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

RM-MCC30
Reference Manual

Salvo Compiler Reference Manual – Microchip MPLAB C30

Salvo™
The RTOS that runs in tiny places.™

created by Andrew E. Kalman on Aug 21, 2007 updated on Jan 29, 2010

Introduction

This manual is intended for Salvo users who are targeting Microchip PIC24® and dsPIC® single-chip microcontrollers with Microchip's (http://www.microchip.com/) MPLAB C30 C compiler.

Related Documents

The following Salvo documents should be used in conjunction with this manual when building Salvo applications with Microchip's MPLAB C30 C compiler:

- *Salvo User Manual*

Example Projects

Example Salvo projects for use with Microchip's MPLAB C30 C compiler can be found in the:

```
\Pumpkin\Salvo\Example\PIC\PIC24
```

directories of every Salvo for Microchip PIC24® MCUs and dsPIC® DSCs distribution.

Features

Table 1 illustrates important features of Salvo's port to Microchip's MPLAB C30 C compiler.

General	
Abbreviated as	MCC30
Available distributions	Salvo Lite, LE & Pro for Microchip PIC24® MCUs and dsPIC® DSCs
Supported targets	all PIC24, dsPIC30F & dsPIC33F devices
Header file(s)	`salvoportmcc30.h`
Other target-specific file(s)	`salvoportmcc30.s`, `salvohook_interrupt_PIC24_IRQ.c`
salvocfg.h	
Compiler auto-detected?	yes[1]
Include target-specific header file in `salvocfg.h`?	recommended
Libraries	
Located in	`Lib\MCC30-v2` (for v2.x compilers and early linker) `Lib\MCC30-v3` (for v3.x compilers and later linker)
Behavior of user hooks in libraries	do nothing (dummy functions)
Same libraries for PIC24 & dsPIC families?	`yes`
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 65,535 8-bit bytes
Interrupts	
Interrupt latency in context switcher	0 cycles
Interrupts in critical sections controlled via	`OSDisableHook()`, `OSEnableHook()`, `OSRestoreHook()`, `OSSaveHook()`
Interrupt status preserved in critical sections?	optional, via appropriate user functions
Method used in critical sections	see example user functions
Memory	
Memory models supported	`small` and `large`
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 Microchip's MPLAB C30 C compiler

Libraries

Nomenclature

The Salvo libraries for Microchip's MPLAB C30 C compiler C compiler follow the naming convention shown in Figure 1.

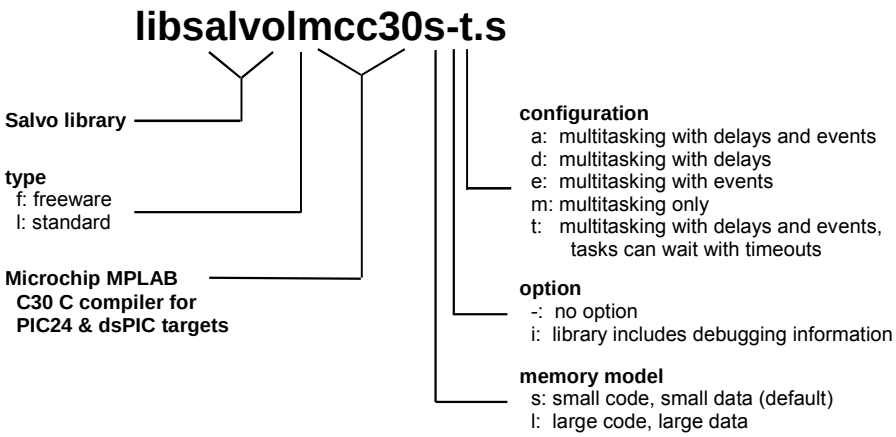

Figure 1: Salvo library nomenclature – Microchip's MPLAB C30 C compiler

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

Since the CPU instruction set is common to both the PIC24® and dsPIC® architectures, all PIC24® and dsPIC® targets use the same Salvo libraries.

Memory Model

The MPLAB C30 C compiler's `small` (default) and `large` code and data memory models are supported:

Code Models	
small	up to 32K words program memory
large	over 32K words of program memory

Table 2: Code models for Microchip's MPLAB C30 C compiler

Data Models	
small	up to 8KB of data memory
large	over 8KB of data memory

Table 3: Code models for Microchip's MPLAB C30 C compiler

Note Salvo libraries for Microchip's MPLAB C30 C compiler C compiler are compiled with either *small code and small data models*, or *large code and large data models*.

If additional flexibility is required, a Salvo Pro user can build an application with a different combination of memory models, e.g., the small code and the large data models by using Salvo source code.

See Configuring for Different Memory Models for information on properly selecting the different memory models.

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 the appropriate command-line options. This adds source-level debugging information to the libraries, making them ideal for source-level debugging and stepping in the MPLAB IDE. To use these libraries, simply select one that includes the debugging information (e.g. `libsalvolmcc30lit.a`) instead of one without (e.g. `libsalvolmcc30l-t.a`) in your MPLAB 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 Microchip's MPLAB C30 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	4
Max. number of events	8
Max. number of event flags	1
Max. number of message queues	1
Target-specific Settings	
Delay sizes	8 bits
Idling hook	enabled
Interrupt-enable bits during critical sections	controlled via user functions
System tick counter	available, 32 bits
Task priorities	enabled
Watchdog timer	controlled via user functions

Table 4: Build settings and overrides for Salvo libraries for Microchip's MPLAB C30 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

Salvo Lite for Microchip PIC24® MCUs and dsPIC® DSCs contains a single freeware library in a single configuration. Salvo LE for Microchip PIC24® MCUs and dsPIC® DSCs adds standard libraries in multiple configurations. Salvo Pro for Microchip PIC24® MCUs and dsPIC® DSCs adds standard libraries in multiple configurations with debugging information included.

Each Salvo for Microchip PIC24® MCUs and dsPIC® DSCs distribution contains the Salvo libraries of the lesser distributions beneath it. Additionally, Salvo Pro distributions contain makefiles for all possible library configurations.

Target-Specific Salvo Source Files

Depending on the desired code model, Table 5 illustrates that different target-specific source files are required for Salvo Pro source-code builds.

Target-specific Source Files	
small	Src\MCC30\salvoportmcc30-sm.s
large	Src\MCC30\salvoportmcc30-lm.s

Table 5: Target-specific files required for different code models of Microchip's MPLAB C30 C compiler

Note These files are independent of the OSMCC30_LARGE_CM symbol.

salvocfg.h Examples

Below are examples of `salvocfg.h` project configuration files for various different Salvo distributions and the PIC24HJ256GP610.

Salvo Lite Library Build

```
#define OSUSE_LIBRARY           TRUE
#define OSLIBRARY_TYPE          OSF
#define OSLIBRARY_CONFIG        OST
#define OSTASKS                 3
#define OSEVENTS                4
#define OSEVENT_FLAGS           0
#define OSMESSAGE_QUEUES        1
```

Listing 1: Example salvocfg.h for library build using libsalvofmcc30s-t.a

Salvo LE & Pro Library Build

```
#define OSUSE_LIBRARY           TRUE
#define OSLIBRARY_TYPE          OSL
#define OSLIBRARY_CONFIG        OSA
#define OSTASKS                 7
#define OSEVENTS                11
#define OSEVENT_FLAGS           0
#define OSMESSAGE_QUEUES        4
```

Listing 2: Example salvocfg.h for library build using libsalvolmcc30s-a.a or libsalvolmcc30sia.a

Salvo Pro Source-Code Build

```
#define OSEVENTS                    9
#define OSEVENT_FLAGS               1
#define OSMESSAGE_QUEUES            2
#define OSTASKS                     17

#define OSENABLE_BINARY_SEMAPHORES  TRUE
#define OSENABLE_TIMEOUTS           TRUE
#define OSBYTES_OF_DELAYS           4
#define OSBYTES_OF_TICKS            4
```

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

Performance

Interrupt Latencies

Since Salvo's context switcher for Microchip's MPLAB C30 C compiler 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 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

Examples of the total memory usage of actual Salvo-based applications are listed below.

Example Application[2]	Program Memory Usage[3]	Data Memory Usage[4]
`tut5lite (for PIC24)`	3813	84
`tut5le (for PIC24)`	3747	84
`tut5pro (for PIC24)`	3651	82

Table 6: Program and data memory requirements for Salvo applications built with Microchip's MPLAB C30 C compiler

Special Considerations

Configuring for Different Memory Models

When building a Salvo application with MPLAB C30, the memory models of all objects linked together to form an application must be consistent. The memory models are specified in the MPLAB IDE under Project → Build Options → Project → MPLAB C30 → Memory Model.[5]

In library builds, the memory models applied to all of the source files must match that used in the library – a mismatch may generate link-time errors and or runtime errors. For source-code builds, the same memory models must be applied to all of the source files.

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(s). Therefore particular attention must be paid to the memory model settings used to build an application. The memory model is usually specified on a project-wide basis in the MPLAB IDE.

Code Models

Use of the large code model requires that the Salvo symbol OSMCC30_LARGE_CM be defined for all Salvo code modules.[6] Symbols can be defined in the MPLAB IDE under Project → Build Options → Project→ MPLAB C30 → General → Preprocessor Macros.

Note OSMCC30_LARGE_CM should *not* be defined when using the small code model.

See also Target-Specific Salvo Source Files, above.

Data Models

No symbols are required for the small or large data models.

Compiler Issues

Incompatible Optimizations

There are no known incompatible optimizations.

---

1 This is done automatically through the `C30`, `__C30` and/or `__C30__` symbols defined by the compiler.

2 Salvo 4.0.0.

3 In bytes. Includes `.reset`, `.ivt`, `.aivt`, `.text`, `.dinit` & `.isr` sections.

4 In bytes. Includes `.nbss` section. This represents all of Salvo's objects. Does not include RAM allocated to the heap or stack. Salvo applications typically require the same (small) stack size as simple, non-multitasking applications.

5 The MPLAB IDE passes command-line arguments to the MPLAB C30 compiler and linker as part of the build process. E.g., `–mlarge-code -mlarge-data` for large code and data models.

6 Unfortunately the MPLAB C30 compiler does not emit any symbols that identify which memory model(s) are in use … therefore Salvo users must define this symbol, which drives conditional compilation of the Salvo source code that is affected by the memory model settings.