PUMPKIN™
REAL-TIME SOFTWARE

RM-EPS1C17

Reference Manual

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

Salvo Compiler Reference Manual – Seiko Epson S1C17

Salvo™

The RTOS that runs in tiny places.™

© Copyright 2008 Pumpkin, Inc. last updated on Feb 29, 2008
All trademarks mentioned herein are properties of their respective companies.

Introduction

This manual is intended for Salvo 4 users who are targeting Seiko Epson S1C17xxx 16-bit microcontrollers with Seiko Epson's (http://www.epson.co.jp/e/) S5U1C17001C C compiler v1.1 and later (hereafter referred to as "Seiko Epson's C Compiler Package for S1C17 Family").

Note Seiko Epson's C Compiler Package for S1C17 Family is normally used with the GNU17 IDE running under Eclipse. All of this software is available directly from Seiko Epson.

Related Documents

The following Salvo documents should be used in conjunction with this manual when building Salvo applications with Seiko Epson's C Compiler Package for S1C17 Family:

- *Salvo User Manual*

Example Projects

Example Salvo projects for use with Seiko Epson's C Compiler Package for S1C17 Family and the ImageCraft IDE can be found in the:

```
\Pumpkin\Salvo\Example\S1C17\
```

directories of every Salvo for S1C17 Family distribution.

Tip These example projects can be easily modified for any device in the S1C17 family.

Features

Table 1 illustrates important features of Salvo's port to Seiko Epson's C Compiler Package for S1C17 Family.

General	
Abbreviated as	`EPS1C17`
Available distributions	Salvo Lite, LE & Pro for S1C17 Family
Supported targets	entire S1C17 family
Header file(s)	`salvoporteps1c17.h`
Other target-specific file(s)	`salvoporteps1c17-small.s`, `salvoporteps1c17-middle.s`, `salvoporteps1c17-regular.s`
salvocfg.h	
Compiler auto-detected?	yes[1]
Include target-specific header file in `salvocfg.h`?	yes
libraries	
Located in	`Lib\EPS1C17`
Context Switching	
Method	function-based via `OSDispatch()` & `OSCtxSw()`
Labels required?	no
Size of auto variables and function parameters in tasks	total size must not exceed 255 8-bit bytes
Memory & Registers	
R4-R7 (call-saved registers)	saved in each tcb, 32 bits each
Interrupts	
Interrupt latency in context switcher	0 cycles
Interrupts in critical sections controlled via	user hooks
Default behavior in critical sections	see example user hooks
Debugging	
Source-level debugging with Pro library builds?	yes
Compiler	
Bitfield packing support?	no
printf() / %p support?	yes / yes
va_arg() support?	yes

Table 1: Features of Salvo port to Seiko Epson's C Compiler Package for S1C17 Family

Libraries

Nomenclature

The Salvo libraries for Seiko Epson's C Compiler Package for S1C17 Family follow the naming convention shown in Figure 1.

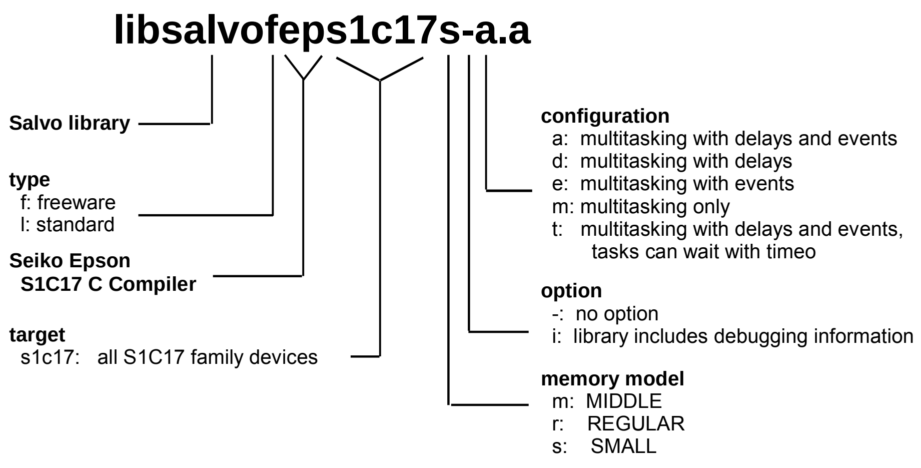

libsalvofeps1c17s-a.a
Salvo library
type
f: freeware
l: standard
Seiko Epson
S1C17 C Compiler
target
s1c17: all S1C17 family devices
configuration
a: multitasking with delays and events
d: multitasking with delays
e: multitasking with events
m: multitasking only
t: multitasking with delays and events, tasks can wait with timeo
option
-: no option
i: library includes debugging information
memory model
m: MIDDLE
r: REGULAR
s: SMALL

Figure 1: Salvo Library Nomenclature – Seiko Epson's C Compiler Package for S1C17 Family

Type

Salvo Lite distributions contain *freeware* libraries. All other Salvo distributions contain *standard* libraries. See the *Libraries* chapter of the *Salvo User Manual* for more information on library types.

Target

Each library works on all members of the S1C17 family.

Memory Model

Seiko Epson's C Compiler Package for S1C17 Family supports three distinct memory models – SMALL, MIDDLE and REGULAR. When doing a library build, the Salvo library's memory model must match that of the project.

Note A project's `salvocfg.h` configuration file does not affect the memory model in use. Therefore when building applications with Salvo libraries you must ensure that the Salvo library used corresponds to the project's memory model.

Option

Salvo Pro users can select between two sets of libraries – standard libraries, and standard libraries incorporating source-level debugging information. The latter have been built with Seiko Epson's C Compiler Package for S1C17 Family's `+gstabs` command-line option. This adds source-level debugging information to the libraries, making them ideal for source-level debugging and stepping in the GNU17 IDE. To use these libraries, simply select one that includes the debugging information (e.g. `libsalvolieps1c17rit.a`) instead of one without (e.g. `libsalvolieps1c17r-t.a`) in your GNU17 project.

Configuration

Different library configurations are provided for different Salvo distributions and to enable the user to minimize the Salvo kernel's footprint. See the *Libraries* chapter of the *Salvo User Manual* for more information on library configurations.

Build Settings

Salvo's libraries for Seiko Epson's C Compiler Package for S1C17 Family are built using the default settings outlined in the *Libraries* chapter of the *Salvo User Manual*. Target-specific settings and overrides are listed in Table 2.

Target-specific Settings	
Delay sizes	8 bits
Idling hook	dummy, can be overridden
Interrupt hook	disables then restores PSR's I bit, can be overridden
Watchdog hook	dummy, can be overridden
System tick counter	available, 32 bits
Task priorities	enabled

Table 2: Build settings and overrides for Salvo libraries for Seiko Epson's C Compiler Package for S1C17 Family

Note Salvo Lite libraries have precompiled limits for the number of supported tasks, events, etc. Salvo LE and Pro libraries have no such limits. See the *Libraries* chapter of the *Salvo User Manual* for more information.

Available Libraries

There are a total of 33 Salvo libraries for Seiko Epson's C Compiler Package for S1C17 Family. Each Salvo for S1C17 Family distribution contains the Salvo libraries of the lesser distributions beneath it.

Target-Specific Salvo Source Files

Depending on the memory model chosen, one of three different context-switcher files is required for Salvo Pro source-code builds, as shown in Table 3:

Context-switcher Filename	Application
`salvoporteps1c17-small.s`	small memory model (64KB)
`salvoporteps1c17-middle.s`	middle memory model (1MB)
`salvoporteps1c17-regular.s`	regular memory model (16MB)

Table 3: Target-specific context-switcher files for Seiko Epson's C Compiler Package for S1C17 Family

These context-switching files vary in the manner in which function pointers are employed for task vectoring, as well as in the extended instructions used.

salvocfg.h Examples

Below are examples of `salvocfg.h` project configuration files for different Salvo for S1C17 Family distributions targeting any device in the S1C17 family.

Salvo Lite Library Build

```
#define OSUSE_LIBRARY              TRUE
#define OSLIBRARY_TYPE             OSF
#define OSLIBRARY_CONFIG           OST
#define OSTASKS                    2
#define OSEVENTS                   4
#define OSEVENT_FLAGS              0
#define OSMESSAGE_QUEUES           1
```

Listing 1: Example salvocfg.h for library build using libsalvofeps1c17s-t.a

Salvo LE & Pro Library Build

```
#define OSUSE_LIBRARY              TRUE
#define OSLIBRARY_TYPE             OSL
#define OSLIBRARY_CONFIG           OST
#define OSTASKS                    7
#define OSEVENTS                   13
#define OSEVENT_FLAGS              3
#define OSMESSAGE_QUEUES           2
```

Listing 2: Example salvocfg.h for library build using libsalvoleps1c17r-t.a or libsalvoleps1c17m-t.a or libsalvoleps1c17s-t.a

Salvo Pro Source-Code Build

```
#define OSENABLE_IDLING_HOOK     TRUE
#define OSENABLE_SEMAPHORES      TRUE
#define OSTASKS                  9
#define OSEVENTS                 17
#define OSEVENT_FLAGS            2
#define OSMESSAGE_QUEUES         4
```

Listing 3: Example salvocfg.h for source-code build

Interrupt Latencies

Since Salvo's context switcher for Seiko Epson's C Compiler Package for S1C17 Family does not need to control interrupts, Salvo applications can easily be created with zero total interrupt latency for interrupts of interest.

In a properly-configured application, only those interrupts that call Salvo services will (by necessity) experience interrupt latency from Salvo's operations. Users must ensure that these interrupt sources are disabled (and re-enabled) via the user interrupt hooks.

Disabling and re-enabling interrupts globally in the user interrupt hooks (i.e., the default user interrupt hook behavior) is of course permitted, but will result in non-zero interrupt latencies for all interrupt sources, even those that do not call Salvo services. See the target-specific source files of this distribution for examples.

Memory Usage

Example Application[2]	Program Memory Usage[3]	Data Memory Usage[4]
`\S1C17\…\tut5lite`	???	??
`\S1C17\…\tut5le`	???	??
`\S1C17\…\tut5pro` `(SMALL memory model)`	1692	114
`\S1C17\…\tut5pro` `(REGULAR memory model)`	2056	166

Table 4: ROM and RAM requirements for Salvo applications built with Seiko Epson's C Compiler Package for S1C17 Family

User Hooks

Overriding Default Hooks

In library builds, users can define new hook functions in their projects and the linker will choose the user function(s) over the default function(s) contained in the Salvo library.

In source-code builds, users can remove the default hook file(s) from the project and substitute their own hook functions.

Idling

The default idling hook in `salvohook_idle.c` is a dummy function, as shown below.

```
void OSIdlingHook ( void )
{
  ;
}
```

Listing 4: Default Salvo idling hook for Seiko Epson's C Compiler Package for S1C17 Family

Users can replace it (e.g. with a directive to put the S1C17 to sleep) by building their own version with their application.

Interrupt

The default interrupt hooks in `salvohook_interrupt.c` are shown below.[5]

```
void OSDisableHook ( void )
{
  asm("di;");
}


void OSEnableHook ( void )
{
  asm("ei;");
}
```

Listing 5: Default Salvo interrupt hooks for Seiko Epson's C Compiler Package for S1C17 Family

These functions clear the `I` bit (i.e. disable global interrupts) in the PSR across Salvo's critical section. The default interrupt hooks are suitable for any application that calls only `OSTimer()` from an ISR.

Note The S1C17 architecture and instruction set do not provide a direct means of reading or writing the PSR's I (global interrupt enable) bit.

The astute reader will recognize that the default hooks will cause interrupts to be re-enabled inside an ISR (i.e. before exiting from the ISR via a `reti` instruction) when a Salvo API service that controls interrupts (e.g. `OSSignalBinSem()`) is called from the foreground / ISR level.[6] This may or may not cause problems in your application. Two possible solutions are presented below.

The most expedient solution is to create your own interrupt hooks to selectively disable and re-enable only those (peripheral) sources of interrupts whose associated ISRs call Salvo services. For example, if incoming UART characters signal to Salvo via a call to `OSSignalBinsem()` that the input stream must be processed by a task, then the interrupt hooks should be configured as

```
void OSDisableHook ( void )
{
  UART_CTL &= ~RIEN; // suppress Rx ints
}


void OSEnableHook ( void )
{
  UART_CTL |=  RIEN; // allow Rx ints
}
```

Listing 6: Example Salvo interrupt hooks (pseudocode) for Seiko Epson's C Compiler Package for S1C17 Family when receiving a character leads to a Salvo API call

Note The hooks in Listing 6 assume that interrupt are enabled globally at all times, and that only the receive character ISR calls a Salvo API service.

This approach avoids any direct manipulation of the PSR's I bit, thereby leaving the S1C17 to automatically disable global interrupts upon servicing an interrupt, and restoring them thereafter to their pre-interrupt state.

Also, this approach (disabling only those peripheral interrupt sources that call Salvo API services from the foreground / interrupt level) is the highest-performance approach, as it minimizes interrupt latency and guarantees that all interrupt sources that are not associated with Salvo API service calls will have no interrupt latency due to Salvo.

Tip You can write your application's interrupt hooks to disable and enable as many peripheral interrupt enable bits as required by adding additional code to `OSDisableHook()` and `OSEnableHook()`.

An alternative (and lower-performance) approach is to augment the default interrupt hooks so that interrupts are only re-enabled in `OSEnableHook()` when `OSEnableHook()` is called from the background (i.e. non-ISR) level. This would require additional user code at the entry and exit of each ISR that calls a Salvo service. Said user code would set a flag at the start of the ISR and clear the flag at the end of the ISR. `OSEnableHook()` would be configured to read this flag, and would only re-enable global interrupts when this flag was found to be cleared. The net effect would be that when a Salvo service was called from an ISR, it would not re-enable interrupts at the end of its critical section(s).

Warning Not disabling all source of interrupts that call Salvo services during critical sections will cause the Salvo application to fail.

Watchdog

The default watchdog hook in `salvohook_wdt.c` is a dummy function, is shown below.

```
void OSClrWDTHook ( void )
{
   ;
}
```

Listing 7: Default Salvo watchdog hook for Seiko Epson's C Compiler Package for S1C17 Family

Users can replace it (e.g. with a directive to put clear the S1C17's watchdog timer) by building their own version with their application. Clearing the watchdog timer from within `OSClrWDTHook()` will clear the watchdog timer every time Salvo's scheduler `OSSched()` is called.

Warning Salvo's clearing of the watchdog timer via `OSClrWDTHook()` is only a basic means of using a watchdog timer and is *not* a robust solution for production. It is, however, a reasonable starting point for developing a robust watchdog timer scheme.

Compiler Issues

Incompatible Optimizations

There are no known incompatibilities between Seiko Epson's C Compiler Package for S1C17 Family's optimizations (e.g. `-O1`) and Salvo. The context switcher saves and restores registers R4 through R7 where necessary.

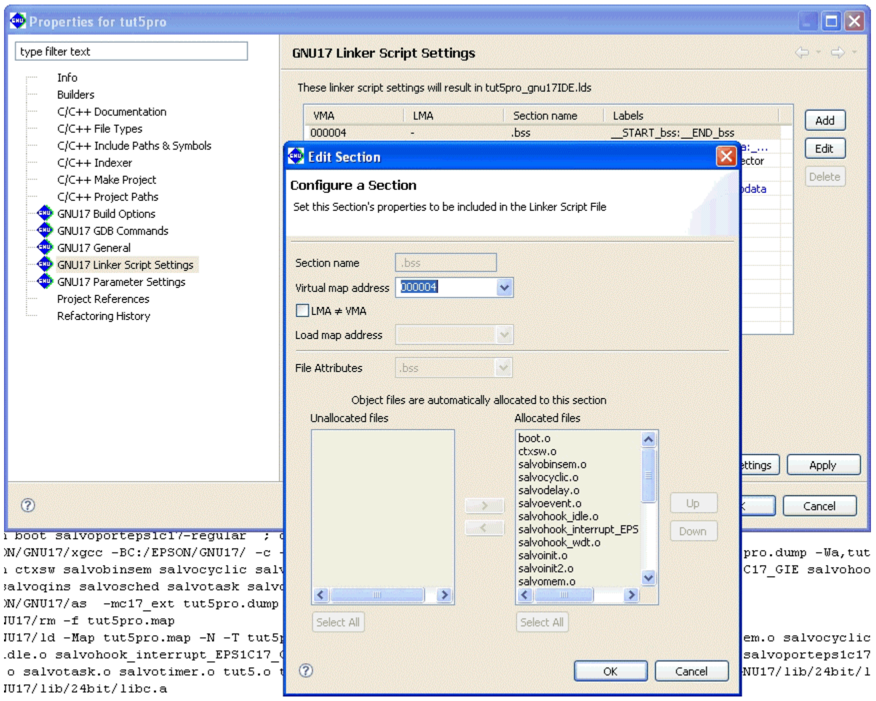
Special Considerations

Objects at RAM Address 0

The S1C17's memory map normally includes RAM at address 0. Locating any of Salvo's global objects at 0 is likely to cause problems when Salvo references that object (due to null pointer references).[7] Therefore you should adjust the allowable RAM memory range to start at 4 instead of 0. This will guarantee that no object is placed at RAM memory address 0.

Warning The start address for the `.bss` (RAM objects) section must be a multiple of 4. Odd addresses or multiples of 2 and not 4 will cause runtime errors.

To do so in the GNU17 IDE, select Project → Properties → GNU17 Linker Script Settings → (Section name) `.bss`, and click on Edit. Set the Virtual map address to 4 and click on OK. Then select Project → Clean, then Project → Build All.



Properties for tut5pro

GNU17 Linker Script Settings

These linker script settings will result in tut5pro_gnu17IDE.lds

VMA	LMA	Section name	Labels
000004	-	.bss	__START_bss:__END_bss

Edit Section

Configure a Section

Set this Section's properties to be included in the Linker Script File

Section name: .bss
Virtual map address: 000004
LMA ≠ VMA
Load map address:
File Attributes: .bss

Object files are automatically allocated to this section

Allocated files: boot.o, ctxsw.o, salvobinsem.o, salvocyclic.o, salvodelay.o, salvoevent.o, salvohook_idle.o, salvohook_interrupt_EPS, salvohook_wdt.o, salvoinit.o, salvoinit2.o, salvomem.o

```
ı boot salvoportepslc17-regular  ; c
)N/GNU17/xgcc -BC:/EPSON/GNU17/ -c -                    pro.dump -Wa,tut
ı ctxsw salvobinsem salvocyclic salv                    C17_GIE salvohoo
ıalvoqins salvosched salvotask salvo
)N/GNU17/as  -mc17_ext tut5pro.dump
JU17/rm -f tut5pro.map
JU17/ld -Map tut5pro.map -N -T tut5p                    em.o salvocyclic
.dle.o salvohook_interrupt_EPS1C17_C                    salvoportepslc17
 o salvotask.o salvotimer.o tut5.o t                    NU17/lib/24bit/l
JU17/lib/24bit/libc.a
```

Figure 2: Setting the .bss section to begin at RAM address 0x000004 – GNU17 IDE for Seiko Epson's C Compiler Package for S1C17 Family

You can verify the results by opening the project's map file and finding the value of the `.bss` section (should be 0x00000004).

Project Configuration

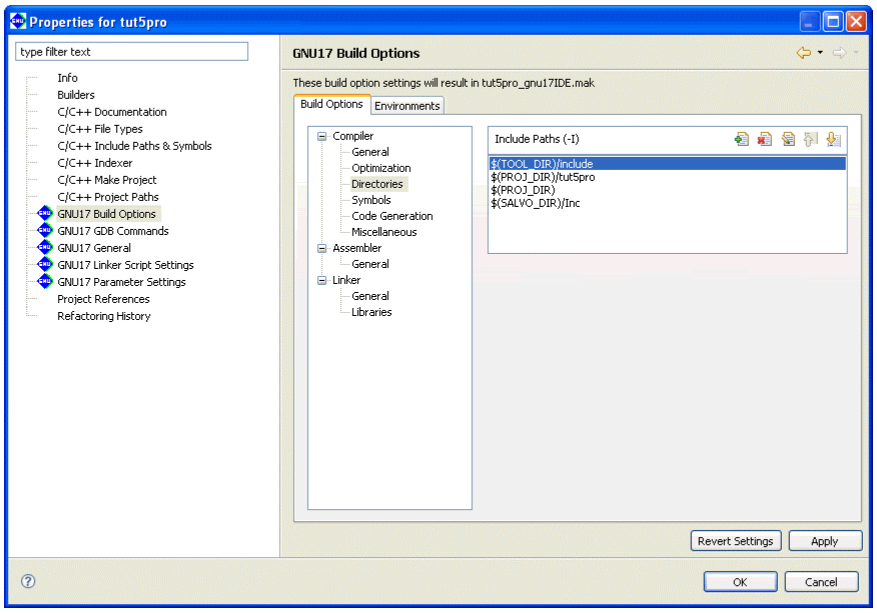
Include Paths

At a minimum, every Salvo project requires include paths to:

- The project's `salvocfg.h` configuration file
- `salvo.h` and other Salvo header files

In the GNU17 IDE, these are configured under the GNU17 Build Options properties of your project.

To add an include path, select Project → Properties → GNU17 Build Options → Build Options → Directories, and click on the Add icon. Add include paths to both the directory that holds your project's unique `salvocfg.h`, and also to Salvo's header file directory (usually `C:/Pumpkin/Salvo/Inc`).[8]



These build option settings will result in tut5pro_gnu17IDE.mak

```
$(TOOL_DIR)/include
$(PROJ_DIR)/tut5pro
$(PROJ_DIR)
$(SALVO_DIR)/Inc
```

Figure 3: Setting the Include Paths – GNU17 IDE for Seiko Epson's C Compiler Package for S1C17 Family

Tip For portability and other reasons, you may find it advantageous to use environment settings in setting include paths. To do so, select Project → Properties → GNU17 Build Options → Build Options → Environments, and click on New. For example, you could create an environment setting named `SALVO_DIR` and assign it a value of `C:/Pumpkin/Salvo`. Then, add an include path of `$(SALVO_DIR)/Inc` instead of `C:/Pumpkin/Salvo/Inc`.

Note Your project is likely to require other, non-Salvo include paths as well. These can be set using the same procedures as outlined above.

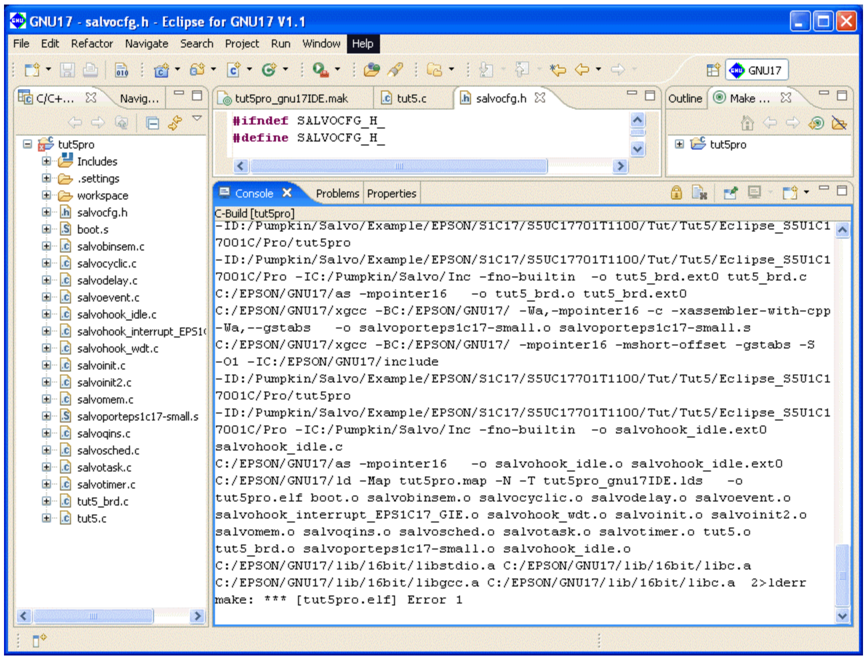
Debugging

Compiler Errors

Generally speaking, the Salvo code and user code calling the Salvo API will compile successfully as long you are using the Salvo API correctly and your `salvocfg.h` configuration file is correct for the type of build (Salvo Pro source-code build or Salvo Lite / LE / Pro library build) you are doing.

Linker Errors

In the GNU17 IDE, link-time errors are reported rather cryptically, as illustrated below:



```
#ifndef SALVOCFG_H_
#define SALVOCFG_H_
```

```
C-Build [tut5pro]
-ID:/Pumpkin/Salvo/Example/EPSON/S1C17/S5UC17701T1100/Tut/Tut5/Eclipse_S5U1C1
7001C/Pro/tut5pro
-ID:/Pumpkin/Salvo/Example/EPSON/S1C17/S5UC17701T1100/Tut/Tut5/Eclipse_S5U1C1
7001C/Pro -IC:/Pumpkin/Salvo/Inc -fno-builtin  -o tut5_brd.ext0 tut5_brd.c
C:/EPSON/GNU17/as -mpointer16   -o tut5_brd.o tut5_brd.ext0
C:/EPSON/GNU17/xgcc -BC:/EPSON/GNU17/ -Wa,-mpointer16 -c -xassembler-with-cpp
-Wa,--gstabs   -o salvoporteps1c17-small.o salvoporteps1c17-small.s
C:/EPSON/GNU17/xgcc -BC:/EPSON/GNU17/ -mpointer16 -mshort-offset -gstabs -S
-O1 -IC:/EPSON/GNU17/include
-ID:/Pumpkin/Salvo/Example/EPSON/S1C17/S5UC17701T1100/Tut/Tut5/Eclipse_S5U1C1
7001C/Pro/tut5pro
-ID:/Pumpkin/Salvo/Example/EPSON/S1C17/S5UC17701T1100/Tut/Tut5/Eclipse_S5U1C1
7001C/Pro -IC:/Pumpkin/Salvo/Inc -fno-builtin  -o salvohook_idle.ext0
salvohook_idle.c
C:/EPSON/GNU17/as -mpointer16   -o salvohook_idle.o salvohook_idle.ext0
C:/EPSON/GNU17/ld -Map tut5pro.map -N -T tut5pro_gnu17IDE.lds   -o
tut5pro.elf boot.o salvobinsem.o salvocyclic.o salvodelay.o salvoevent.o
salvohook_interrupt_EPS1C17_GIE.o salvohook_wdt.o salvoinit.o salvoinit2.o
salvomem.o salvoqins.o salvosched.o salvotask.o salvotimer.o tut5.o
tut5_brd.o salvoporteps1c17-small.o salvohook_idle.o
C:/EPSON/GNU17/lib/16bit/libstdio.a C:/EPSON/GNU17/lib/16bit/libc.a
C:/EPSON/GNU17/lib/16bit/libgcc.a C:/EPSON/GNU17/lib/16bit/libc.a  2>lderr
make: *** [tut5pro.elf] Error 1
```

Figure 4: Example link-time error – GNU17 IDE for Seiko Epson's C Compiler Package for S1C17 Family

The best way to resolve these sorts of errors is to view the project's `lderr` file – this will rapidly point you in the direction of a solution:

Undefined References to Salvo Objects

When building a Salvo application, failure to include Salvo's `salvomem.c` in your project will lead to a slew of undefined references to Salvo objects in the `lderr` file:

```
salvobinsem.o: In function `OSSignalBinSem':
salvobinsem.c:258: undefined reference to `OSsigQoutP'
salvobinsem.c:258: undefined reference to `OSsigQoutP'
…
salvobinsem.c:258: undefined reference to `OSsigQinP'
salvobinsem.c:258: undefined reference to `OSsigQinP'
…
salvodelay.o: In function `OSDelay':
```

```
salvodelay.c:111: undefined reference to `OScTcbP'
salvodelay.c:111: undefined reference to `OScTcbP'
…
salvodelay.c:163: undefined reference to `OStimerTicks'
salvodelay.c:163: undefined reference to `OStimerTicks'
…
salvodelay.o:salvodelay.c:215: more undefined references
to `OScTcbP' follow
salvoinit.o: In function `OSInit':
salvoinit.c:48: undefined reference to `OSeligQP'
salvoinit.c:48: undefined reference to `OSeligQP'
salvoinit.c:49: undefined reference to `OScTcbP'
salvoinit.c:49: undefined reference to `OScTcbP'
salvoinit.c:53: undefined reference to `OSdelayQP'
salvoinit.c:53: undefined reference to `OSdelayQP'
salvoinit.c:57: undefined reference to `OSlostTicks'
salvoinit.c:57: undefined reference to `OSlostTicks'
…
tut5.o: In function `main':
tut5.c:136: undefined reference to `OSecbArea'
tut5.c:136: undefined reference to `OSecbArea'
```

Listing 8: lderr contents after failing to include salvomem.c – GNU17 IDE for Seiko Epson's C Compiler Package for S1C17 Family

In this example, `salvomem.c` was not included in the project, and therefore the linker could not find any of the Salvo objects that are referenced throughout the Salvo code. Adding `salvomem.c` to the project resulted in a successful build.

Undefined References to Salvo Functions

When building a Salvo application, failure to include a required Salvo library (Salvo Lite / LE / Pro library builds) or a Salvo source file (Salvo Pro source-code builds) in your project will lead to undefined references to Salvo services in the `lderr` file:

```
salvoevent.o: In function `OSWaitEvent':
salvoevent.c:1325: undefined reference to `OSDelDelayQ'
salvoevent.c:1325: undefined reference to `OSDelDelayQ'
salvosched.o: In function `OSSched':
salvosched.c:369: undefined reference to `OSDelPrioQ'
salvosched.c:369: undefined reference to `OSDelPrioQ'
salvosched.c:370: undefined reference to `OSDelPrioQ'
salvosched.c:370: undefined reference to `OSDelPrioQ'
salvosched.c:371: undefined reference to `OSDelPrioQ'
salvosched.o:salvosched.c:371: more undefined references
  to `OSDelPrioQ' follow
```

Listing 9: lderr contents after failing to include one of Salvo's source files in a Salvo Pro source-code build – GNU17 IDE for Seiko Epson's C Compiler Package for S1C17 Family

In this Salvo Pro source-code-build example, the two functions `OSDelDelayQ()` and `OSDelPrioQ()` could not be found. A review of the Salvo User Manual or a `grep` on the Salvo source-code directory reveals that these functions are located in `salvoqdel.c`. Adding `salvoqdel.c` to the project resulted in a successful build.

---

[1] This is done automatically through the `_c17` symbol defined by the compiler.

[2] Salvo 4.1.2-rc0 with v1.1 compiler.

[3] In bytes. Entire application, including `.text` section. Does not include `.vector`, `.rodata` or other sections.

[4] In bytes. Entire application, including `.bss` section. Does not include RAM reserved for the hardware stack.

[5] This hook is valid for all S1C17 targets.

[6] `OSTimer()` does not control interrupts because it should never be called from more than one location in user code.

[7] For example, if Salvo's tcb array starts at address 0, the first tcb has a handle of 0, which leads to problems when adding that tcb to any queue … the queueing algorithm reads the zero handle as there being no element in the queue at that position.

[8] All of Salvo's non-target-specific header files reside in this directory. Target-specific header files reside in subdirectories, e.g. in `./EPS1C17`.