PUMPKIN™
REAL-TIME SOFTWARE

AN-29
Application Note

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

Building a Salvo Application with IAR's AVR C Compiler and Embedded Workbench IDE

Introduction

This Application Note explains how to use IAR's (http://www.iar.com/) AVR C compiler and Embedded Workbench IDE to create a multitasking Salvo application for Atmel's' (http://www.atmel.com/) AVR and MegaAVR 8-bit microcontrollers.

We will show you how to build the Salvo application contained in `\salvo\ex\ex1\main.c` for an AT90S8515 using IAR Embedded Workbench for Atmel AVR.

Note IAR Embedded Workbench for Atmel AVR underwent changes between v2 (e.g. EW 2.31E, with the last version of the compiler being v2.28A) and v3 (e.g. EW 3.3A, with v3.x compilers, e.g. v3.10A). The procedures and illustrations in this document are from IAR Embedded Workbench for Atmel AVR v2.31E and the associated IAR AVR C compiler v2.28A. Where substantive differences exist, they will be noted as such.

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 Atmel AVR. You may also install the latest version of Atmel's AVR Studio, available at http://www.avrfreaks.com/.

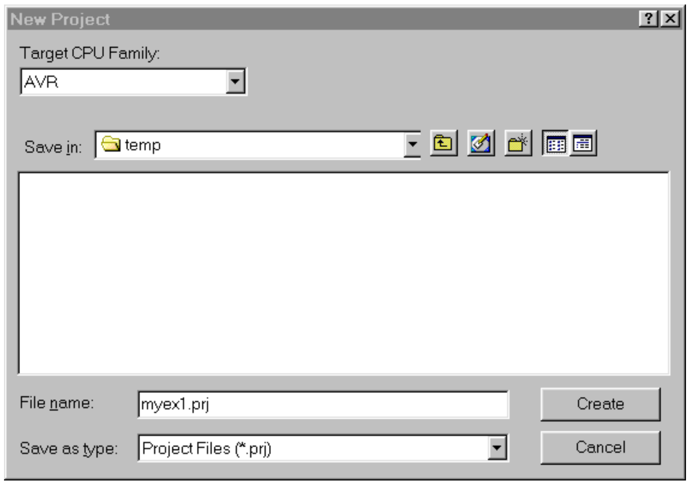
Related Documents

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

Salvo User Manual
Salvo Compiler Reference Manual RM-IARAVR

Creating and Configuring a New Project

Create a new Embedded Workbench project under File → New → Project → OK. Select AVR as the Target CPU Family, navigate to your working directory (in this case we've chosen `c:\temp`) and create a project named `myex1.prj`:



New Project

Target CPU Family: AVR

Save in: temp

File name: myex1.prj

Save as type: Project Files (*.prj)

Figure 1: Creating the New Project

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

Note In Embedded Workbench for Atmel AVR v3, first you create a *workspace*, and then you create one or more projects within that workspace.

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.

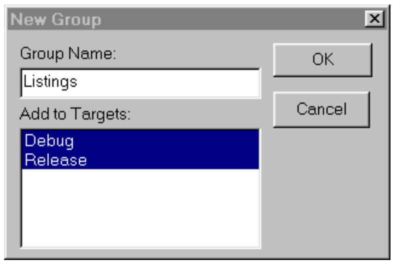


Figure 2: Creating a Group

When finished, your project window should look like this:

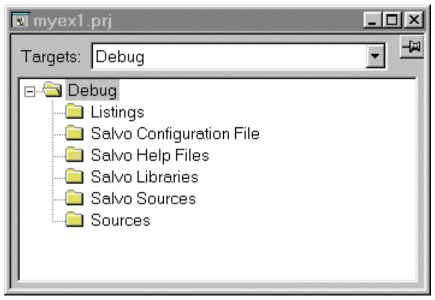


Figure 3: Project Window with Groups

Now let's setup the project's options for processor type, Salvo's pathnames, etc. for your particular Atmel AVR microcontroller.

Select → Options → General → Target, and under Processor configuration and Memory model make the appropriate selections for your project.

Note Wherever possible, you should always choose a Processor configuration that matches your target processor as closely as possible. Avoid using the generic (`-vN`) options unless your particular configuration requires them.

Your choice of Memory model will dictate which Salvo library you use in this project.

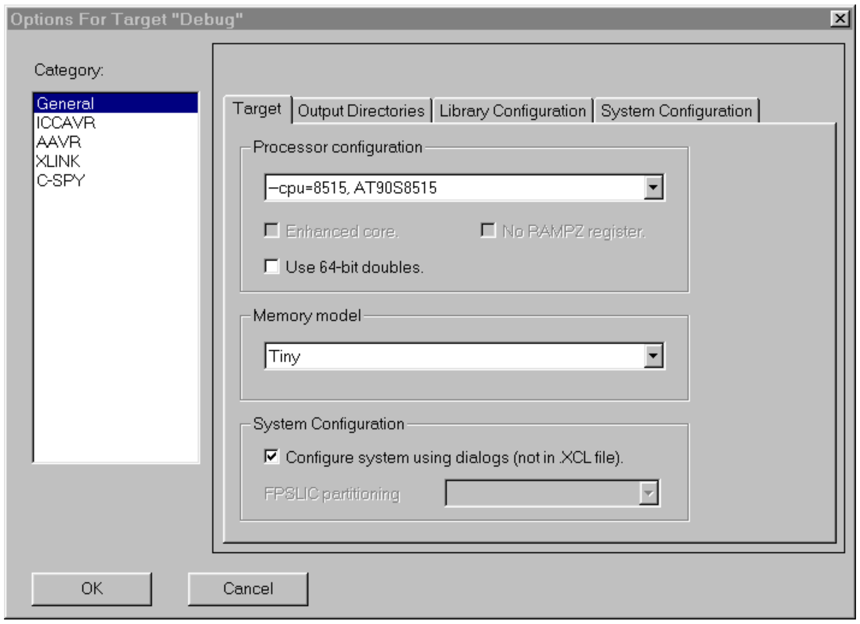
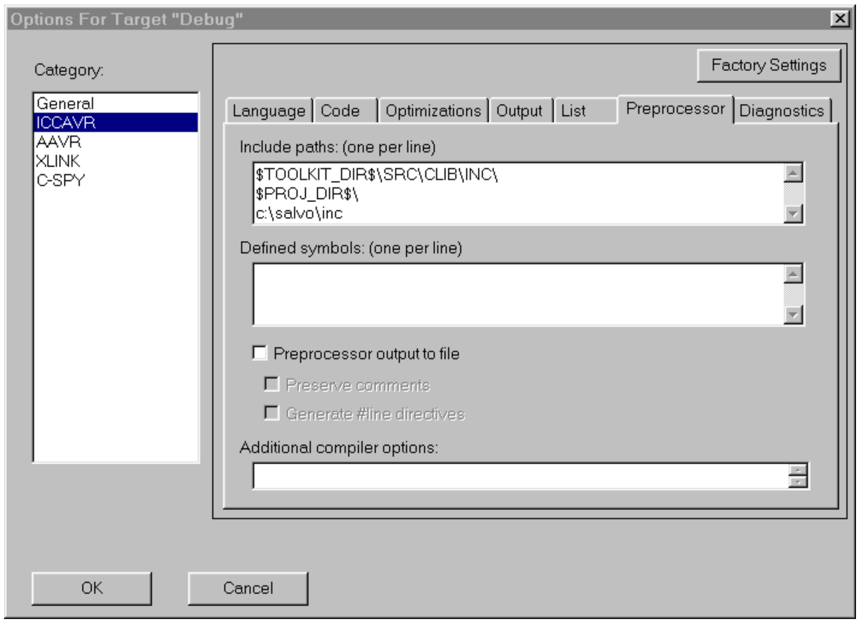


Figure 4: General Settings – Processor Configuration and Memory Model

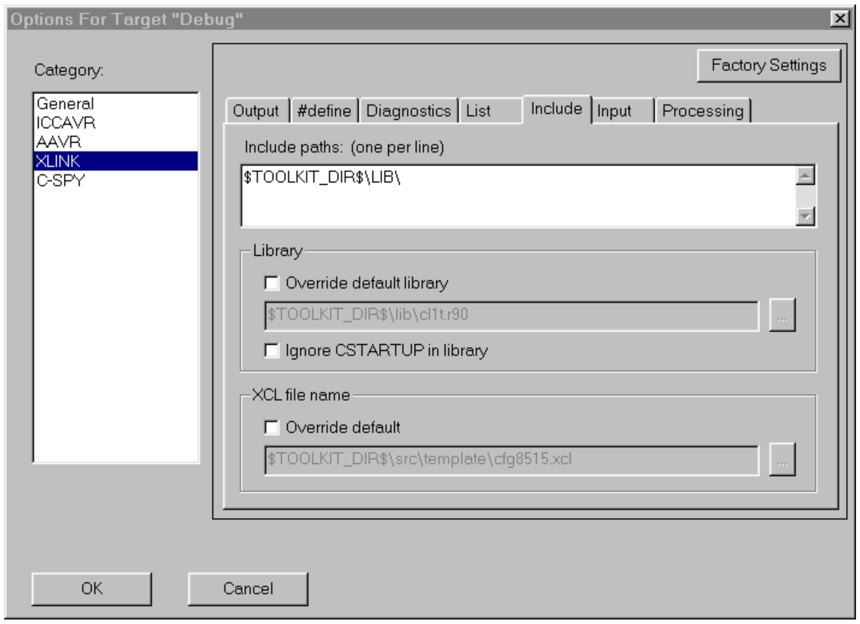
Select Project → Options → ICCAVR → Preprocessor and define any symbols you may need for your project[1], and add the include paths `$PROJ_DIR$\` and `c:\salvo\inc\`:


Options For Target "Debug"
Factory Settings
Category:
General
ICCAVR
AAVR
XLINK
C-SPY
Language
Code
Optimizations
Output
List
Preprocessor
Diagnostics
Include paths: (one per line)
Defined symbols: (one per line)
Preprocessor output to file
Preserve comments
Generate #line directives
Additional compiler options:
OK
Cancel

```
$TOOLKIT_DIR$\SRC\CLIB\INC\
$PROJ_DIR$\
c:\salvo\inc
```

Figure 5: ICCAVR Settings – Project Include Paths and Defined Symbols

Next, select XLINK → List → Generate Linker listing. This will create a useful `.map` file with the application's ROM and RAM requirements, etc. Under XLINK → Include, ensure that the automatically selected `.xcl` linker filename matches your target processor.[2]



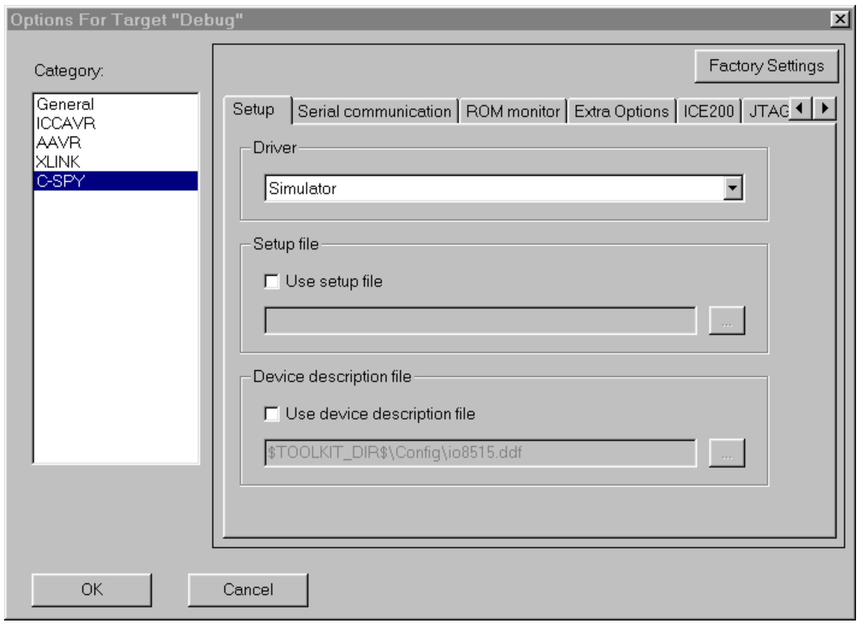
Include paths: (one per line)
$TOOLKIT_DIR$\LIB\

Override default library
$TOOLKIT_DIR$\lib\cl1t.r90
Ignore CSTARTUP in library

Override default
$TOOLKIT_DIR$\src\template\cfg8515.xcl

Figure 6: XLINK Settings

Lastly, under C-SPY → Setup, select the Driver (ICE200 emulator, JTAG emulator Tool, ROM-monitor CC or Simulator) and ensure that the Device description file is the appropriate one for your AVR:



Driver: Simulator

Use setup file

Use device description file
$TOOLKIT_DIR$\Config\io8515.ddf

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, select 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:

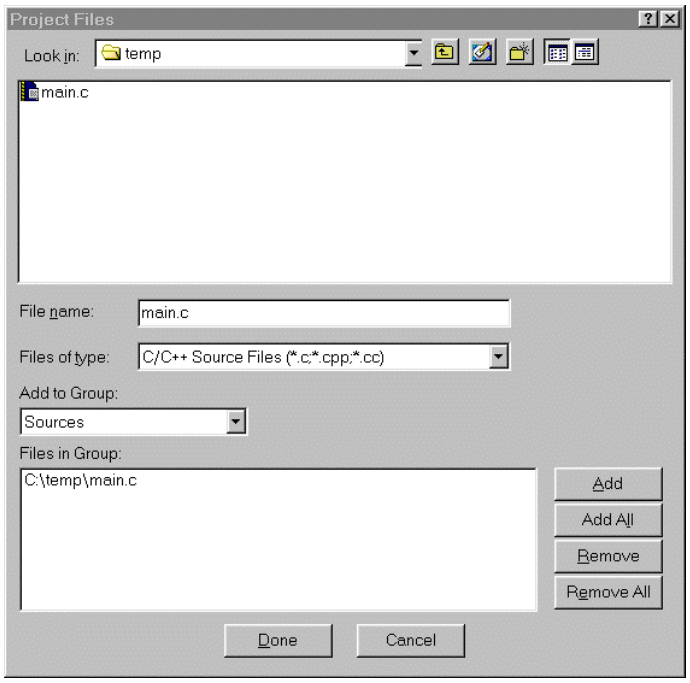


Figure 8: Project Files Window

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

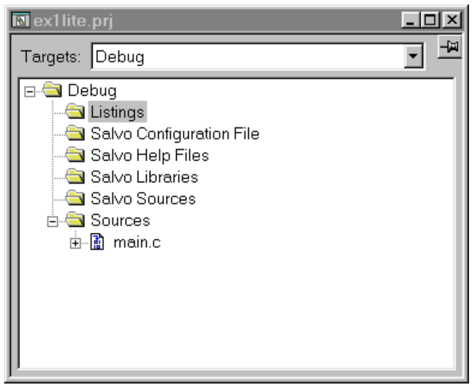


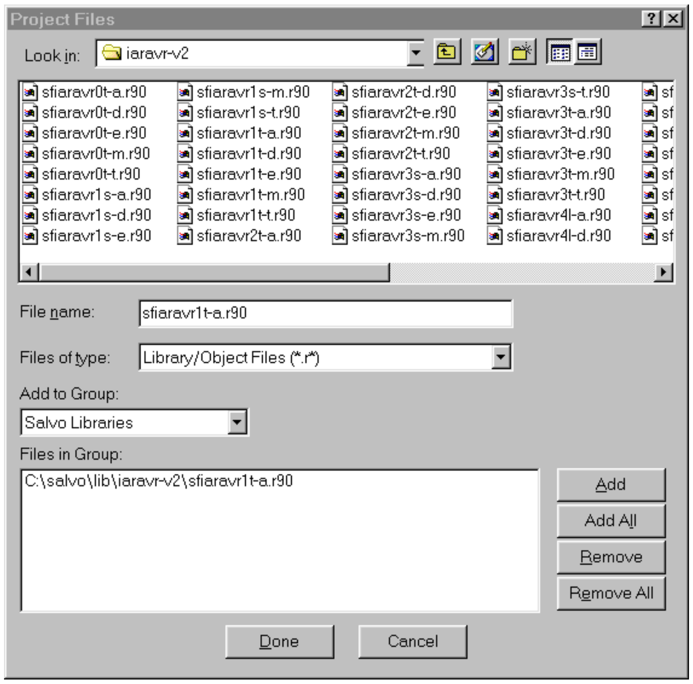
Figure 9: Project Window with Project-Specific Source Files

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 AT90S8515 when using the tiny memory model is `sfiaravr1t-a.r90`.[3] Select Project → Files, Library/Object Files (*.r*) under Files of type, Salvo Libraries under Add to Group, navigate to the `\salvo\lib\iaravr-v2` directory, select `sfiaravr1t-a.r90` and Add:



Look in: iaravr-v2

sfiaravr0t-a.r90	sfiaravr1s-m.r90	sfiaravr2t-d.r90	sfiaravr3s-t.r90	sf
sfiaravr0t-d.r90	sfiaravr1s-t.r90	sfiaravr2t-e.r90	sfiaravr3t-a.r90	sf
sfiaravr0t-e.r90	sfiaravr1t-a.r90	sfiaravr2t-m.r90	sfiaravr3t-d.r90	sf
sfiaravr0t-m.r90	sfiaravr1t-d.r90	sfiaravr2t-t.r90	sfiaravr3t-e.r90	sf
sfiaravr0t-t.r90	sfiaravr1t-e.r90	sfiaravr3s-a.r90	sfiaravr3t-m.r90	sf
sfiaravr1s-a.r90	sfiaravr1t-m.r90	sfiaravr3s-d.r90	sfiaravr3t-t.r90	sf
sfiaravr1s-d.r90	sfiaravr1t-t.r90	sfiaravr3s-e.r90	sfiaravr4l-a.r90	sf
sfiaravr1s-e.r90	sfiaravr2t-a.r90	sfiaravr3s-m.r90	sfiaravr4l-d.r90	sf

File name: sfiaravr1t-a.r90

Files of type: Library/Object Files (*.r*)

Add to Group: Salvo Libraries

Files in Group: C:\salvo\lib\iaravr-v2\sfiaravr1t-a.r90

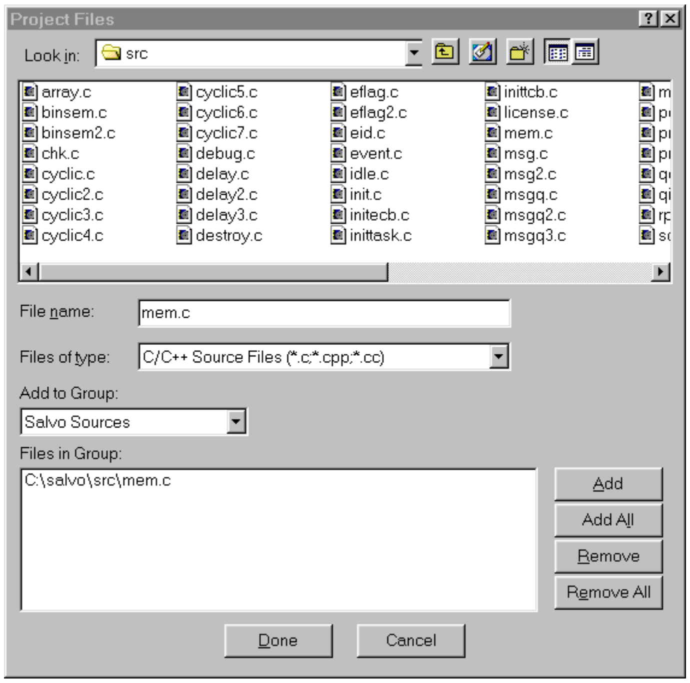
Figure 10: Adding the Library to the Project

Note Salvo libraries for IAR's v2.x AVR C compilers are located in `\salvo\lib\iaravr-v2`. Libraries for v3.x compilers are located in `\salvo\lib\iaravr-v3`. The libraries are not interchangeable.

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-IARAVR*.

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:


Project Files
Look in: src
array.c binsem.c binsem2.c chk.c cyclic.c cyclic2.c cyclic3.c cyclic4.c
cyclic5.c cyclic6.c cyclic7.c debug.c delay.c delay2.c delay3.c destroy.c
eflag.c eflag2.c eid.c event.c idle.c init.c initecb.c inittask.c
inittcb.c license.c mem.c msg.c msg2.c msgq.c msgq2.c msgq3.c
Add
Add All
Remove
Remove All
Done
Cancel

File name: mem.c
Files of type: C/C++ Source Files (*.c;*.cpp;*.cc)
Add to Group: Salvo Sources
Files in Group: C:\salvo\src\mem.c

Figure 11: Project Files Window

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

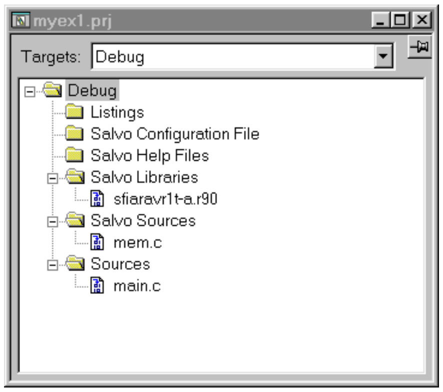


Figure 12: Project Window with Library, mem.c and User Source File(s)

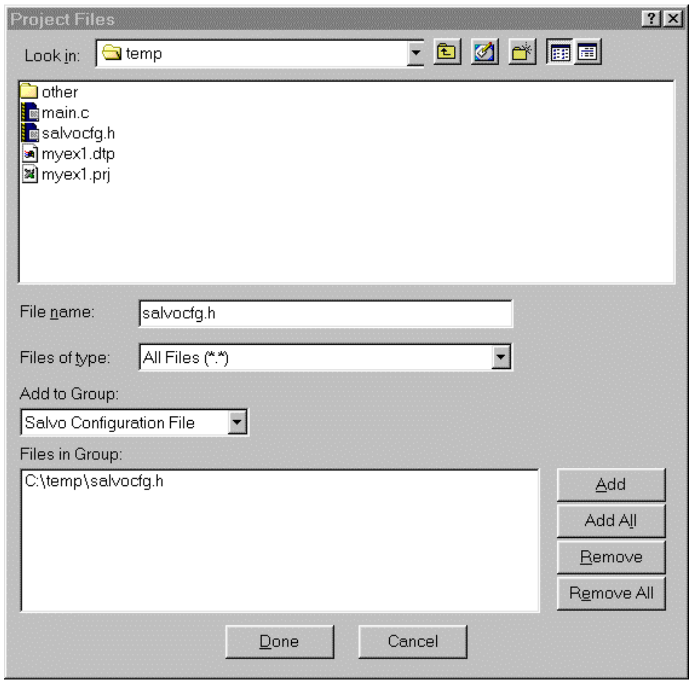
The salvocfg.h Header File

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

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:



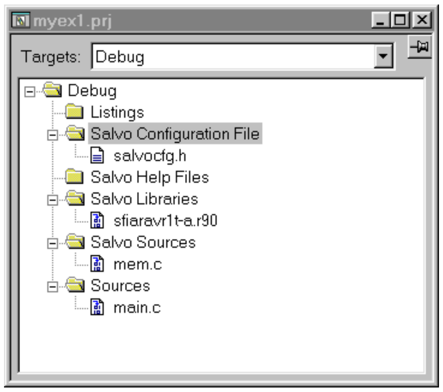
temp

other
main.c
salvocfg.h
myex1.dtp
myex1.prj

File name: salvocfg.h
Files of type: All Files (*.*)
Add to Group: Salvo Configuration File
Files in Group:
C:\temp\salvocfg.h

Figure 13: Adding the Configuration File to the Project

Your project window should now look like this:



Targets: Debug

```
Debug
  Listings
  Salvo Configuration File
    salvocfg.h
  Salvo Help Files
  Salvo Libraries
    sfiaravr1t-a.r90
  Salvo Sources
    mem.c
  Sources
    main.c
```

Figure 14: 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

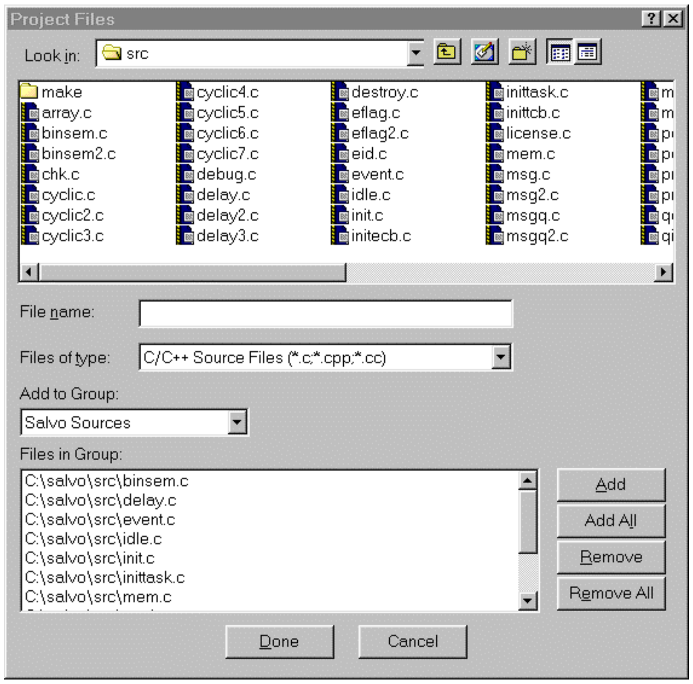
If you have a Salvo Pro distribution, you can do a *source code build* instead of a library build. The application in `\salvo\ex\ex1\main.c` contains calls to the following Salvo user services:

```
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                 portiaravr.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[4] the files listed above and Add:



Files of type: C/C++ Source Files (*.c;*.cpp;*.cc)

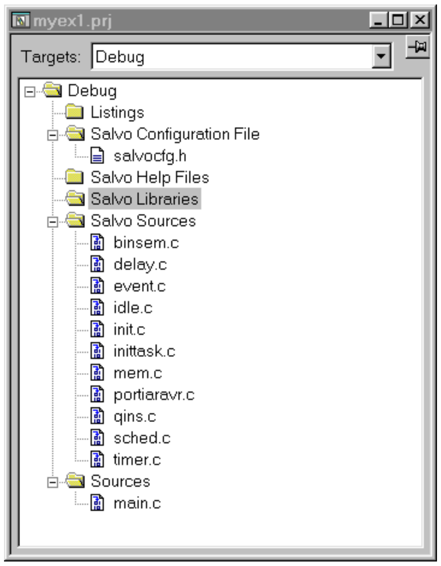
Add to Group: Salvo Sources

Files in Group:

```
C:\salvo\src\binsem.c
C:\salvo\src\delay.c
C:\salvo\src\event.c
C:\salvo\src\idle.c
C:\salvo\src\init.c
C:\salvo\src\inittask.c
C:\salvo\src\mem.c
```

Figure 15: Adding Salvo Source Files to the Project

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



```
Debug
  Listings
  Salvo Configuration File
    salvocfg.h
  Salvo Help Files
  Salvo Libraries
  Salvo Sources
    binsem.c
    delay.c
    event.c
    idle.c
    init.c
    inittask.c
    mem.c
    portiaravr.c
    qins.c
    sched.c
    timer.c
  Sources
    main.c
```

Figure 16: 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

Building the Project

For a successful compile, your project must also include a header file (e.g. `#include <io8515.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:[5]

```
################################################################################
#                                                                              #
#      IAR Universal Linker V4.530/WIN                                         #
#                                                                              #
#           Link time     =  21/Feb/2004  16:03:32                             #
#           Target CPU    =  A90                                               #
#           List file     =  C:\temp\Debug\List\myex1.map                      #
#           Output file 1 =  C:\temp\Debug\Exe\myex1.d90                       #
#                            Format: debug                                     #
#                            UBROF version 8.0.2                               #
#                            Using library modules for C-SPY (-rt)             #
#           Command line  =  C:\salvo\lib\iaravr-v2\sfiaravr1t-a.r90           #
#                            C:\temp\Debug\Obj\mem.r90                         #
#                            C:\temp\Debug\Obj\main.r90                        #
#                            C:\IAR\EW23\avr\lib\cl1t.r90 -o                   #
#                            C:\temp\Debug\Exe\myex1.d90 -rt -l                #
#                            C:\temp\Debug\List\myex1.map -xms                 #
#                            -IC:\IAR\EW23\avr\LIB\ -f                         #
#                            C:\IAR\EW23\avr\src\template\cfg8515.xcl          #
#                            (-D_..X_INTVEC_SIZE=1A -D_..X_FLASH_END=1FFF      #
#                            -D_..X_SRAM_BASE=60 -D_..X_SRAM_TEND=FF           #
#                            -D_..X_SRAM_END=25F -D_..X_EEPROM_END=1FF)        #
#                            -D_..X_HEAP_SIZE=10 -D_..X_CSTACK_SIZE=20         #
#                            -D_..X_RSTACK_SIZE=20                             #
#                            -e_small_write=_formatted_write                   #
#                            -e_medium_read=_formatted_read -f                 #
#                            C:\IAR\EW23\avr\src\template\cfg1t.xcl (-ca90     #
#                            -w29 -Z(CODE)INTVEC=0-(_..X_INTVEC_SIZE-1)        #
#                            -Z(CODE)TINY_F=_..X_FLASH_BASE-FF                 #
#                            -Z(CODE)NEAR_F=_..X_FLASH_BASE-_..X_FLASH_END     #
#                            -Z(CODE)SWITCH=_..X_FLASH_BASE-_..X_FLASH_END     #
#                            -Z(CODE)INITTAB=_..X_FLASH_BASE-_..X_FLASH_END    #
#                            -Z(CODE)DIFUNCT=_..X_FLASH_BASE-_..X_FLASH_END    #
#                            -Z(CODE)CODE=_..X_FLASH_BASE-_..X_FLASH_END       #
#                            -Z(CODE)TINY_ID=_..X_FLASH_BASE-_..X_FLASH_END    #
#                            -Z(CODE)NEAR_ID=_..X_FLASH_BASE-_..X_FLASH_END    #
#                            -Z(CODE)CHECKSUM=_..X_FLASH_BASE-_..X_FLASH_END   #
#                            -Z(DATA)TINY_I,TINY_Z,TINY_N=_..X_SRAM_BASE-_..X_ #
#                            SRAM_TEND                                         #
#                            -Z(DATA)CSTACK+_..X_CSTACK_SIZE=_..X_SRAM_BASE-_. #
#                            .X_SRAM_TEND                                      #
#                            -Z(DATA)HEAP+_..X_HEAP_SIZE=_..X_SRAM_BASE-_..X_S #
#                            RAM_TEND                                          #
#                            -Z(DATA)RSTACK+_..X_RSTACK_SIZE=_..X_SRAM_BASE-_. #
#                            .X_SRAM_END,_..X_EXT_RSTACK_BASE-_..X_EXT_RSTACK_ #
#                            END                                               #
#                            -Z(DATA)NEAR_I,NEAR_Z=_..X_SRAM_BASE-_..X_SRAM_EN #
#                            D,_..X_EXT_SRAM_BASE-_..X_EXT_SRAM_END            #
#                            -Z(DATA)NEAR_C=_..X_EXT_ROM_BASE-_..X_EXT_ROM_END #
#                            -Z(DATA)NEAR_N=_..X_EXT_NV_BASE-_..X_EXT_NV_END   #
#                            -Z(XDATA)EEPROM_I,EEPROM_N=1-_..X_EEPROM_END)     #
#                            -D_..X_FLASH_BASE=_..X_INTVEC_SIZE -H1895         #
#                            -h(CODE)0-(_..X_INTVEC_SIZE-1)                    #
#                            -D_..X_EXT_SRAM_BASE=_..X_SRAM_BASE               #
#                            -D_..X_EXT_SRAM_END=_..X_SRAM_BASE                #
#                            -D_..X_EXT_ROM_BASE=_..X_SRAM_BASE                #
#                            -D_..X_EXT_ROM_END=_..X_SRAM_BASE                 #
#                            -D_..X_EXT_NV_BASE=_..X_SRAM_BASE                #
#                            -D_..X_EXT_NV_END=_..X_SRAM_END                  #
#                            -D_..X_EXT_CSTACK_BASE=_..X_SRAM_BASE            #
#                            -D_..X_EXT_CSTACK_END=_..X_SRAM_BASE             #
#                            -D_..X_EXT_RSTACK_BASE=_..X_SRAM_BASE            #
#                            -D_..X_EXT_RSTACK_END=_..X_SRAM_BASE             #
#                                                                             #
#                        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 - 00000009              A  com    1
?FILL1               CODE          0000000A - 00000019             10  rel    0
INITTAB              CODE          0000001A - 0000001F              6  rel    0
CODE                 CODE          00000020 - 000005FD            5DE  rel    1
ABSOLUTE (ABS)       DATA               0000001C                       rel    0
                     DATA               0000001D
                     DATA               0000001E
                     DATA               0000001F
                     DATA          00000037 - 00000037              1
                     DATA          00000038 - 00000038              1
                     DATA          0000004A - 0000004B              2
                     DATA          0000004C - 0000004D              2
                     DATA          0000004E - 0000004E              1
                     DATA          00000059 - 00000059              1
TINY_I               DATA               00000060                       dse    0
TINY_Z               DATA          00000060 - 0000007F             20  rel    0
CSTACK               DATA          00000080 - 0000009F             20  dse    0
RSTACK               DATA          000000A0 - 000000BF             20  dse    0

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

 1 518 bytes of CODE memory (16 range fill)
   104 bytes of DATA memory

Errors: none
Warnings: none
```

Listing 3: Library Build Results (Abbreviated)

Note The Embedded Workbench for Atmel AVR projects supplied in the Salvo for Atmel AVR and MegaAVR distributions contain additional help files in each project's Salvo Help Files group.

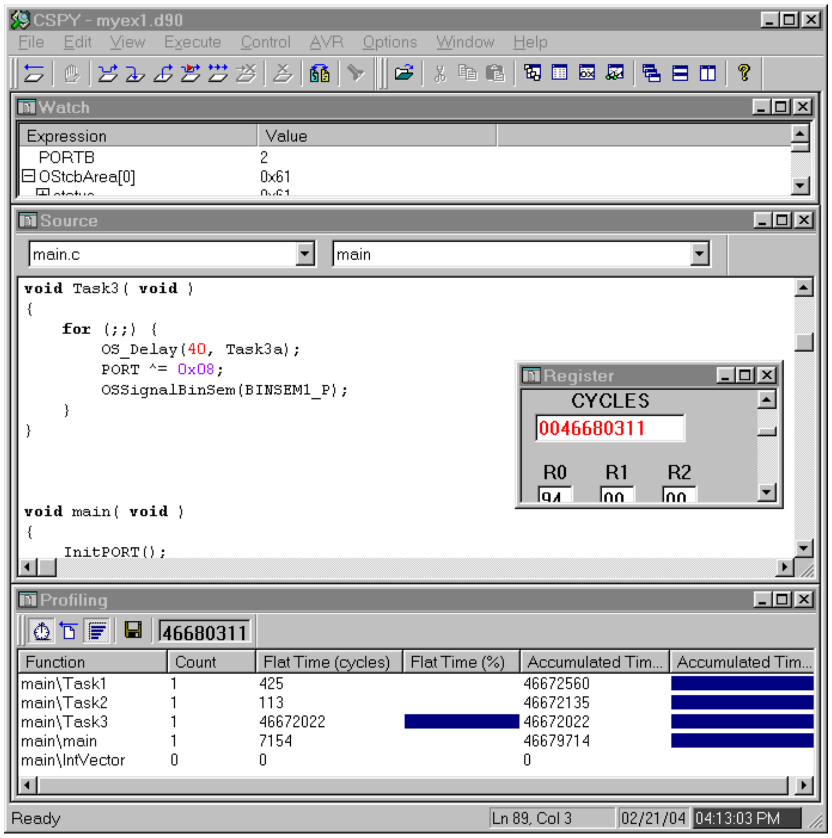
Tip If you configure Embedded Workbench to display the memory utilization for individual source files and the complete application you won't have to look in the map file. Select Options → Settings → Make Control → Message Filtering Level and choose All.

Testing the Application

C-SPY

You can test and debug this application using the C-SPY debugger and either the simulator or other debugging hardware. 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, cycle counting, etc.



Expression	Value
PORTB	2
⊟ OStcbArea[0]	0x61

```
void Task3( void )
{
    for (;;) {
        OS_Delay(40, Task3a);
        PORT ^= 0x08;
        OSSignalBinSem(BINSEM1_P);
    }
}


void main( void )
{
    InitPORT();
```

```
CYCLES
0046680311

R0    R1    R2
94    00    00
```

46680311

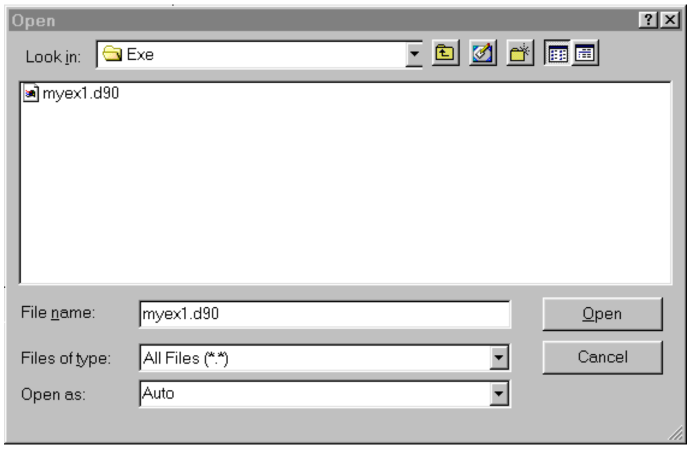
Function	Count	Flat Time (cycles)	Flat Time (%)	Accumulated Tim...	Accumulated Tim...
main\Task1	1	425		46672560	■
main\Task2	1	113		46672135	■
main\Task3	1	46672022	■	46672022	■
main\main	1	7154		46679714	■
main\IntVector	0	0		0	

Figure 17: 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.

AVR Studio

To test your Salvo application in AVR Studio,[6] under File → Open File …, navigate to your project's `Debug\Exe` subdirectory and open the object (`*.d90`) file:



Open

Look in: Exe

myex1.d90

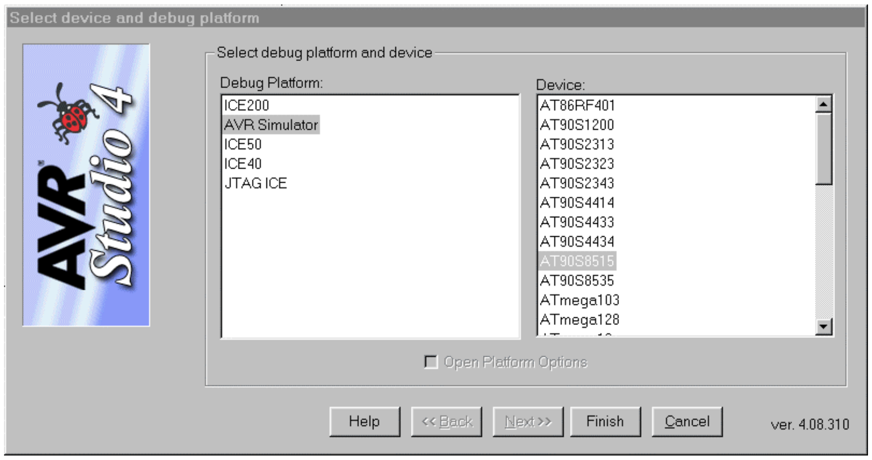
File name: myex1.d90

Files of type: All Files (*.*)

Open as: Auto

Figure 18: Opening the Object File

Select Open. AVR Studio will load the object file, then present another window wherein you can select a Debug Platform and a Device:



Select device and debug platform

Select debug platform and device

Debug Platform: ICE200, AVR Simulator, ICE50, ICE40, JTAG ICE

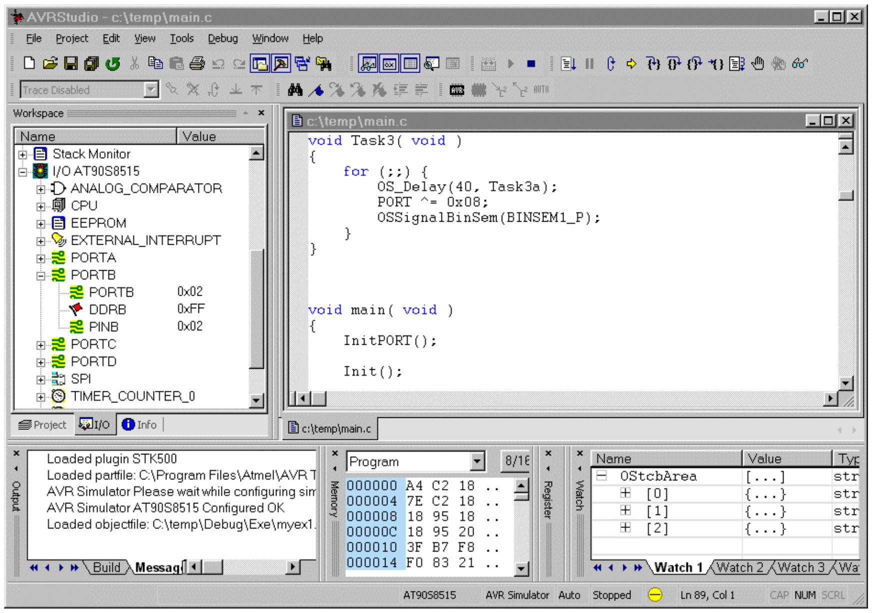
Device: AT86RF401, AT90S1200, AT90S2313, AT90S2323, AT90S2343, AT90S4414, AT90S4433, AT90S4434, AT90S8515, AT90S8535, ATmega103, ATmega128

Open Platform Options

ver. 4.08.310

Figure 19: Selecting the Debug Platform and Device

Select Finish. AVR Studio will configure the simulator, load the object file, and setup for source-level debugging at the start of your `main()`:



Name	Value
Stack Monitor	
I/O AT90S8515	
ANALOG_COMPARATOR	
CPU	
EEPROM	
EXTERNAL_INTERRUPT	
PORTA	
PORTB	
PORTB	0x02
DDRB	0xFF
PINB	0x02
PORTC	
PORTD	
SPI	
TIMER_COUNTER_0	

```
void Task3( void )
{
    for (;;) {
        OS_Delay(40, Task3a);
        PORT ^= 0x08;
        OSSignalBinSem(BINSEM1_P);
    }
}


void main( void )
{
    InitPORT();

    Init();
```

```
Loaded plugin STK500
Loaded partfile: C:\Program Files\Atmel\AVR T
AVR Simulator Please wait while configuring sim
AVR Simulator AT90S8515 Configured OK
Loaded objectfile: C:\temp\Debug\Exe\myex1.
```

```
000000 A4 C2 18 ..
000004 7E C2 18 ..
000008 18 95 18 ..
00000C 18 95 20 ..
000010 3F B7 F8 ..
000014 F0 83 21 ..
```

Name	Value	Typ
OStcbArea	[...]	str
[0]	{...}	str
[1]	{...}	str
[2]	{...}	str

Figure 20: Source-Level Debugging in AVR Studio

In addition to debugging in the AVR Studio simulator, you can also download your application to an Atmel STK500 development board by selecting Tools → STK500/… → STK500/…, and then programming the device by selecting Use Current Simulator/Emulator FLASH Memory,

Migrating to Embedded Workbench for Atmel AVR v3

Existing Salvo applications built as projects (`*.pjt`) under IAR's Embedded Workbench for Atmel AVR v2 can be migrated to v3 using the following steps.

- In Embedded Workbench for Atmel AVR v3, choose File → New → Workspace to create a new workspace file (`*.eww`).
- Choose Project → Add Existing Project …, select Files of type: Old Project Files (*.prj, *.pew), navigate to the old project and select Open, then OK.
- Under Project → Options, select the device (e.g. AT90S8515) under General → Target → Processor configuration, if necessary. Set the desired optimizations under ICCAVR → Optimizations.
- Remove the existing Salvo library from the project, and replace it with a same-named one from `\salvo\lib\iaravr-v3`.

When finished, the new project window will look like this:

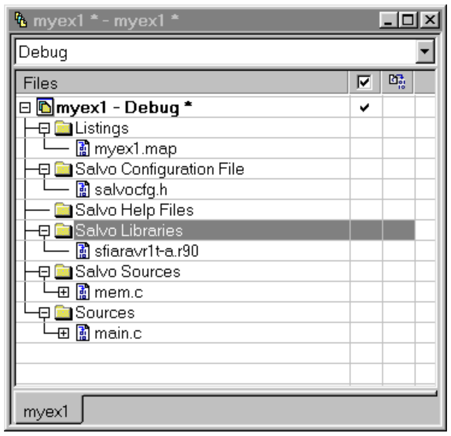
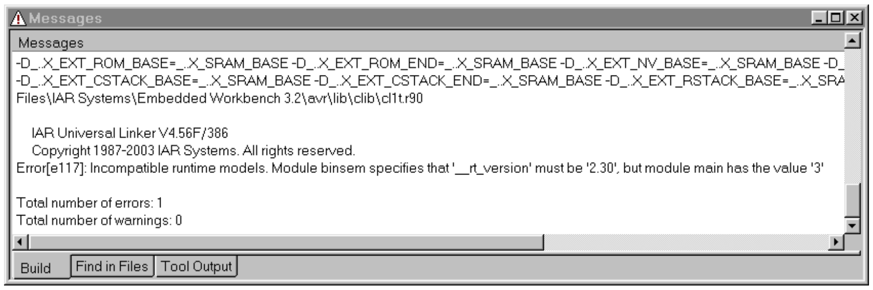


Figure 21: Project Window for a Library Build in Embedded Workbench for Atmel AVR v3

Troubleshooting

Linker Error: Incompatible runtime models

If you are doing a library build and your version of the IAR AVR C Compiler is different from that used to build the Salvo library you have added to your project, the linker will issue an error like this one:



```
Messages
-D_..X_EXT_ROM_BASE=_..X_SRAM_BASE -D_..X_EXT_ROM_END=_..X_SRAM_BASE -D_..X_EXT_NV_BASE=_..X_SRAM_BASE -D_
-D_..X_EXT_CSTACK_BASE=_..X_SRAM_BASE -D_..X_EXT_CSTACK_END=_..X_SRAM_BASE -D_..X_EXT_RSTACK_BASE=_..X_SRA
Files\IAR Systems\Embedded Workbench 3.2\avr\lib\clib\cl1t.r90

   IAR Universal Linker V4.56F/386
   Copyright 1987-2003 IAR Systems. All rights reserved.
Error[e117]: Incompatible runtime models. Module binsem specifies that '__rt_version' must be '2.30', but module main has the value '3'

Total number of errors: 1
Total number of warnings: 0
```

Figure 22: Linker Error due to Version Mismatch (EW v3)

This type of error occurs because the Salvo library – in this case, `salvo\lib\iaravr-v2\sliaravr1t-a.r90` – was built with v2 of the IAR AVR C Compiler, which defines a symbol called `__rt_version` as 2.30, yet the project was built in EW v3, which expects `__rt_version` to be 3. The solution is simply to use the Salvo library appropriate for the version of the IAR AVR C compiler you are using – in this case, `salvo\lib\iaravr-v3\sliaravr1t-a.r90`.

Note `__rt_version` is used by XLINK at link time to ensure consistency between modules (C, Embedded C++ or assembler). See the *IAR AVR C/EC++ Compiler Reference Guide* for more information.

Note As long as the major version number of the IAR AVR C Compiler you're using (and therefore `__rt_version`'s value) matches that used to generate the Salvo libraries, you will be able to link to the precompiled Salvo libraries that are supplied in each Salvo for Atmel AVR and MegaAVR distribution. E.g. v3.x can be used with Salvo libraries built with v3.10A.[7]

`__rt_version` mismatches like this will occur whenever Salvo users and the Salvo for Atmel AVR and MegaAVR distribution are at substantially different versions of the IAR AVR C compiler. Therefore, we *strongly recommend that Salvo users keep their IAR AVR C compiler up-to-date* to avoid any potential difficulties. Salvo Pro users can avoid any potential problems by simply re-building the Salvo libraries with their own version of the IAR AVR C compiler.

This type of linker error will not happen with source code builds, e.g. when using Salvo Pro to build an application using the Salvo source files as project nodes instead of linking to a Salvo library.

Application Crashes After Changing Processor Type

Remember to `#include` the appropriate header file for your AVR variant (see *Building the Project*, above). A common cause for such crashes is a difference in interrupt vector locations or definitions between two members of a processor family. Mainline code may work correctly, but the application will crash if interrupt vectors are not in the right locations.

Example Projects

Example projects for IAR's AVR C compiler can be found in the `\salvo\tut\tu1-6\sysac` directories. The include path for each of these projects includes `\salvo\tut\tu1\sysac`, and each project defines the `SYSAC` symbol.

Complete Salvo Lite library-build projects are contained in the project files `\salvo\tut\tu1-6\sysac\tu1-6lite.*`. These projects also define the `MAKE_WITH_FREE_LIB` symbol.

Complete Salvo LE library-build projects are contained in the project files `\salvo\tut\tu1-6\sysac\tu1-6le.*`. These projects also define the `MAKE_WITH_STD_LIB` symbol.

Complete Salvo Pro library-build projects are contained in the project files `\salvo\tut\tu1-6\sysac\tu1-6prolib.*`. These projects also define the `MAKE_WITH_STD_LIB` symbol.

Complete Salvo Pro source-code-build projects are contained in the project files `\salvo\tut\tu1-6\sysac\tu1-6pro.*`. These projects also define the `MAKE_WITH_SOURCE` symbol.

Note Tutorial and example projects are provided for IAR Embedded Workbench for Atmel AVR v2 (`*.prj` files) and IAR Embedded Workbench for Atmel AVR v3 (`*.ewp` & `*.eww` files).

[1] This Salvo project supports a wide variety of targets and compilers. For use with IAR's AVR C compiler, it requires the `SYSAC` 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.

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

[3] This Salvo Lite library contains all of Salvo's basic functionality. The corresponding Salvo LE and Pro libraries are sliaravr1t-a.r90 and sliaravr1tia.r90, respectively.

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

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

[6] AVRStudio v4.08 is shown.

[7] In this example, the major version number is 3.