PUMPKIN™
REAL-TIME SOFTWARE

RM-MSCX86
Reference Manual

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

Salvo Compiler Reference Manual – Microsoft C++

Salvo™
The RTOS that runs in tiny places.™

created by Andrew E. Kalman on May 13, 2008 updated on May 23, 2008

Introduction

This manual is intended for Salvo users who are targeting PCs with Microsoft's (http://www.microsoft.com/) C++ compiler. Normally, such users wish to *simulate* their Salvo application on a PC before eventually running it on another target (e.g. 8051). By using Salvo for x86, you can develop and debug the multitasking and other aspects of your application on a PC before retargeting it for your intended embedded platform.

Note A recent version of Microsoft's Visual C++ tool suite is required.[1] Microsoft offers *free* versions of their Visual C++ compiler in the form of Visual C++ 2008 Express Edition on their website at http://www.microsoft.com/express.[2]

Tip By downloading and installing Visual C++ and Salvo Lite for x86 you can immediately begin building Win32 Salvo applications.

Compatibility

Salvo for x86 is compatible with the following Microsoft Visual C++ compiler versions:

Release Name	Visual C++ Version Number
Visual C++ .NET 2003	`7.1`
Visual C++ 2005	`8.0`
Visual C++ 2008	`9.0`

Note Salvo for x86 may also be compatible with earlier versions of the Visual C++ toolset, but this has not been verified.[3] Since Visual C++ 2008 Express is available as a free download, we expect most users to be working with the version 9.0 compiler.

Related Documents

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

- *Salvo User Manual*

Example Projects

Example Salvo projects for use with Microsoft's C++ compiler and the Visual Studio IDE can be found in the:

```
\Pumpkin\Salvo\Example\x86
```

directories of every Salvo for x86 distribution.

Tip These example projects can be easily modified for nearly all x86-class PCs.

Features

Table 1 illustrates important features of Salvo's port to Microsoft's C++ compiler.

General	
Abbreviated as	`MSCX86`
Available distributions	Salvo Lite, LE & Pro for x86
Supported targets	all 32-bit x86 architectures
Header file(s)	`salvoportmscx86.h`
Other target-specific file(s)	`salvoportmscx86.c`
salvocfg.h	
Compiler auto-detected?	yes[4]
Include target-specific header file in `salvocfg.h`?	n/a
Libraries	
`Located in`	`Lib\MSCX86`
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 2^24 8-bit bytes
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 Microsoft's C++ compiler

Libraries

Nomenclature

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

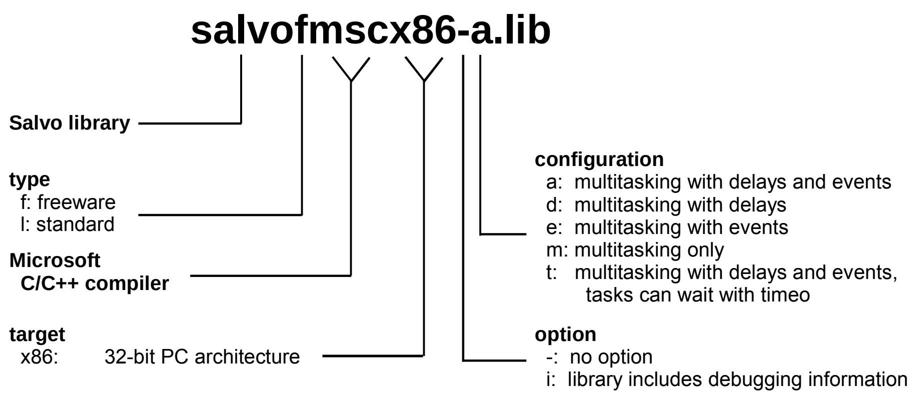
salvofmscx86-a.lib
Salvo library
type
f: freeware
l: standard
Microsoft
C/C++ compiler
target
x86: 32-bit PC architecture
configuration
a: multitasking with delays and events
d: multitasking with delays
e: multitasking with events
m: multitasking only
t: multitasking with delays and events, tasks can wait with timeo
option
-: no option
i: library includes debugging information

Figure 1: Salvo library nomenclature – Microsoft's 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

Salvo for x86 is compatible with all 32-bit x86 architectures.

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 Microsoft's C++ compiler 's `/Zd` command-line option. This adds source-level debugging information to the libraries, making them ideal for source-level debugging and stepping in the Visual Studio debugger. To use these libraries, simply select one that includes the debugging information (e.g. `salvolmscx86it.lib`) instead of one without (e.g. `salvolmscx86-t.lib`) in your Visual Studio 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.

Note Unlike most Salvo Lite distributions – which include only the 't' (timeouts enabled) library configuration – Salvo Lite for x86 includes all five of the standard 'a' through 't' library configurations.

Build Settings

Salvo's libraries for Microsoft's 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 2.

Note Certain Salvo for x86 library settings (e.g. the 32-bit delays, set via the advanced configuration option `OSBYTES_OF_DELAYS`) differ from the defaults used to build the libraries in other Salvo distributions. This is because there is no real cost in data and program memory sizes for these configuration options when running on a PC with unlimited memory. Failure to observe these differences may lead to confusion when comparing a Salvo application on x86 to one on a native target.

For example, a Salvo Lite for PICmicro MCUs application will be limited to 8-bit delays.[5] The same application built with Salvo Lite for x86[6] can be built with 32-bit delays. Long delays in the Salvo Lite for x86 application will work properly, whereas they will appear as `(delay modulo 256)` delays in the Salvo for PICmicro MCUs application. In this example, both Salvo Lite distributions can handle 8-bit delays, but only Salvo Lite for x86 can handle 32-bit delays. Users who wish to use 32-bit delays in their target applications must upgrade to Salvo Pro, which allows user access to advanced configuration options (like delays).

Target-specific Settings	
Delay sizes	**32** bits
Idling hook	dummy, can be overridden
Interrupt hook	dummy, 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 Microsoft's C++ compiler

Note Unlike other Salvo distributions, Salvo Lite for x86 libraries have *no precompiled limits*[7] for the number of supported tasks, events, etc.

Tip Additional Salvo configuration options are enabled for Salvo for x86 libraries (e.g. `OSENABLE_STACK_CHECKING`, `OSGATHER_STATISTICS`, etc.). See the `salvolib.h` header file for more information.

Available Libraries

Salvo Lite for x86 contains freeware libraries in all configurations. Salvo LE for x86 adds standard libraries in all configurations. Salvo Pro for x86 adds standard libraries in all configurations with debugging information included. Each Salvo for x86 distribution contains the Salvo libraries of the lesser distributions beneath it.

Target-Specific Salvo Source Files

The source file `salvoportmscx86.c` is needed for Salvo Pro source-code builds.

salvocfg.h Examples

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

Salvo Lite Library Build

```
#define OSUSE_LIBRARY              TRUE
#define OSLIBRARY_TYPE             OSF
#define OSLIBRARY_CONFIG           OST
#define OSTASKS                    23
#define OSEVENTS                   17
#define OSEVENT_FLAGS              6
#define OSMESSAGE_QUEUES           5
```

Listing 1: Example salvocfg.h for library build using salvofmscx86-t.lib

Salvo LE & Pro Library Build

```
#define OSUSE_LIBRARY             TRUE
#define OSLIBRARY_TYPE            OSL
#define OSLIBRARY_CONFIG          OST
#define OSTASKS                   42
#define OSEVENTS                  13
#define OSEVENT_FLAGS             3
#define OSMESSAGE_QUEUES          2
```

Listing 2: Example salvocfg.h for library build using salvolmscx86-t.lib or salvolmscx86it.lib

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

Performance

Interrupt Latencies

Since Salvo's context switcher for Microsoft's 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.

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 Microsoft's C++ compiler

Users can replace it (e.g. with code to log the idling performance of the system) by building their own version with their application.

Interrupt

The default interrupt hooks in `salvohook_interrupt.c` are dummy functions, as shown below.

```
void OSDisableHook ( void )
{
  ;
}


void OSEnableHook ( void )
{
  ;
}
```

Listing 5: Default Salvo interrupt hooks for Microsoft's C++ compiler

Tip Since the PC's interrupt scheme is considerably more complex and/or less transparent than that of a typical microcontroller, it is recommended that interrupt control *not* be used in a Salvo for x86 application. Interrupt should be simulated at the background / `main()` level, preferably from within the loop that calls the Salvo scheduler.

Watchdog

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

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

Listing 6: Default Salvo watchdog hook for Microsoft's C++ compiler

Users can replace it by building their own version with their application.

Compiler Issues

Incompatible Optimizations

There are no know Microsoft C++ compiler optimizations that are incompatible with Salvo for x86.

IDE Issues

Precompiled Headers

Many users are likely to develop in C++ when using Salvo with Visual C++. Salvo 4 is compatible with C++ applications. To avoid build errors, set the Precompiled Headers property[8] of each Salvo source file (e.g. `salvomem.c`) to Not Using Precompiled Headers. Otherwise it will generate an error:

```
fatal error C1010: unexpected end of file while
looking for precompiled header directive.
```

Listing 7: Error when using precompiled headers with Visual C++

LIBCD.lib

You may encounter the linker error below if you are using Visual C++ 2005 or later, and the libraries in the Salvo Lite for x86 distribution were built with an earlier version of the Visual C++ compiler:

```
LINK : fatal error LNK1104: cannot open file
'LIBCD.lib'
```

Listing 8: Error when LIBCD.lib cannot be found

This occurs because "(t)his file is the static library for the debug single threaded version of the C runtime. Visual Studio 2005 no longer supports this version of the C runtime … " [9]

The simplest solution is to tell Visual Studio to ignore the LIBCD library. To do so, in the project's Properties pages, choose Configuration Properties → Linker → Input and enter LIBCD in the Ignore Specific Library field. The build will then complete without error, as shown below:[10]

```
------ Rebuild All started: Project: Lite, Configuration: Debug Win32 ------
Deleting intermediate and output files for project 'Lite', configuration
'Debug|Win32'
Compiling...
cl : Command line warning D9035 : option 'Wp64' has been deprecated and will be
removed in a future release
stdafx.cpp
Compiling...
cl : Command line warning D9035 : option 'Wp64' has been deprecated and will be
removed in a future release
salvomem.c
salvohook_idle.c
salvohook_interrupt.c
salvohook_wdt.c
tut5.c
Generating Code...
Compiling manifest to resources...
Microsoft (R) Windows (R) Resource Compiler Version 6.0.5724.0
Copyright (C) Microsoft Corporation.  All rights reserved.
Linking...
Embedding manifest...
Microsoft (R) Windows (R) Resource Compiler Version 6.0.5724.0
Copyright (C) Microsoft Corporation.  All rights reserved.
Build log was saved at
"file://c:\Pumpkin\Salvo\Example\x86\x86\Win32\Tut\Tut5\Microsoft Visual
Studio\tut5\Lite\Debug\BuildLog.htm"
Lite - 0 error(s), 2 warning(s)
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========
```

Listing 9: Successful build while ignoring LIBCD.lib

[1] Salvo was ported to x86 using Visual C++ .NET 2003 (aka Visual C++ 7.1).
[2] Older versions – e.g. Visual C++ 2005 Express – are also available via the Previous Version link.
[3] Compatibility with earlier versions is more likely when doing source-code builds.
[4] This is done automatically through the `_MSC_VER` symbol defined by the compiler.
[5] Because the precompiled Salvo Lite for PICmicro MCUs libraries are built with 8-bit delays.

[6] Or Salvo Pro for PICmicro MCUs, since all Salvo Pro distributions permit changes to advanced configuration options.

[7] The preprocessor will flag as an error numbers of tasks, events, etc. in excess of 1000.

[8] This is done in Visual Studio 2003 .NET by selecting the Properties of one or more files, then General Configuration Properties, then C++, then Precompiled Headers, and under Create/Use Precompiler Header selecting Not Using Precompiled Headers.

[9] Jonathan Caves, Microsoft Visual C++ Compiler Team, http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=332295.

[10] Salvo Lite x86 project tut5lite built with Visual C++ 2008 Express. The Salvo Lite x86 was built using Visual Studio .NET 2003.