NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Operating System Abstraction Layer (OSAL)

Configuration Guide

582-2007-00

January 31, 2016 (Version 4.2)

Authors:

X Cudmore, Alan P. (GSFC-5820)

Digitally signed by Cudmore, Alan P. (GSFC-5820)
DN: cn=Cudmore, Alan P. (GSFC-5820), o=NASA GSFC, ou=Code 582, email=Alan.P.Cudmore@nasa.gov, c=US
Date: 2016.02.22 09:49:55 -05'00'

Alan Cudmore
Flight Software Engineer/GSFC-5820

Changed 2 hours ago by jphickey

X I approve of the final version of the document

Joseph Hickey
Flight Software Engineer/GRC:LSS0

Approvals:

X *Susanne Strege*

Susanne Strege
cFS Product Development Lead/GSFC-5820

SUSANNE STREGE

Digitally signed by SUSANNE STREGE
DN: c=US, o=U.S. Government, ou=NASA, ou=PIV, cn=SUSANNE STREGE, 0.9.2342.19200300.100.1.1=sstrege
Date: 2016.02.18 22:16:22 -05'00'

Acknowledgements

Revision History

Revision Number	Release Date	Changes to Prior Revision	Approval
1.0	10/17/07	Initial Release.	A. Cudmore
1.1	02/13/08	Updates for RTEMS, Linux, and Cygwin for 2.11 release	A. Cudmore
1.2	09/05/08	Updates for OSAL 2.12 release	A. Cudmore
1.3	03/10/10	Updates for OSAL 3.1 release	A. Cudmore
1.4	5/25/2011	Updates for OSAL 3.3 release	A. Cudmore
1.5	12/13/2011	Updates for OSAL 3.4 release – added support for sis-rtems Removed cFE configuration text	A. Cudmore
1.6	12/21/2012	Updates for OSAL 4.0 release – Removed Cygwin and OS X configurations	A. Cudmore
4.1	1/17/2014	Updates for OSAL 4.1 release. Sync document version ID to software version. Add information for building and running unit tests.	A. Cudmore
4.2	1/31/2016	Moved osconfig.h description into section 2.2. Consolidated the "classic" build and prerequisites setup into section 2.3. Added new section 2.4 on provisioning a build using cmake. Minor modifications to subsequent sections only where there was a difference between cmake and classic builds; i.e. the cmake build has no "make config" or "make depend". Updated title page to replace Code 582 banner with cFS. Added header.	J.Hickey S.Strege

Table of Contents

1 Introduction

1.1 Scope

The purpose of this document is to provide guidelines and conventions for the configuration and deployment of the Operating System Abstraction Layer (OSAL) to a desired platform or platforms.

1.2 Background

The goal OS Abstraction Layer is to promote the creation of portable and reusable real time embedded system software. Given the necessary OS abstraction layer implementations, the same embedded software should compile and run on a number of platforms ranging from spacecraft computer systems to desktop PCs.

1.3 Applicable Documents

Document ID	Document Title

1.4 Acronyms

Acronym	Description
OS	Operating System
API	Application Programming Interface
CM	Configuration Management
CPU	Central Processing Unit
EEPROM	Electrically Erasable Programmable Read-Only Memory
HW, H/W	Hardware
RAM	Random-Access Memory
SW, S/W	Software
TBD	To Be Determined

1.5 Glossary of Terms

The following table defines the terms used throughout this document. These terms are identified as proper nouns and are capitalized.

Term	Definition
Application (APP)	A generic term for a computer program in a desktop or embedded system. An Application is generally not part of the operating system.
Application Programmer's Interface (API)	A set of routines, protocols, and tools for building software applications
Board Support Package (BSP)	A collection of user-provided facilities that interface an OS and the cFE with a specific hardware platform. The BSP is responsible for hardware initialization.
Core Flight Executive (cFE)	A runtime environment and a set of services for hosting FSW Applications

Cyclic Redundancy Check	A polynomial based method for checking that a data set has remained unchanged from one time period to another.
Developer	Anyone who is coding a software Application.
Hardware Platform	The target hardware that hosts the Operating System and Applications.
Interface Control Document	A document that describes the software interface, in detail, to another piece of software or hardware.
I/O Data	Any data being written to and read from an I/O port. No structure is placed on the data and no distinction as to the type of I/O device. I/O data is defined separately from memory data because it has a separate API and it's an optional interface of the cFE.
Log	A collection of data that an application stores that provides information to diagnose and debug FSW problems.
Memory Data	Any data being written to and read from memory. No structure is placed on the data and no distinction as to the type of memory is made.
MMU	Memory Management Unit. A piece of hardware that manages virtual memory systems. It automatically translates addresses into physical addresses so that an application can be linked with one set of addresses but actually reside in a different part of memory.
Network	A connection between subsystems used for communication purposes.
Platform	See "Hardware Platform" above.
User	Anyone who interacts with the Software Application or system in its operational state. A user can be a developer, a tester, an operator, or a maintainer.

2 How to Configure, Build, and Run the OSAL

The OSAL distribution includes a complete development environment with support for a number of processors and operating systems. The OSAL development environment has been designed to isolate the portable OS source code from the OSAL applications, configuration parameters, and build products. The development environment is an example of how to configure and build portable software using the OSAL code, but it is by no means a requirement to use the OSAL. The included platforms for the OSAL can be used as starting points for other boards and CPUs.

The following sections provide instructions on how to:

- Setup the build environment
- Configure the build directory for an OSAL application
- Configure a OSAL Application
- Build the OSAL Application
- Load the OSAL Application on to the target platform
- Run the OSAL Application on the target platform

In the current OSAL release, two build systems are available: the "classic" build using ordinary makefiles, and a new build utilizing the "cmake" tool. The "classic" build is mostly carried over from previous OSAL releases and preserves compatibility with existing projects/workflows. The "cmake" build offers increased features by introducing additional scripting and build-time configurability, allowing direct inclusion into larger projects with less need to modify files to support a specific target or configuration.

2.1 Setup the Build Environment

This section details the steps needed to setup the OSAL source distribution and prepare the host development environment to build the OSAL.

2.1.1 Setup the OSAL Source Distribution

Get a copy of the OSAL source distribution directory on your build machine. The source distribution has the following directories:

OSAL source distribution directories

Directory	Description
osal	The top level OSAL source distribution directory. OSAL version 2.10 is being used as an example.
osal/src	The src directory contains the OSAL source, and make rules.
osal/src/examples	The sample directory contains the sample applications for the osal.
osal/src/tests	The tests directory contains a small number of OSAL tests that can run on the targets.
osal/src/unit-tests	The unit-tests directory contains a suite of OSAL unit tests.
osal/src/bsp	The bsp directory contains the platform specific code for the OSAL as well as code to make the OSAL run on a particular platform. Everything in this directory is used to adapt the OSAL and Applications to a particular hardware platform. This directory also contains the startup code for the example programs. The included platforms are generic enough that they may be easy to port to other platforms and processor architectures. For example: The bsp/mcf5235-rtems board support package was ported to an ARM processor running RTEMS with minimal effort.

osal/src/make	The make directory contains common makefiles for building the OSAL and its applications (classic build only)
osal/src/os	The os directory is the heart of the OSAL, containing the implementation of the OSAL for each supported operating system. There is a sub-directory for each supported operating system in this directory. The OSAL include files are also contained in this directory (src/os/inc).
osal/src/inc	The inc directory contains system wide include files that are used by the OSAL on all platforms.
osal/build	The build directory contains the classic framework for building an OSAL application. The files in this directory allow easy customization and configuration for any supported OS or platform for the OSAL. By changing two variables in a file, the OSAL examples and test can be built for any of the supported platforms.
osal/doc	The doc directory contains the documentation and release notes for the OSAL.

The osal directory can go just about anywhere on a host development system.

Example directory structure locations

Host Operating System	Example Directory	Notes
Windows/vxWorks 6 Development Shell	C:\osalproject\osal	1. Building on Windows with the vxWorks 6.x development tools requires using the "vxWorks Development Shell". The system will not build on a standard Cygwin Shell, or a windows DOS prompt.
Linux	/home/osaluser/osal	

2.2 Configure the OSAL Parameter File

The file **osconfig.h** has configuration parameters for tailoring the OSAL parameters. Most parameters set upper bounds on the number of OS objects that can be created. The OSAL keeps track of allocated OS objects using fixed size tables. The OSAL source distribution contains a sample osconfig.h file in the "osal/src/bsp/<bsp>/config" directory, which can either be used as-is or tuned for specific project needs. The osconfig.h file is required regardless of the build system in use (classic or cmake).

OSAL configuration parameters

Parameter	Description
OS_MAX_TASKS	The maximum number of tasks that can be created in the running OSAL application.
OS_MAX_QUEUES	The maximum number of queues that can be created in the running OSAL application.
OS_MAX_COUNT_SEMAPHORES	The maximum number of counting semaphores that can be created in the running OSAL application.
OS_MAX_BIN_SEMAPHORES	The maximum number of binary semaphores that can be created in the running OSAL application.
OS_MAX_MUTEXES	The maximum number of mutexes that can be created in the running OSAL application
OS_MAX_PATH_LEN	The maximum length for an absolute path length in the OSAL File API.
OS_MAX_API_NAME	The maximum length for an individual file name

	in the OSAL File API.
OS_BUFFER_SIZE	The maximum size of a formatted text message for the OS_printf API.
OS_BUFFER_MSG_DEPTH	The maximum number of messages buffered by the OS_printf API.
OS_UTILITY_TASK_ON	Turns on a utility task that will read the statements to print from the OS_printf function. If this define is commented out OS_printf will print the text under the context of the caller.
OS_UTILITYTASK_STACK_SIZE	The size of the stack for the utility task.
OS_UTILITYTASK_PRIORITY	The priority of the utility task.
OSAL_SOCKET_QUEUE	If this is defined, the posix port will use the socket implementation for message queues, rather than the POSIX message queue implementation. This needs to be defined for OS X to work.
OS_MAX_MODULES	Used for defining the maximum number of loadable modules that the OS AL can keep track of. This is used for the new Module Load and Symbol API.
OS_MAX_SYM_LEN	Used for setting the maximum length of a symbol name in the symbol API.
OS_MAX_TIMERS	Used for defining the maximum number of timers in the OSAL.

2.3 Setting up "classic" build

The following procedures are relevant only when using the classic makefile build. For the equivalent cmake instructions, see the next section.

2.3.1 Create System Environment Variable

The OSAL development environment requires one system environment variable to be set in order to build the example programs. The directory also contains a shell script "setvars.sh" to set the environment to the current OSAL directory.

Environment Variables needed by the cFE

Environment Variable	Value (in Linux as an example)	Notes
OSAL_SRC	/home/osaluser/osal	The location of the OS Abstraction Layer source code. This directory can be moved anywhere as long as the environment variable is set accordingly.

Example Environment Variable for Different Development Hosts

Host Operating System	Example Environment Variables	Notes
Windows/vxWorks 6 Development Shell	% set OSAL_SRC=C:/osalproject/osal	1. These environment variables can be set in the Windows control panel under

		system/environment variables. 2. Note the forward slash directory separators in the DOS environment variables. Because the vxWorks tools are half DOS and half-Unix, they don't seem to like the DOS style backslash.
Linux	$ export OSAL_SRC=/home/osaluser/osal	These settings can be set in the user's .bash_profile

2.3.2 Configure the Build Directory for the OSAL application

The build directory is where the OSAL is configured and compiled for a particular processor, board, and OS. The build directory is designed to hold the OSAL configuration for the selected platform. The **core** directory is where the core OS code, and bsp code are built. They are left in the core directory for the applications to link against. The build directory can have multiple OSAL applications to build for a particular platform. The OSAL distribution contains directories for example and test applications. Multiple build directories can be used to configure the OSAL for different platforms in the same environment, each with its own unique OSAL configuration.

2.3.3 Configure the 'build' Directory

In order to build the OSAL for one of the supported platforms, the OSAL build directory must be properly configured. This involves editing a couple of configuration files and setting up one or more sample applications that use the OSAL API.

2.3.4 Define the CPU, Operating System, and Processor Board

In the build directory, edit the **'osal-config.mak'** file and set the options for your target. The default settings in the osal-config.mak are for running vxWorks6.4 on a generic PowerPC board.

osal-config.mak Settings

osal-config.mak variable	Valid selections	Notes
OS	vxworks6, rtems, posix	1. VxWorks 5.5 is no longer supported. 2. posix is tested for 32 bit linux 2.6.x
BSP	genppc-vxworks6.4, mac-posix, pc-posix, mcf5235-rtems, sis-rtems	Use posix for linux
OSAL_M32	-m32 (or commented out)	See below.

Note that not all combinations are valid. See the Platform Specific Section for more information on each supported cFE target.

In some cases, developers may find their compiler toolchain producing "64-bit" images, when "32-bit" images are needed – for example, when using "native" GCC on a 64-bit X86 Linux. The OSAL build files

support use of an **OSAL_M32** build variable to insert the appropriate "please build 32-bit images" compiler parameter into all compilation and linkage commands.

In this use case, the setting of the **OSAL_M32** build variable should be uncommented in the **osal-config.mak** file, and set to the correct flag for the compiler toolchain in use. The build system will also honor settings of this variable made in the developer's shell environment or on the "make" command line..

Usage of this flag may require an optional "multilib" (or similar) package to be installed. Refer to your operating system and toolchain documentation for details, if adding the appropriate flag causes your builds to fail due to (for example) missing 32-bit or multilib related headers or libraries.

2.4 Setting up the "cmake" build

This section covers how to set up the host machine for building OSAL using the cmake build system. Rather than using pre-written makefiles as the classic build does, the cmake build generates makefiles entirely from instructions specified in files contained with the source code itself. This has several advantages:

- Enhanced script-like capabilities
- Generally no need to modify files to adapt to a particular target.
- Easier integration with larger mission projects

Typically, OSAL is not built by itself, but rather as a library to be used within a larger application. The OSAL cmake build system allows both options; building as a standalone entity for testing purposes, or building as a sub-component within a larger project. The same scripts are used in both cases, and no modification is required.

2.4.1 Prerequisites

In order to build OSAL using cmake, the "cmake" package must be installed for the host/development machine. On Linux, this is generally available via the respective Linux distribution package management system, i.e. "yum" on RedHat and derivatives, or "apt-get" on Debian and derivatives. For other operating systems, the cmake tool is available in source and binary form from http://cmake.org. OSAL requires at least version 2.6.4 of the cmake tool.

2.4.2 Variables that must be specified

The OSAL cmake build is controlled by several user-supplied variables when the build is first provisioned:

CMake variable	Valid selections	Notes
OSAL_SYSTEM_OSTYPE	Any directory name that exists under osal/src/os	All OS-specific source files in this directory will be built. An optional, OS-specific "build-options.cmake" file will also be included which may add necessary compiler options specific to that OS.
OSAL_SYSTEM_BSPTYPE	Any directory name that exists under osal/src/bsp	All BSP-specific source files in this directory will be built. An optional, BSP-specific "build-options.cmake" file will also be included which may add necessary compiler options specific to that BSP.
OSAL_INCLUDEDIR	Any directory on the host system (absolute path)	Optional; if specified, this will be included in the compiler include file search path. Typically this is used to specify the location of "osconfig.h" for

		standalone OSAL builds.
ENABLE_UNIT_TESTS	TRUE or FALSE	Optional; defaults to "FALSE" if not specified. If set to TRUE, the included unit test code will also be built in addition to the runtime library.
OSAL_USER_C_FLAGS	Any valid switches for the compiler in use.	Optional; the user may specify any arbitrary compiler switches to use.

It is important to note that the values specified for these variables are **automatically cached** by the cmake build system. It is only necessary to specify these values when first provisioning/configuring a build; these are *not* required when simply building the binaries.

This caching function removes the need for environment variables to be set as in the "setvars.sh" file in the classic build. The cmake build does not require the user to set environment variables; all necessary context information is automatically stored in the cache when the build is first provisioned.

2.4.3 Setting up a standalone OSAL build

The OSAL may be built standalone in order to evaluate the library for a particular target and/or execute the included unit tests.

In the cmake build system, all generated files are placed in a dedicated "binary" directory that is separate from the source tree. To provision a build, the user must first create the binary directory by issuing the "mkdir" command (on Linux), preferably outside the OSAL source tree. Then, the "cmake" provisioning tool is invoked to generate the actual makefiles, supplying values for the required variables:

```
$ mkdir build
$ cd build
$ cmake -DOSAL_SYSTEM_OSTYPE=posix -DOSAL_SYSTEM_BSPTYPE=pc-linux \
  -DENABLE_UNIT_TESTS=TRUE -DOSAL_INCLUDEDIR=/path/to/user/config \
  /path/to/osal/source
```

The cmake provisioning tool generates standard makefiles in the build directory. To build the OSAL binaries, simply run "make" in the build directory.

2.4.4 Integrating OSAL into a larger build

Modularity is a key feature of the cmake system. As such, the OSAL cmake build system can be directly used as a component within a larger "mission" build, as long as the same variables are supplied via the parent cmake script:

```
SET(OSAL_SYSTEM_OSTYPE "posix")
SET(OSAL_SYSTEM_BSPTYPE "pc-linux")
ADD_SUBDIRECTORY(path/to/osal)
```

The values for the variables can be obtained by any means, shown here is just a simplified example of how it can be done for a known target.

2.4.5 Cross compiling with Cmake

To cross compile, cmake uses a separate "toolchain file" that indicates the specific compiler to use, and any machine-specific compiler options that may be required. Documentation for the toolchain files is available on the cmake website at http://cmake.org. The toolchain file is specified when the build is provisioned. No OSAL build scripts need to be modified in order to cross compile or add extra machine-specific options.

2.5 Check over or customize the OSAL BSP directory

The glue logic that ties an OSAL application to a specific processor board and platform is in the src/bsp directory. This directory contains the BSP code, which contains all of the specific rules, glue code, and startup code to make an OSAL application run on a particular board with a particular OS.
The platforms supported in the OSAL distribution should run out of the box. They provide a starting point for a complete port to a new processor board.
This section will be expanded in the future to include information needed for new OSAL ports.

2.6 Configure one or more OSAL Applications

Once the OSAL is configured and ready to build, an OSAL application can be configured in the build directory. Multiple OSAL applications can be created in this directory. The application source code can come from the src/examples directory, or the applications can be contained completely within the build directory. The OSAL source distribution has a set of test and example applications in the src/examples and src/tests directories and a set of corresponding application directories and makefiles in build directory.

2.6.1 Configure a sample application in the build directory

The following show the files needed for a sample OSAL application in the build directory.

Sample OSAL Applications and the associated files

File	Description
build/examples/tasking-example	Directory for the included OSAL example Application.
build/examples/tasking-example/Makefile	Makefile for the example OSAL app. Because the source is in the src/examples/tasking-example directory, there is no need to include it here. The Makefile will find it using the OSAL_SRC environment variable. The source could be copied here in order to customize it.
build/examples/new_osal_app	Directory for a new OSAL application.
build/examples/new_osal_app/Makefile	Makefile for a new OSAL application.
build/examples/new_osal_app/new_osal_app.c	Source file for the new OSAL application.
build/examples/new_osal_app/new_osal_app.h	Header file for the new OSAL application.

The Application Makefiles have a specific format, so it is best to copy one of the application Makefiles from the build directory, such as build/examples/tasking-example.

2.6.2 Configure the application's main entry point

The OSAL development environment provides the main entry point/startup code for the Application. This code is located in the src/<bsp>/src directory. The startup code will call the Application's entry point which is named: void OS_Application_Startup(void)

2.7 Build the OSAL core and Applications

Once the OSAL Core and Applications are set up in a build directory, everything can be compiled. The OSAL Core or any of the Applications can be built from individual make files, or they can be built from the top-level Makefile in the build directory.

Build Commands

Shell command	Description
`$ cd build`	Change to the build directory.
`$ make`	Build the OSAL Core, and all Applications
`$ make clean`	Clean the OSAL Core, and all Applications

The following additional make targets apply only to the "classic" build; the cmake build handles configuration and dependencies automatically.

`$ make config`	Copy the osconfig.h for the BSP to the build directory
`$ cd examples/tasking-example; make`	Build the tasking-example Application only.
`$ make depend`	Recalculate the dependencies on the OSAL Core files and apps

Once the OSAL Applications are built, they are ready to load and execute on the target. The filename of the executable is dependent on the OS it is built for.

OSAL Application executable name

Target Operating System	Application executable name	Notes
vxWorks 6.x dynamic link	example1.elf	The vxWorks PowerPC platforms use a dynamically loaded object without the kernel.
Linux	example1.bin	
Rtems/Coldfire	example1.nxe	This is a static linked executable, linked with the RTEMS kernel and BSP.
RTEMS/SIS	example1.nxe	This is a static linked executable, linked with the RTEMS kernel and BSP.

2.8 Load and Run the OSAL Applications

Depending on the Target, it is usually straightforward to run an OSAL Application on a target platform. On desktop platforms, it is just a matter of running the executable program. On vxWorks, the example programs are loadable modules.

2.8.1 Load the OSAL Application Executable on the Target

On desktop targets the cFE Core can be run from the directory where it was compiled. On embedded targets, the Application has to be loaded into a remote file system, or booted over the network. On the vxWorks PowerPC targets, the Application can be loaded into the EEPROM or Flash disk after the vxWorks kernel is booted. On RTEMS targets, the Application can be loaded using the CEXP dynamic loader or it can be linked in with an RTEMS Binary. See the target specific sections for details on each platform.

2.8.2 Setup the Target File Systems

Because the OSAL runs on many different platforms, it must be able to deal with different file system types and different paths. The OSAL accomplishes this by using a file system abstraction. The abstracted OSAL file system is similar to a UNIX file system, where the root directory starts with "/" and all disks are mounted on directory trees. For example:

- /ram0/apps/ → RAM disk 0, apps subdirectory
- /ram1/data/ → RAM disk 1, data subdirectory
- /hd0/tables/ → Hard Disk 0, tables subdirectory

Using this abstraction, a file "datafile1.dat" on RAM disk 1 might be accessed from the OSAL by using the path "/ram1/data/datafile1.dat". Using the host vxWorks tools, the path to the same file would be: "RAM:0/data/datafile1.dat". If the OSAL is running on a Linux development workstation, the file might be located at: "/tmp/ramdev1/data/datafile1.dat". The important part is that the OSAL Application can access the files using a generic path, allowing the software to remain portable.

There are a few ways to map these host file systems to OSAL file systems:

- **Map existing target file systems to a OSAL path**. This is one of the most common ways to map the Non-Volatile disk to the OSAL. The OSAL relies on the target OS to create/mount a file system and it simply is given a mapping to the disk to allow the OSAL to access it.
- **Create EEPROM/Flash/ATA File systems**. The OSAL has the ability on some targets to format or initialize a EEPROM or ATA disk device. This is less commonly used.
- **Create RAM File Systems**. The OSAL can create RAM disks on the vxWorks targets. The OSAL will create or re-initialize the RAM disk for the vxWorks targets.

RTEMS Note: The RTEMS OS provides a base file system, called IMFS that provides the root directory. Because this closely matches what the OSAL file system abstraction provides, the RTEMS directories and filenames are a one to one mapping. In other words the path on RTEMS is the same as the path in the OSAL.

The following table shows examples of these file system mappings on various hosts. **Note** the change in the way the POSIX ports are mapped. Linux will no longer remove or create sub-directories based on the volume name. The path mapping for the FS_BASED option is now a simple mapping from an OSAL path to a host path. This makes the OSAL easier to use on linux platforms:

OSAL File system mapping

Target Operating system	cFE File system path	Target OS File system path	Notes
vxWorks 6.x	/ram	RAM:0/	Most vxWorks targets
	/cf	CF:0/ or CF:1/	MCP750
	/cf	EEP:0/	RAD750 target
Linux	/ram	./ram0	Note the "." This will map the RAM disk to the current working directory + the "ram0" subdirectory.
	/cf	./cf	Again, starts with the current working directory.
RTEMS	/ram	/ram	RTEMS has 1-1 mapping with the OSAL
	/cf	/cf	

2.8.3 Start the OSAL Application on the Target

Starting an OSAL Application is a highly target dependant activity. The following table gives examples of how to start an Application on various platforms. For full details see the notes for each section.

How to start an OSAL Application on Various Target Systems:

"Target" operating system	How to start the cFE
RTEMS / mcf5235	Loaded through GDB/BDM using a shell script: "debug.sh"
RTEMS / SIS	Loaded through GDB/SIS simulator: $ sparc-rtems4.10-gdb tasking-example.nxe (gdb) target sim (gdb) load (gdb) run
vxWorks 6.2 / RAD750	Started from the vxWorks Target Shell commands: Vx> ld < tasking-example.elf Vx> OS_BSPMain
Linux	Start directly from the linux shell: $ ./tasking-example.bin

Target Specific Instructions

This section provides details on how to load and run each of the supported OSAL configurations.

2.9 Generic PPC / vxWorks 6.4 Platform:

The Generic PPC applications will work on both the Motorola MCP750 and the BAE RAD750 running vxWorks 6.4. On this platform, the OSAL Applications are built as dynamic loadable vxWorks modules, rather than being linked to the vxWorks kernel/BSP. The OSAL Applications are loaded into the compact flash disk on the MCP750, so it can be started from a vxWorks shell or startup script after the kernel comes up.

2.9.1 OSAL Configuration for the Generic PPC / VxWorks 6.4

osal-config.mak Settings

osal-config.mak variable	Required selection	Notes
OS	vxworks6	
BSP	genppc-vxworks6.4	

2.9.2 File System Mappings on the MCP750 PPC Board

The cFE uses the following file system mappings for the MCP750 PPC Board. The file system mappings are defined in the bsp_voltab.c file in the src/arch/ppc/genppc/vxworks6.4/bsp directory:

OSAL File System Mappings

OSAL "device"	File System Type	OSAL Path	Host Path	Notes
/ramdev0	Real RAM Disk (vxWorks)	/ram	RAM:0/	
/eedev0	File System Mapped (FS_BASED)	/cf	eep:0/	This is the Compact Flash drive on the MCP750
/ramdev1 – /ramdev5	Real RAM Disk	N/A	N/A	Unused table entries for applications to create new RAM disks
/ssedev0 - /ssrdev2	File System Mapped (FS_BASED)	N/A	/ssr:0/SSR1 - /ssr:0/SSR3	Unused table entries for applications to map Hard Disk device directories to "pseudo" SSR file systems.

2.9.3 How to run the OSAL Applications on the MCP750 or RAD750

1. Load the kernel. The custom vxWorks kernel is loaded into the MCP750 via TFTP. We use a vxWorks boot image (Rather than the Motorola boot monitor/loader) to boot the MCP750 board, TFTP the "real" kernel to RAM, and execute it. This vxWorks boot image also sets the network settings for the "real" kernel image. On our OSAL/cFE development system, we keep the loadable vxWorks kernel image in a TFTP directory on the development workstation. So the vxWorks kernel image goes in /tftpboot/cpu1/cfecpu1.st. **($ cp /opt/workspace/mcp750image/default/vxWorks /tftpboot/cpu1/cfecpu1.st)**

2. Copy the “example1.elf” (or other executable name) loadable module into the non-volatile disk. On the MCP750, this is done simply by FTPing the tasking-example.elf file to the target:

```
$ ftp 192.168.1.4
 ftp> username: target
 ftp> password: password
 ftp>  cd “CF:0”
ftp> binary
ftp> put tasking-example.elf
```

3. Load the example Application in the vxWorks shell:

```
vx> cd “CF:0”
vx> ld < tasking-example.elf
```

4. Run the example Application in the vxWorks shell:

```
vx> OS_BSPMain
```

(The entry point for the examples and test programs is always OS_BSPMain)

2.10 Axiom M5235 BCC / RTEMS 4.10:

The OSAL supports the Axiom 5235 BCC single board computer with an RTEMS 4.10 board support package. The 4.10.2 version of RTEMS was used (as of December 2012) along with the RTEMS 4.10 compiler for the m68k/coldfire. The tests and examples are built as static RTEMS executable programs for the board and can be loaded using the DBUG monitor or BDM port. When developing for RTEMS, the libraries and BSP code is usually located in /opt/rtems-4.10. The OSAL Makefiles use an environment variable "RTEMS_BSP_BASE" to determine where the RTEMS libraries and BSPs are installed. This variable is set from the "setvars.sh" file along with an example of how to set the variable..

2.10.1 OSAL Configuration for the Axiom M5235 BCC / RTEMS 4.10

osal-config.mak Settings

osal-config.mak variable	Required selection	Notes
OS	rtems	
BSP	rtems-mcf5235	

2.10.2 File System Mappings on the Axiom M5235 BCC / RTEMS 4.10

The RTEMS port of the OSAL has a one to one file system mapping. The OSAL RAM disk will format an RTEMS NVRAM disk with the RFS file system.The file system mappings are defined in the bsp_voltab.c file in the src/bsp/mcf5235-rtems/src directory:

OSAL File System Mappings

OSAL "device"	File System Type	OSAL Path	Host Path	Notes
/ramdev0	RAM_DISK (NVRAM/RFS)	/ram	/ram	Mapped to the IMFS root directory
/eedev0	File System Mapped (FS_BASED)	/cf	/cf	Mapped to the IMFS root directory
/ramdev1 – /ramdev5	Unused	N/A	N/A	Unused table entries for applications to create new RAM disks. RTEMS does not currently have support for creating new RAM disks.
/ssedev0 - /ssrdev2	File System Mapped (FS_BASED)	N/A	N/A	Unused table entries for applications to map Hard Disk device directories to "pseudo" SSR file systems.

2.10.3 How to run the OSAL Applications on the Axiom M5235 BCC with RTEMS 4.10

When the example application and test programs are all built as static executables for the M5235BCC board. The example programs can be loaded in the following ways:
Using the BDM port through the GNU debugger. If the board is connected to the host PC with a BDM debugger cable, then the example programs can be loaded and run from there. For our environment we use the Gnu Debugger that was included with the RTEMS 4.10 tools and the m68k-bdm-gdbserver from the BDM Tools project: http://bdmtools.sourceforge.net . The gdb-init script and a debug.sh file are included in the src/bsp/mcf5235-rtems/bsp/rtems-support directories. The debug.sh script gives the proper command line to load the application to the board using the GDB debugger and BDM interface.

2.11 SPARC SIS Simulator / RTEMS 4.10:

The OSAL supports the SPARC SIS simulator built into GDB with the sis RTEMS 4.10 board support package. The 4.10.1 version of RTEMS was used (as of December 2011) along with the RTEMS 4.10 compiler for the sparc. The tests and examples are built as static RTEMS executable programs for the simulator. The OSAL Makefiles use an environment variable "RTEMS_BSP_BASE" to determine where the RTEMS libraries and BSPs are installed. This variable is set from the "setvars.sh" file along with an example of how to set the variable.

2.11.1 OSAL Configuration for the SPARC SIS Simulator / RTEMS 4.10

osal-config.mak Settings

osal-config.mak variable	Required selection	Notes
OS	rtems	
BSP	sis-rtems	

2.11.2 File System Mappings on the SPARC SIS Simulator / RTEMS 4.10

The RTEMS port of the OSAL has a one to one file system mapping. The OSAL RAM disk will format an RTEMS NVRAM disk with the RFS file system. The file system mappings are defined in the bsp_voltab.c file in the src/bsp/sis-rtems/src directory:

OSAL File System Mappings

OSAL "device"	File System Type	OSAL Path	Host Path	Notes
/ramdev0	RAM_DISK (NVRAM/RFS)	/ram	/ram	Mapped to the IMFS root directory
/eedev0	File System Mapped (FS_BASED)	/cf	/cf	Mapped to the IMFS root directory
/ramdev1 – /ramdev5	Unused	N/A	N/A	Unused table entries for applications to create new RAM disks. RTEMS does not currently have support for creating new RAM disks.
/ssedev0 - /ssrdev2	File System Mapped (FS_BASED)	N/A	N/A	Unused table entries for applications to map Hard Disk device directories to "pseudo" SSR file systems.

2.11.3 How to run the OSAL Applications on the SPARC SIS Simulator with RTEMS 4.10

When the example application and test programs are all built as static executables for the SIS Simulator built into the sparc-rtems4.10-gdb executable.
To run an example or test, simply do the following:

```
$ sparc-rtems4.10-gdb tasking-example.nxe
(gdb) target sim
(gdb) load
(gdb) run
```

When you are finished running/debugging, hit <ctrl>-c and quit the debugger.

2.12 PC / Linux Platform

The OSAL can run on linux distributions. Testing is done with CentOS 6.5 **32 bit** and Ubuntu 13.10 64 bit. Newer versions of the Linux 2.6 kernel have POSIX message queues, which can be used for the OSAL Queue implementation. If the POSIX message queues are not available, then the OSAL Queues can use UDP sockets. (see the OS_SOCKET_QUEUE configuration parameter). In general, the older versions of linux (2.4 kernel) are not supported, but most modern Linux releases such as Ubuntu 12.10 and later, and Cent OS/Redhat Enterprise Linux 5 and later should work. The OSAL has also been used on the Raspberry Pi computer with the Raspbian Debian based linux. The OSAL is primarily run on 32 bit linux distributions, but it should compile and run as a 32 bit application on 64 bit linux.

2.12.1 OSAL Configuration for the PC / Linux Platform

osal-config.mak Settings

Prolog.mak variable	Required selection	Notes
OS	posix	
BSP	pc-linux	

Additional configuration notes:
To enable the POSIX message queues, make sure the OS_SOCKET_QUEUE parameter is not defined in osconfig.h.
If the OS_SOCKET_QUEUE option is not used, the OSAL will use POSIX message queues. If POSIX message queues are used, your application may need to run as root in order to create the queues you need. There are kernel parameters that can be adjusted in order to avoid running as root.

2.12.2 How to Run the OSAL on the PC / Linux Platform

1. To run an OSAL Application, simply execute the binary from a shell prompt:

build/examples/tasking-example]$ ./tasking-example.bin

3 OSAL Unit Tests

The OSAL distribution includes a suite of black box unit tests, white box unit tests (/src/unit-test-coverage), and functional tests (/src/tests). Tam Ngo at NASA Johnson Space Flight Center developed the suite of black box unit tests located in the /src/unit_tests directory. This section describes how to build and run the suite of black box unit tests using the "classic" build. Note: it is assumed the steps in Section 2 How to Configure, Build, and Run the OSAL have been followed to setup the build environment for the "classic" build. The unit tests run on Linux, but could also be configured to run on other operating systems (and will in future releases).
Currently the osprintf tests are not compiled and run, and the ARINC 653 tests are not used. A future release of the OSAL will contain support for the ARINC653 operating system API.

3.1.1 OSAL Unit Test Configuration for the PC / Linux Platform

osal-config.mak Settings

Prolog.mak variable	Required selection	Notes
OS	posix	Unit tests are for the POSIX port
BSP	pc-linux-ut	This is a special BSP for the unit tests

3.1.2 How to Run the OSAL Unit Tests on the PC / Linux Platform

To build and run the OSAL unit tests, run the following commands from the "build" directory:

Build Commands

Shell command	Description
`$ cd build`	Change to the build directory.
`$ make clean-unit-tests`	Clean the OSAL Core, and all Applications
`$ make config`	Copy the osconfig.h for the BSP to the build directory
`$ make unit-tests`	Build all of the unit tests.
`$ make gcov`	Run the unit tests, collecting code coverage information.

Once the unit tests have run, the log files are available in the individual unit test binary directories

OSAL Unit Test Directories

Directory	Description
build	The top level OSAL build directory.
build/unit-tests	The top level unit test build directory.
build/unit-tests/oscore-test	Contains the test binary, log, and gcov files for the OSAL core tests
build/unit-tests/osfile-test	Contains the test binary, log, and gcov files for the OSAL file API tests
build/unit-tests/osfilesys-test	Contains the test binary, log, and gcov files for the OSAL file system tests
build/unit-tests/osloader-test	Contains the test binary, log, and gcov files for the OSAL loader tests
build/unit-tests/ostimer-test	Contains the test binary, log, and gcov files for the OSAL timer tests