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

RM-IAR18
Reference Manual

Salvo Compiler Reference Manual – IAR PIC18 C

Salvo™
The RTOS that runs in tiny places.™

Introduction

This manual is intended for Salvo users who are targeting Microchip (http://www.microchip.com/) PIC18 PICmicro® MCUs with IAR's (http://www.iar.com/) PIC18 C compiler.

Related Documents

The following Salvo documents should be used in conjunction with this manual when building Salvo applications with IAR's PIC18 C compiler:

Salvo User Manual
Application Note AN-14

Example Projects

Example Salvo projects for use with IAR's PIC18 C compiler and the Microchip MPLAB v5.x IDE can be found in the:

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

directories of every Salvo for Microchip PICmicro® MCUs distribution.

Features

Table 1 illustrates important features of Salvo's port to IAR's PIC18 C compiler.

general	
available distributions	Salvo Lite, LE & Pro for Microchip PICmicro® MCUs
supported targets	PIC18 PICmicro® MCUs
header file(s)	`portiar18.h`
other target-specific file(s)	`portpic18.c`
project subdirectory name(s)	SYSP
salvocfg.h	
compiler auto-detected?	yes[1]
libraries	
`\salvo\lib` subdirectory	`iar18`
context switching	
method	via `OSCtxSw(label)`
`_OSLabel()` required?	no
size of auto variables and function parameters in tasks	unrestricted
memory	
memory models supported	`small` and `large`
interrupts	
controlled via	`GIEL` and/or `GIEH` bits. Controlled via `OSPIC18_INTERRUPT_MASK` configuration option
interrupt status preserved in critical sections?	yes
method used	external function to mimic operation of __monitor keyword, with flexibility to control GIEL and/or GIEH
nesting limit	8 levels
alternate methods possible?	yes[2]
debugging	
source-level debugging?	yes
compiler	
bitfield packing support?	no
printf() / %p support?	yes / yes
va_arg() support?	yes

Table 1: Features of Salvo Port to IAR's PIC18 C Compiler

Compiler Optimizations

Incompatible Optimizations

None of IAR's PIC18 C compiler's optimizations are known to be incompatible with Salvo.[3]

Libraries

Nomenclature

The Salvo libraries for IAR's PIC18 C compiler follow the naming convention shown in Figure 1.

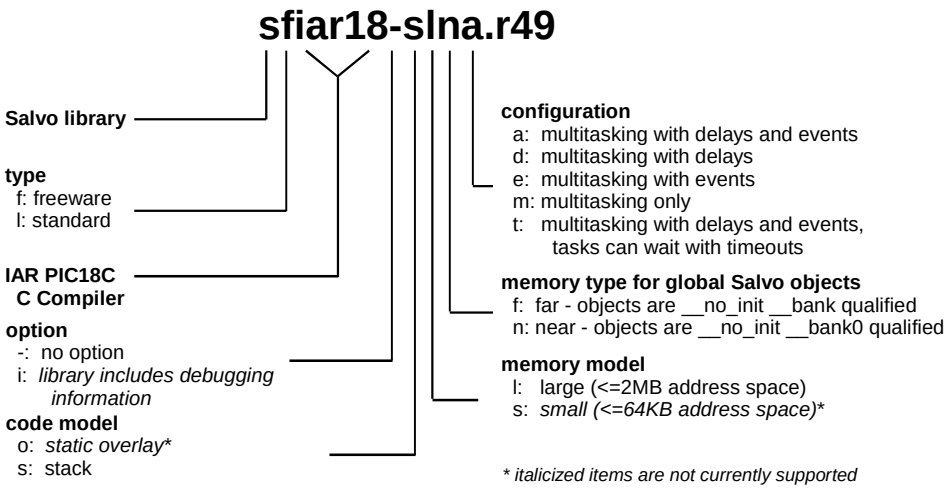

Figure 1: Salvo Library Nomenclature – IAR's PIC18 C Compiler

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

No target-specific identifiers are required.

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 IAR's PIC18 C compiler C compiler's `--debug` command-line option. This adds source-level debugging information to the libraries, making them ideal for source-level debugging and stepping in the C-SPY debugger. To use these libraries, simply select one that includes the debugging information (e.g. `sliar18islna.r49`)

instead of one without (e.g. `sliar18-slna.r49`) in your Embedded Workbench project.

Code Model

Currently, only the IAR PIC18 C compiler's stack code model is supported. This allows for reentrancy, etc.

Memory Model

Currently, only the IAR PIC18 C compiler's `large` memory model is supported. In library builds, the memory model applied to all of the source files must match that used in the library. For source-code builds, the same memory model must be applied to all of the source files.

memory model code	description
`l / OSL:`	*L*arge memory model. Program space is a maximum of 1M words (2MB).
`s / OSS:`	*S*mall memory model. Program space is a maximum of 32K words (64KB).

Table 2: Memory Models for Salvo Libraries – IAR's PIC18 C Compiler

Note Unlike the library configuration and variant options specified in the `salvocfg.h` file for a library build, none is specified for the selected memory model. Therefore particular attention must be paid to the memory model settings used to build an application. The memory model is usually specified on a node-by-node basis inside an IDE (e.g. MPLAB).

Memory Type for Global Salvo Objects

You can choose the memory type for Salvo's global objects in your application by choosing the appropriate library. `near` type objects can be accessed the fastest, but consume precious RAM in the Access Bank. `far` type objects will be placed in banked RAM, which will result in slower accesses. The global object codes are listed in Table 3.

memory type code	description
`f / OSF:`	Salvo objects are declared as type `__no_init __bank`, and will be located in banked RAM.
`n / OSN:`	Salvo objects are declared as type `__no_init __bank0`, and will be located in the first 128 bytes of internal RAM (i.e. in access RAM).

Table 3: Memory Types for Salvo Libraries – IAR's PIC18 C Compiler

The code required to access Salvo's global objects (e.g. the task control blocks, or `tcbs`) will vary in size and speed depending on where the objects are located. `__bank0` type objects can be accessed the fastest, but consume precious RAM in the Access Bank. `__bank` type objects will be placed in banked RAM, which will result in slower accesses.

Since there are only 128 bytes of access RAM in the PIC18 architecture, in larger applications it may be necessary to place Salvo's global objects in banked RAM.

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 IAR's PIC18 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.

compiled limits	
max. number of tasks	3
max. number of events	5
max. number of event flags[4]	1
max. number of message queues[5]	1
target-specific settings	
delay sizes	8 bits
idling hook	enabled
interrupt-enable bits during critical sections	`GIEH = GIEL = 0`
message pointers	can point to ROM or RAM
Salvo objects	far
system tick counter	available, 32 bits
task priorities	enabled
watchdog timer	cleared in `OSSched()`.

Table 4: Build Settings and Overrides for Salvo Libraries for IAR's PIC18 C Compiler

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 20 Salvo libraries for IAR's PIC18 C compiler. 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 PIC18C452.

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_GLOBALS       OSF
#define OSLIBRARY_CONFIG        OSA
```

Listing 1: Example salvocfg.h for Library Build Using sfiar18-slfa.lib

Salvo LE & Pro Library Build

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

Listing 2: Example salvocfg.h for Library Build Using sliar18-slfa.lib

Salvo Pro Source-Code Build

```
#define OSENABLE_IDLING_HOOK    TRUE
#define OSENABLE_SEMAPHORES     TRUE
#define OSEVENTS                1
#define OSIAR_PIC18_ATTR_ALL    __no_init
#define OSLOC_ALL               __bank0
#define OSTASKS                 3
```

Listing 3: Example salvocfg.h for Source-Code Build

Performance

Memory Usage

tutorial memory usage[6]	total ROM[7]	total RAM[8]
`tu1lite`	494	11
`tu2lite`	858	24
`tu3lite`	942	26
`tu4lite`	1902	34
`tu5lite`	3042	53
`tu6lite`	3632	56
`tu6pro`[9]	3338	52

Table 5: ROM and RAM requirements for Salvo Applications built with IAR's PIC18 C Compiler

Special Considerations

Stack Issues

For architectural reasons, IAR's PIC18 C compiler passes parameters on a software stack, and uses the PIC18's hardware stack for `call…return` addresses. While the compiler supports both stack (reentrant) and static overlay models, Salvo supports only the stack model.

Locating Global Salvo Objects in Source-Code Builds

With IAR's PIC18 C compiler, the initialization of Salvo's global objects can be controlled *en masse* through the `OSIAR_PIC18_ATTR_ALL` configuration option. When set to `__no_init`, Salvo's global objects will not be initialized. This is useful in cases where you wish to maintain Salvo's state across wake-from-sleep resets, etc. When used thusly, `OSInit()` must be called to initialize Salvo's global objects at least once.

To selectively place certain Salvo global objects in access or banked RAM, set Salvo's `OSLOC_XYZ` configuration parameters to `__bank`, `__bank0`, etc..

Interrupt Control

The PIC18 architecture supports two distinct priority levels. When enabled, two separate global-interrupt-enable bits, `GIEH` and `GIEL`, are used to control high- and low-priority interrupts, respectively.

Interrupts are automatically disabled within Salvo's critical sections. By default, both `GIEH` and `GIEL` are reset (i.e. made `0`) during critical sections. This is controlled by Salvo's `OSPIC18_INTERRUPT_MASK` configuration option (default value: `0xC0`).

Salvo Pro users can reconfigure the way in which interrupts are disabled during critical sections by redefining `OSPIC18_INTERRUPT_MASK` in the project's `salvocfg.h`. For example, *if Salvo services* (e.g. `OSTimer()`) *are called only from low-priority interrupts*, then a value of `0x40` for `OSPIC18_INTERRUPT_MASK` ensures that only low-priority interrupts are disabled during a Salvo critical section. In this configuration, high-priority interrupts will therefore be unaffected by Salvo. This is especially useful when high-rate interrupts are present.

---

1 This is done automatically through the `__IAR_SYSTEMS_ICC__` and `__TID__` symbols defined by the compiler.
2 Via either in-line assembly or a function call.
3 As of v2.10, the __monitor keyword was known to behave incorrectly.
4 Each event flag has RAM allocated to its own event flag control block.
5 Each message queue has RAM allocated to its own message queue control block.
6 Salvo v3.2.1 with IAR PIC18 C v2.10A.
7 In bytes.
8 In bytes, all banks, `udata`. Does not include stack (default: 0x130 bytes). Salvo global objects are in access RAM (`near`).
9 Salvo Pro build differs slightly from Salvo Lite build due to configuration – see tutorial's `salvocfg.h`.