INLINE_OSSCHED
        OSDecCallDepth();
        #endif

        OSEndCriticalSection();

        /* there was no eligible task, so run the idling
        #if OSENABLE_IDLING_HOOK
        OSIdlingHook();
```

Name	Value
OScTcbP	0x0000
&OScTcbP	0x022a
OSeligQP	0x0000

Figure 17: Testing a Salvo Application in CS430

Note CS430 supports debugging at the source code level. Only Salvo Pro source-code-build or library-build projects 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 the IDE / debugger.

Troubleshooting

Cannot find and/or read 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:

```
c:/temp/main.c(15): error E1023: can't find
include file "salvo.h"
```

Figure 18: 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:

```
c:/temp/main.c(14): error E1023: can't find
include file "main.h"
/salvo/inc/salvo.h(331): error E1023: included
from c:/temp/main.c(15): can't find include
file "salvocfg.h"
```

Figure 19: 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 CS430 are found in the `\salvo\tut\tu1-6\sysx` directories. The include path for each of these projects includes `\salvo\tut\tu1\sysx`, and each project defines the `SYSX` symbol.

Complete projects using Salvo freeware libraries are contained in the project files `\salvo\tut\tu1-6\sysx\tu1-6lite.hzp`. 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\sysx\tu1-6le.hzp`. These projects also define the `MAKE_WITH_STD_LIB` symbol.

Complete projects using Salvo standard libraries with embedded debugging information are contained in the project files `\salvo\tut\tu1-6\sysx\tu1-6prolib.hzp`. 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\sysx\tu1-6pro.hzp`. These projects also define the `MAKE_WITH_SOURCE` symbol.

---

[1] The Source Files folder is already present in a new project.

[2] Use View → Project Explorer if not visible.

[3] Use View → Properties if not visible. This App Note assumes you have selected Categorized View to arrange the properties.

[4] You can type directory names separated by semicolons (';'), or you can click on the ellipsis ('…') button at the end of the user entry field, which will

present you with the User Include Directories window, which greatly facilitates user entry.

5 CS430 also supports pathnames relative to the project's home directory. Using relative pathnames is recommended, as it makes a project much more portable.

6 This Salvo project supports a wide variety of targets and compilers. For use with CS430, it requires the `SYSX` 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.

7 This Salvo Lite library contains all of Salvo's basic functionality. The corresponding Salvo LE and Pro libraries are `libslcs430-a.a` and `libslcs430ia.a`, respectively.

8 You can Ctrl-select multiple files at once.

9 Use View → Symbol Browser if not visible.