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

RM-PICC
Reference Manual

Salvo Compiler Reference Manual – HI-TECH PICC

Salvo™
The RTOS that runs in tiny places.™

Introduction

This manual is intended for Salvo users who are targeting Microchip (http://www.microchip.com/) PIC12|1400|16|17 PICmicro® MCUs with HI-TECH's (http://www.htsoft.com/) PICC C compiler.

Related Documents

The following Salvo documents should be used in conjunction with this manual when building Salvo applications with HI-TECH's PICC C compiler:

Salvo User Manual
Application Note AN-1 (obsolete)
Application Note AN-3
Application Note AN-4 (obsolete)
Application Note AN-9
Application Note AN-17
Application Note AN-26

Note PICC users are strongly advised to upgrade to Microchip's MPLAB IDE v6.30. Use *AN-26* in place of *AN-1* and *AN-4*.

Example Projects

Example Salvo projects for use with HI-TECH's PICC C compiler and the Microchip MPLAB IDEs v5 and v6 can be found in the:

```
\salvo\ex\ex1\sysa
\salvo\tut\tu1\sysa
\salvo\tut\tu2\sysa
\salvo\tut\tu3\sysa
\salvo\tut\tu4\sysa
\salvo\tut\tu5\sysa
\salvo\tut\tu6\sysa

\salvo\ex\ex1\sysh
\salvo\tut\tu1\sysh
\salvo\tut\tu2\sysh
\salvo\tut\tu3\sysh
\salvo\tut\tu4\sysh
\salvo\tut\tu5\sysh
\salvo\tut\tu6\sysh
```

directories of every Salvo for Microchip PICmicro® MCUs distribution.[1]

Features

Table 1 illustrates important features of Salvo's port to HI-TECH's PICC C compiler.

general	
available distributions	Salvo Lite, LE & Pro for Microchip PICmicro® MCUs
supported targets	PIC12|1400|16|17 PICmicro® MCUs
header file(s)	`portpicc.h`
other target-specific file(s)	--
project subdirectory name(s)	SYSA, SYSH
salvocfg.h	
target-specific header file required?	no
compiler auto-detected?	yes[2]
context switching	
method	label-based via `OSCtxSw(label)`
`_OSLabel()` required?	yes
size of auto variables and function parameters in tasks	unrestricted
interrupts	
controlled via	`GIE` / `GLINTD` bit
interrupt status preserved in critical sections?	no
method used	interrupts disabled on entry and enabled on exit of critical sections
nesting limit	no nesting permitted
alternate methods possible?	yes[3]
debugging	
source-level debugging?	only in source-code builds
compiler	
bitfield packing support?	yes
printf() / %p support?	yes / no
va_arg() support?	yes

Table 1: Features of Salvo Port to HI-TECH's PICC C Compiler

Compiler Optimizations

Incompatible Optimizations

No optimizations in PICC v8.02 or higher are known to be incompatible with Salvo.

Libraries

Nomenclature

The Salvo libraries for HI-TECH's PICC C compiler follow the naming convention shown in Figure 1. It is similar to that used by HI-TECH for the standard PICC libraries.[4]

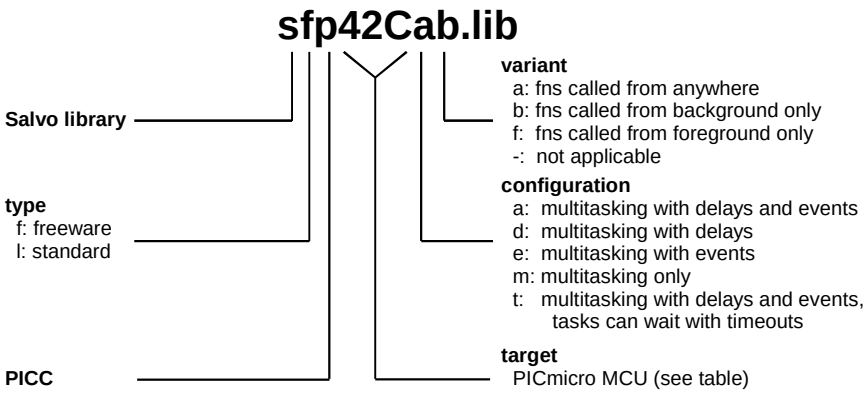

Figure 1: Salvo Library Nomenclature – HI-TECH's PICC C Compiler

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 is intended for one or more specific processors. Table 2 lists the correct library for each PICmicro® MCU.

target code	processor(s)
200:	12C508(A), 12CE518, 16C52, 16C(R)54(A, B, C), 16HV540, 16C55(A)
210:	16C(R)56(A)
211:	12C(R)509(A, AF, AG), 12CE519
212:	16C505
222:	16C(R)57(A, B, C), 16C(R)58(A, B)
401:	16C554(A), 16C556(A), 16C558(A), 16C(R)62(A, B), 16C620, 16C621, 16C622, 16C(R)64(A), 16C712, 16C715, 16C716, 16C(R)72(A)
40a:	16C61, 16C71, 16C710, 16C711, 16CR83, 16C(R)84, 16F83, 16F84(A)
40b:	12C671, 12C672, 12CE673, 12CE674, 16C432, 16C433, 16C620A, 16C621A, 16C622A, 16C641, 16C661, 16CE623, 16CE624, 16CE625
40c:	16C717, 16C770, 16C781, 16C782, 16F627, 16F628, 16F870, 16F871, 16F872
411:	14000, 16C(R)63(A), 16C(R)65(A, B), 16C73(A, B), 16(L)C74(A, B)
412:	16F73, 16F74, 16F873(A), 16F874(A)
41b:	16C642, 16C662
41c:	16C771, 16C773, 16C774, 16C923, 16C924, 16C925
42c:	16C66, 16C67, 16C745, 16C76, 16C765, 16C77, 16F76, 16F77, 16F876(A), 16F877(A)
700:	17C42(A)
710:	17C42(A) with external ROM @ 7000-7FFFh[5]
704:	17C(R)43, 17C44, 17C752, 17C762
714:	17C756(A), 17C766
926:	16C926

Table 2: Processors for Salvo Libraries – HI-TECH's PICC C Compiler

Verifying the Target Code

You can verify that you have chosen the right Salvo library by observing the PICC C compiler's actions. Open the project's `*.map` file and look towards the end of the Linker command line entry. There, you will see which PICC library was used to build your application. Use the same target code for your Salvo library.

For example, Listing 1 shows the linker command line entry for a PICC project built for the PIC16F877:

```
Linker command line:

-z -Mtu5lite.map -ol.obj \
  -
ppowerup=00h,intentry=04h,intcode,intret,init,init23,end_init,clrtext,stringtable,ps
trings,strings \
  -ABANK0=020h-07Fh -prbit_0=BANK0,rbss_0=BANK0,rdata_0=BANK0,idata_0=CODE \
  -ABANK1=0A0h-0EFh -prbit_1=BANK1,rbss_1=BANK1,rdata_1=BANK1,idata_1=CODE \
  -ABANK2=0110h-016Fh \
  -prbit_2=BANK2,rbss_2=BANK2,rdata_2=BANK2,idata_2=CODE \
  -ABANK3=0190h-01EFh \
  -prbit_3=BANK3,rbss_3=BANK3,rdata_3=BANK3,idata_3=CODE \
  -ACOMBANK=070h-07Fh -ptemp=COMBANK -ACODE=0-7FFhx4 -ACONST=0-0FFhx32 \
  -pconfig=2007h -pidloc=2000h -AEEDATA=2100h-21FFh -peeprom_data=EEDATA \
  -pfloat_text0=CODE,float_text1=CODE,float_text2=CODE \
  -pfloat_text3=CODE,float_text4=CODE \
  -pnvram=BANK0,nvram_1=BANK1,nvram_2=BANK2,nvram_3=BANK3 \
  -pnvbit_0=BANK0,nvbit_1=BANK1,nvbit_2=BANK2,nvbit_3=BANK3 -Q16F877 -W-9 \
  -h+tu5lite.sym -E -EC:\WINDOWS\TEMP\_3VVQHRP.AAA -ver=PICC#V8.02PL1 \
  C:\HT-PIC\LIB\picrt42c.obj C:\salvo\tut\tu1\isr.obj C:\salvo\src\mem.obj \
  C:\salvo\tut\tu5\main.obj C:\salvo\lib\htpicc\sfp42cab.lib \
  C:\HT-PIC\LIB\pic42c-c.lib

Object code version is 3.7

Machine type is 16F877
```

Listing 1: Example Linker Command Line for PIC16F877 (from *.map file)

In this case, the PICC C compiler is linking to its `pic42c-c.lib` library in order to build the application. Therefore the appropriate target code for the Salvo library is `42c`, e.g. `sfp42cab.lib`.[6]

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.

Variant

Because PICmicro® MCUs do not have a general-purpose stack, the Salvo source code must be properly configured via the appropriate configuration parameters. The Salvo libraries for HI-TECH's PICC C compiler are provided in different variants as shown in Table 3.

If your application does not call any Salvo services from within interrupts, use the *b* variant. If you wish to these services exclusively from within interrupts, use the *f* variant. If you wish to do this from both inside and outside of interrupts, use the *a* variant. In each case, you must call the services that you use from the correct place in your application, or either the linker will generate an error or your application will fail during runtime.

variant code	description
`a / OSA:`	Applicable services can be called from *a*nywhere, i.e. from the foreground and the background, simultaneously.
`b / OSB:`	Applicable services may only be called from the *b*ackground (default).
`f / OSF:`	Applicable services may only be called from the *f*oreground.
`- / OSNONE:`	Library has no variants.[7]

Table 3: Variants for Salvo Libraries – HI-TECH's PICC C Compiler

See the `OSCALL_OSXYZ` configuration parameters for more information on calling Salvo services from interrupts.

See *Multiple Callgraph Issues*, below, for more information on using library variants.

Build Settings

Salvo's libraries for HI-TECH's PICC C compiler 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 4. Differences between the PIC12 and other families are due to RAM constraints in the PIC12 series.

compiled limits		
	PIC12	PIC16, PIC17
max. number of tasks	3	3
max. number of events	4	5
max. number of event flags[8]	1	1
max. number of message queues[9]	0	1
target-specific settings		
	PIC12	PIC16, PIC17
delay sizes	8 bits	8 bits
idling hook	disabled	enabled
interrupt level[10]	0	0
Salvo objects[11,12]	bank1 persistent	bank1 persistent
system tick counter	not available	available, 32 bits
task priorities	disabled	enabled
watchdog timer	cleared in `OSSched()`.	cleared in `OSSched()`.

Table 4: Build Settings and Overrides for Salvo Libraries for HI-TECH's PICC C Compiler

Note Because the `persistent` bank qualifier is used to build these libraries, `OSInit()` *must* be used in all applications that use these libraries. Without it, Salvo's variables will be uninitialized, with unpredictable results.

Note PIC12, PIC16 and PIC17 Salvo libraries are configured for 8-bit message pointers (PICC's default pointer type). These pointers can point to any area of banked RAM, *but they cannot point to ROM*, i.e. they cannot be used with, say, `const char` objects. To point to ROM, a source-code build with alternate configuration options is required.

Note The compiled limits for tasks, events, etc. in Salvo libraries can be overridden to be less (all Salvo distributions) or more (all Salvo distributions except Salvo Lite) than the library default. See the *Libraries* chapter of the *Salvo User Manual* for more information.

Available Libraries

There are a total of 470 Salvo libraries for HI-TECH's PICC C compiler (50 `p2xx`/PIC12, 300 `p4xx`/PIC16 and 120 `p7xx`/PIC17). Each Salvo for Microchip PICmicro® MCUs distribution contains the Salvo libraries of the lesser distributions beneath it.

salvocfg.h Examples

Below are examples of `salvocfg.h` project configuration files for different Salvo for PICmicro® MCUs distributions targeting the PIC16F877.

Note When overriding the default number of tasks, events, etc. in a Salvo library build, `OSTASKS` and `OSEVENTS` (respectively) *must also be defined* in the project's `salvocfg.h`. If left undefined, the default values (see Table 4) will be used.

Salvo Lite Library Build

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

Listing 2: Example salvocfg.h for Library Build Using sfp42cab.lib

Salvo LE & Pro Library Build

```
#define OSUSE_LIBRARY           TRUE
#define OSLIBRARY_TYPE          OSL
#define OSLIBRARY_CONFIG        OSA
```

Listing 3: Example salvocfg.h for Library Build Using slp42cab.lib

Salvo Pro Source-Code Build

```
#define OSENABLE_IDLING_HOOK    TRUE
#define OSENABLE_SEMAPHORES     TRUE
#define OSEVENTS                1
#define OSLOC_ALL               bank1 persistent
#define OSTASKS                 3
```

Listing 4: Example salvocfg.h for Source-Code Build

Performance

Memory Usage

tutorial memory usage[13]	total ROM[14]	total RAM[15]
`tu1lite`	156	22
`tu2lite`	281	26
`tu3lite`	307	28
`tu4lite`	603	34
`tu5lite`	920	51
`tu6lite`	1026	56
`tu6pro`[16]	942	52

Table 5: ROM and RAM requirements for Salvo Applications built with HI-TECH's PICC C Compiler

Special Considerations

Stack Issues

For architectural reasons, HI-TECH's PICC C compiler does not pass parameters on the stack. Nor does it allocate memory for auto (local) variables on the stack. Instead, it employs a *static overlay* model. This has advantages in speed and memory utilization, but it precludes recursion and has other impacts.

Multiple Callgraph Issues

By default, it is expected that Salvo services will only be called from the background / main loop / task level. This is the default configuration for source-code builds. *b*-variant libraries allow service calls only from the background level. Should you wish to call certain services from the foreground / interrupt level, you will need to set `OSCALL_OSXYZ` configuration options for source-code builds or use a different library (see Table 3) for library builds.

From *Variant*, above, we find that the `f`-variant libraries allow you to call event-reading and –signaling services from the foreground. Similarly, the `a`-variant libraries allow you to call the applicable services from anywhere in your code.

The interrupt_level Pragma

When using the a-variant libraries, each instance of an applicable service in use must be called from the foreground, i.e. from an interrupt. Also, PICC's `interrupt_level` pragma must be set to 0 and placed immediately ahead of the application's interrupt routine, like this:

```
#pragma interrupt_level 0[17]
void interrupt IntVector( void )
{
  OSStartTask(TASK_P);
}
```

Listing 5: Setting the HI-TECH PICC interrupt_level Pragma for an ISR when Using a-variant Libraries

PICC requires this in order to manage the parameter overlay areas for functions located on multiple call graphs.

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

Example: Foreground Signaling of One Event Type

In a library build, if you were to move a call to `OSSignalBinSem()` from a Salvo task (i.e. from the background) to an interrupt handler (i.e. to the foreground) without changing the library variant, you'd find that the application crashes from a stack overflow almost immediately. This is because the default interrupt control[18] in `OSSignalBinSem()` is incompatible with being placed inside an interrupt. To circumvent this, you must change `OSLIBRARY_VARIANT` to `OSF` and link an `f`-variant library (e.g. `sfp42Caf.lib` — note the `f` for *foreground* in the variant field) in order to properly support event service calls in the foreground.

Example: Foreground and Background Signaling of One Event Type

If we call `OSSignalBinSem()` from a task and from within an interrupt handler without addressing the callgraph issues, the compiler issues an error message:

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

To resolve this, add the `interrupt_level 0` pragma to your interrupt handler (see Listing 5, above) and use the a-variant library after setting `OSLIBRARY_TYPE` to `OSA`.

OSProtect() and OSUnprotect()

HI-TECH's PICC C compiler requires that when a function is contained in multiple callgraphs, interrupts must be disabled "around" that function to prevent corruption of parameters and/or return values.[19] Therefore you must call `OSProtect()` immediately before and `OSUnProtect()` immediately after all background instances of every Salvo service that is called from both the background and foreground levels, e.g.:

```
void TaskN ( void )
{
  …
  OSProtect();
  OSSignalBinSem(SEM_P);
  OSUnProtect();
  …
}

#pragma interrupt_level 0
void interrupt IntVector( void )
{
  OSSignalBinSem(SEM_P);
}
```

Example: Mixed Signaling of Multiple Event Types

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

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

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

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

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

This creates a call graph acceptable to HI-TECH's PICC C compiler and allows a successful compile and execution. Interestingly, the optimizer will remove the call from the final application.

---

1 The SYSH Salvo test system is for use with the Microchip MPLAB-ICD. This debugging tool is not supported in MPLAB IDE v6, therefore the SYSH projects are only for MPLAB IDE v5.

2 This is done automatically through the `HI_TECH_C`, `_PIC12`, `_PIC14` and `_PIC16` symbols defined by the compiler.

3 The lack of an addressable stack severely limits the scope of alternate methods.

4 As of PICC v7.87.

5 This indicates to the code generator that external memory in addition to internal memory is present, and it will use (function) calls of a longer form. See the PICC User's Guide for more information on the `-ROM` command-line option.

6 Note that the PICC library is automatically added to the linker command line. The Salvo library must be added manually by the user as part of setting up the project.

7 A library may have no variants if the target processor does not support interrupts.

8 Each event flag has RAM allocated to its own event flag control block.

9 Each message queue has RAM allocated to its own message queue control block.

10 Argument for PICC's `#pragma interrupt_level` for those services that can be called from within an ISR.

11 PICmicro MCUs with only one bank of RAM have their Salvo variables located in RAM Bank 0.

12 By making Salvo's variables `persistent`, the PICC compiler is able to omit some initialization code and thus reduce ROM requirements.

13 Salvo v3.2.0 with PICC v8.01PL3.

14 In words.

15 In bytes, all banks.

16 Salvo Pro build differs slightly from Salvo Lite build due to configuration – see tutorial's `salvocfg.h`.

17 Salvo always uses level 0.

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

[19] See PICC manual for more information.