|  |
| --- |
|  |
| OSAL Reference Manual |
| This publication is protected by Federal Copyright Law, with all rights reserved. No part of this publication may be reproduced, transcribed, translated, or transmitted without the prior explicit written permission from D2 Technologies, Inc.  The information contained in this document is subject to change without notice. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Document History** | | | | |
| **Date** | **Software Version** | **Document Version** | **Initials** | **Changes** |
| 10/04/04 | 1.0.0.0 | 1.3.0.0 | BC | Initial Release |
| 01/06/05 | 1.0.0.0 | 1.3.1.0 | BC | Miscellaneous font formatting; addition of memory monitor and updated error codes. |
| 3/23/05 | OSAL 1.4 | 1.4 | BC | Added kernel / user space IPC (in Section 5.5) |
| 03/30/07 | vPort 1.3.0 | 1.5 | JC | Update for vPort 1.3.0 Specification |
| 01/10/08 | vPort 1.4 | 1.5 | KR | Clean up in Bug 2546 |
| 11/18/08  1/5/08  1/9/08  8/14/09 | vPort 1.5  vPort 1.5  vPort 1.5  vPort 1.5 | 1.5  1.5  1.5  1.5 | CD  CD  CD  CD | Added OSAL Unit Tests (Section 6)  Updated info on Unit Tests  Updated info on Unit Tests  Added descriptions for conditional application exiting functions |
| 10/22/2012 | vPort 2.0 | 2.0 | SP | Updated Document with interfaces descriptions for IpV6, File System access, TLS. |
| 12/21/2012 | vPort 2.0 | 2.1 | CG | Update for OSAL\_TimeLocal |
| 8/15/2013 | vPort2.0 | 2.2 | SP | Added ipsec, crypto, OSAL\_fileExists() |
| 7/16/2014 | vPort2.0 | 2.3 | IL | Added argument description for OSAL\_memAlloc() and OSAL\_memFree().  Add API OSAL\_memMove() and OSAL\_memRealloc() |
| 8/12/2014 | vPort2.0 | 2.4 | DC | Update OSAL\_msgQCreate() and OSAL\_msgQGrpDelete() |

Table of Contents

[OSAL Reference Manual 1](#_Toc395604289)

[1 Introduction 8](#_Toc395604290)

[1.1 Scope 9](#_Toc395604291)

[1.2 Audience 9](#_Toc395604292)

[1.3 Related Documents 9](#_Toc395604293)

[1.4 Glossary 10](#_Toc395604294)

[2 Protective and Non-Protective Operating System Requirements 11](#_Toc395604295)

[2.1 Standard Libraries 12](#_Toc395604296)

[*3* Modules Requirements 13](#_Toc395604297)

[3.1 User-mode versus Kernel Mode 13](#_Toc395604298)

[3.2 Standard Function Libraries 13](#_Toc395604299)

[3.3 Hardware Architecture 13](#_Toc395604300)

[3.4 Wall Clock Time 13](#_Toc395604301)

[3.5 System Log 14](#_Toc395604302)

[3.6 Blocking and Non-blocking Timeouts 14](#_Toc395604303)

[3.7 Arbitrary Timeouts 14](#_Toc395604304)

[3.7.1 Timeout Example: Delay Rounded-Down 15](#_Toc395604305)

[3.7.2 Timeout Example: Delay Rounded-Up 15](#_Toc395604306)

[3.8 Use of Non-OSAL System Calls 15](#_Toc395604307)

[3.9 Target Operating System Task Cleanup Assumptions 16](#_Toc395604308)

[4 OSAL Function Interfaces 17](#_Toc395604309)

[4.1 Conditional Compilation 17](#_Toc395604310)

[4.2 OSAL Standard Include Files 17](#_Toc395604311)

[4.3 Types 17](#_Toc395604312)

[4.4 Blocking and Non-Blocking Timeout Constants 18](#_Toc395604313)

[4.5 Module ID Constants 18](#_Toc395604314)

[4.6 Message ID Constants 19](#_Toc395604315)

[4.7 Tasks 20](#_Toc395604316)

[4.7.1 Task Creation 21](#_Toc395604317)

[4.7.2 Task Deletion 22](#_Toc395604318)

[4.7.3 Task Delay 22](#_Toc395604319)

[4.8 Inter-Process Communication 23](#_Toc395604320)

[4.8.1 Message Queue Creation 24](#_Toc395604321)

[4.8.2 Message Queue Deletion 25](#_Toc395604322)

[4.8.3 Message Queue Send 25](#_Toc395604323)

[4.8.4 Message Queue Receive 26](#_Toc395604324)

[4.8.5 Message Queue Group Creation 27](#_Toc395604325)

[4.8.6 Message Queue Group Deletion 27](#_Toc395604326)

[4.8.7 Message Queue Group Add Queue 28](#_Toc395604327)

[4.8.8 Message Queue Group Receive 28](#_Toc395604328)

[4.9 Network Access 29](#_Toc395604329)

[4.9.1 Socket Setup 31](#_Toc395604330)

[4.9.2 Socket Communication 31](#_Toc395604331)

[4.9.3 Socket Removal 32](#_Toc395604332)

[4.9.4 Socket Control 32](#_Toc395604333)

[4.9.5 SSL Functions 32](#_Toc395604334)

[4.9.6 Network Address Helpers 33](#_Toc395604335)

[4.9.7 Network Address Conversions 34](#_Toc395604336)

[4.10 Encryption 36](#_Toc395604337)

[4.11 IPSec 38](#_Toc395604338)

[4.12 File I/O 41](#_Toc395604339)

[4.13 Semaphores 41](#_Toc395604340)

[***4.13.1*** Semaphore Creation 42](#_Toc395604341)

[4.13.2 Semaphore Deletion 43](#_Toc395604342)

[4.13.3 Semaphore Acquisition 43](#_Toc395604343)

[4.13.4 Semaphore Giving 44](#_Toc395604344)

[4.13.5 Conditional Application Exiting 44](#_Toc395604345)

[4.14 Timer Management 46](#_Toc395604346)

[4.14.1 Timer Creation 46](#_Toc395604347)

[4.14.2 Timer Deletion 47](#_Toc395604348)

[4.14.3 Timer Status – One-Shot and Periodic Timer Stop 47](#_Toc395604349)

[4.14.4 Timer Control – One-Shot Timer Start 47](#_Toc395604350)

[4.14.5 Timer Control – Periodic Timer Start 48](#_Toc395604351)

[4.14.6 Timer Control – One-Shot and Periodic Timer Stop 49](#_Toc395604352)

[4.14.7 Timer Control – One-Shot and Periodic Timer Interupt 49](#_Toc395604353)

[4.15 Time Retrieval 50](#_Toc395604354)

[4.15.1 Time Since Epoch Fine 50](#_Toc395604355)

[4.15.2 Time Since Epoch Formatted as ISO 8601 50](#_Toc395604356)

[4.15.3 Time Local 50](#_Toc395604357)

[4.16 Memory Management 51](#_Toc395604358)

[4.16.1 Memory Allocation 51](#_Toc395604359)

[4.16.2 Memory De-allocation 52](#_Toc395604360)

[4.16.3 Memory Copying, Comparing, Initializing, Move, Reallocate 53](#_Toc395604361)

[4.17 Memory Monitor 54](#_Toc395604362)

[4.18 Portable Message Logging 54](#_Toc395604363)

[5 Interrupt Context Requirements 55](#_Toc395604364)

[6 Running The OSAL Unit Tests 56](#_Toc395604365)

[6.1 Test Setup 56](#_Toc395604366)

[6.2 Test Commands 56](#_Toc395604367)

[6.3 Step by Step Instructions for Running the Tests 57](#_Toc395604368)

Table of Figures

[Figure 1: Modules 8](#_Toc395604369)

[Figure 2: Modules and Domains 11](#_Toc395604370)

[Figure 3: Opearting System Standard Library Integration 12](#_Toc395604371)

[Figure 4: Single Producer, Multiple Consumer Queue Model 23](#_Toc395604372)

Table of Tables

No table of figures entries found.

Table of Code Examples

Code Example 1: Portable Task Creation 21

Code Example 2: OSAL Task Priorities 22

Code Example 3: Task Deletion 22

Code Example 4: Task Delay 22

Code Example 5: Message Queue Creation 25

Code Example 6: Message Queue Deletion 25

Code Example 7: Message Queue Send 26

Code Example 8: Message Queue Receive 27

Code Example 9: Message Queue Group Deletion 28

Code Example 10: Message Queue Group Receive 29

Code Example 11: Network Socket Data Structures 31

Code Example 14: Counting Semaphore Creation 42

Code Example 15: Mutex Semaphore Creation 42

Code Example 16: Semaphore Deletion 43

Code Example 17: Semaphore Acquisition 44

Code Example 18: Semaphore Release 44

Code Example 19: Registering Conditional Application Exiting 45

Code Example 20: Conditional Application Exiting 45

Code Example 21: Unregistering Conditional Application Exiting 46

Code Example 22: Timer Creation 46

Code Example 23: Timer Deletion 47

Code Example 24: Starting a Timer 48

Code Example 25: Starting a Periodic Timer 48

Code Example 26: Stopping (Canceling) a Timer 49

Code Example 28: Allocating Local Memory 51

Code Example 29: Allocating Array (Cleared) Memory 52

Code Example 30: Memory De-allocation 53

Code Example 31: Message Logging 54

# Introduction

D2 Technologies develops Internet communication software modules that may be ported to different platforms. Modules may be independently linked or integrated together to produce a set of communication services.

Each platform is comprised of processor and hardware architectures that may include: 64-bit, 32-bit and 16-bit microprocessors, and either Real Time Operating Systems or protected-mode (Unix) operating systems.

Each module provides a public software interface and integrates with other software interfaces. The figure below illustrates a conceptual view of Modules.

Fig1

Figure 1: Modules

When creating Modules, D2 software is designed to run across many platforms. This requires platform abstraction, termed the Operating System Abstraction Layer (OSAL). In this context, the Operating System may be a Real Time Operating System (RTOS) or a general purpose communications operating system (like Unix) with specific real time enhancements. OSAL hides the underlying function calls of the native RTOS, thus helping to achieve OS-independence in modules.

OSAL does not abstract every operating system call. OSAL leverages the availability of common standard libraries and abstracts platform-specific functions. It assumes the following is available from every Real Time Operating Sysytem (RTOS):

* a task synchronization method (such as VxWorks or Linux kernel semaphores, or Pthreads conditional variables);
* a means of scheduling tasks/tasklets;
* a memory package;
* a time facility.
* An IP Stack

OSAL may leverage existing RTOS facilities for its inter-process communication (IPC) or create new functions that conform to the software interfaces defined in this specification.

## Scope

This document specifies the system, library, and service interface functions which Modules uses. This specification must be followed by both D2 engineers as well as customers when porting D2 modules to their platform. It also discusses the generic management services that D2 portable products provide to their upper layers.

This document presents an overview of Modules in protective and non-protective environments, discusses assumptions made by this portable methodology, and presents a generic software environment.

This document details the system function calls that OSAL provides to Modules and describes the standard libraries and the OSAL-provisioned libraries that modules may use.

## Audience

This document is intended for programmers who are employees of D2 Technologies and who write Modules. It is also intended to provide guidance to D2 Technologies customers who have a non-standard RTOS and who desire to provide the System and Library Abstraction Layers themselves.

## Related Documents

This document is related to the following D2 Technologies documents:

|  |
| --- |
| *D2 C Coding Standard,* 6/20/2004 |

## Glossary

**Modules** – Code written by D2 Technologies that is designed to be operating system independent and which has well-defined application programmer interfaces in order to provide a service to an upper layer. It may also use the services of a lower layer or **OSAL**. **OSAL** is a D2-proprietary OS-independent software interface that modules invoke for system services.

**OSAL** – the **O**perating **S**ystem **A**bstraction **L**ayer. All OSAL elements are preceded with this acronym.

# Protective and Non-Protective Operating System Requirements

Linux and Unix derivatives are protective-mode operating systems; they have different requirements to a real-time OS which favors timely execution over protection. Therefore, specifying abstractions that work equally well in a diverse range of platforms such as Linux, Nucleus and VxWorks is somewhat challenging.

A protective OS is predominantly characterized by being highly layered, a native system call interface that traps into trusted kernel mode from un-trusted user space, and where users are considered unprivileged. A real-time operating system's principal characteristics are shared memory space for efficient operation and are usually very flatly structured systems with a monolithic (as opposed to distributed) micro-kernel at the core.

The OSAL specification for Modules suffices for both protective and non-protective kernels, with constraints.

In protective-mode operating systems, software either resides in either kernel space or user space. A single software module may not cross the user/kernel boundary. Software which needs to communicate between kernel space and user space uses a specific software interface. In Linux, this requires a Linux Kernel Module (usually used to implement a driver). This is shown in the figure below.

Fig2

Figure 2: Modules and Domains

OSAL contains functions which provide for communication between portable modules in user space and portable modules in kernel space.

The "User's Glue code" which allows the portable module to link or dynamically load into the kernel (for example with GNU/Linux insmod, which uses f\_ops interfaces) is not part of the OSAL specification.

## Standard Libraries

The figure below illustrates how the operating system software libraries interface with Modules and OSAL.

Fig3

Figure 3: Opearting System Standard Library Integration

# Modules Requirements

## User-mode versus Kernel Mode

Modules using OSAL may run in either user or kernel space; OSAL abstracts the module from the run-time environment.

## Standard Function Libraries

Modules uses system-standard libraries (such as ANSI libc, and so on). This specification assumes the standard libraries exist on the platform with the standards-based implementation; i.e., they do not need abstraction from platform-specific implementation.

## Hardware Architecture

Modules may run on either 16-bit, 32-bit, or 64-bit hardware. OSAL abstracts numerical types and other hardware-specific software interfaces.

## Wall Clock Time

ANSI Time is not assumed to exist in any environment. In Nucleus, only elapsed time since boot is available from the operating system. Time is derived from external sources such as via the SNTP protocol. OSAL does not specify abstraction for Time.

## System Log

A system logger such as syslog() is not assumed to exist in every environment. In its place, there is a OSAL\_logMessage facility that allows for module-selectable information to be conveyed to a user.

## Blocking and Non-blocking Timeouts

OSAL abstracts blocking and non-blocking timeouts for some functions.

If the timeout is non-blocking (OSAL\_NO\_WAIT), the OSAL function is performed immediately.

Note that when no blocking is requested, the underlying operating system may still invoke the scheduler.

If the timeout is OSAL\_WAIT\_FOREVER, the OSAL function blocks permanently until the request can be satisfied.

Other timeout values are specified in milliseconds and are handled according to the OSAL timer specification below.

## Arbitrary Timeouts

Modules may have time-sensitive software interfaces. OSAL requires the operating system to provide a timer for scheduling, that the timer have a periodic time of 10 milliseconds or less, and that any system timeout value does not differ +/- 10ms from the timeout requested. OSAL specifies timeouts for some software functions, in units of milliseconds, and scales the value to a 10 millisecond resolution.

For example, if a user requests a timing delay of 50 milliseconds, then as long as the operating system generates the timer event in the range of 40 to 60 milliseconds, the modules will function properly.

OSAL uses the following equation in order to convert user-specified milliseconds into OS-specific clock ticks:

clocktics

This calculation performs a time ceiling. The millisecond delay requested will be quantized by OSAL based upon the resolution of the operating system timer. The system will block at a minimum until the next operating system tick. Otherwise, the number of milliseconds will be rounded up to the nearest operating system tick.

### Timeout Example: Delay Rounded-Down

A module calls an OSAL function with a blocking timeout of 3 ms. If the granularity of the system timer is 10 ms, and the next system clock tick is 1 ms away from occurring, then the task calling the blocking message send will pend for osTicks=1 which is 1 ms instead of the 3 that was requested. The OSAL timing requirement is +/- 10 ms and the result of 1 ms is within +/- 10 ms of the requested 3 ms time period.

This is illustrated in the following diagram:

time0

### Timeout Example: Delay Rounded-Up

A module calls an OSAL function with a blocking timeout of 11 ms. The next operating system tick will occur in 9 ms. In this situation, the conversion between milliseconds and clock ticks produces osTick = 209/100 = 2. This means the system will pend for the next 2 system ticks which are 19 ms from the call. The following diagram illustrates this.

time1

## Use of Non-OSAL System Calls

Modules must not use operating system function calls.

Modules must avoid all use of operating system calls other than ANSI Lib-C standard library calls. Any other function calls make the software non-portable.

OSAL may provide abstraction for common C-library function calls in the future, to remove all non-OSAL external dependencies from Modules.

## Target Operating System Task Cleanup Assumptions

In a protected mode operating system, the operating system usually cleans up all elements allocated and created by a task when the task exits or takes an exception. In a non-protected OS, this frequently is not the case and resources (such as memory and sockets) are not freed.

Modules do not assume that clean up occurs automatically. Modules provide A module-specific shutdown software interface which cleans up all memory, closes all sockets, deletes any message queues and semaphores, and otherwise removes resources that would otherwise cause a system resource leak.

The intent of the shutdown function is to allow management code to facilitate shutdown in an orderly manner if they so choose.

# OSAL Function Interfaces

This section defines the software interface that modules may use and which third parties must provide when porting Modules to a proprietary RTOS.

## Conditional Compilation

In rare cases, conditional compilation is required to make modules portable across platforms. Modules may conditionally compile code fragments for specific systems. The following #defines may be used throughout Modules to create conditionally compiled code.

Modules assume that only single #define is valid; others must be undefined. Operating system-specific code such as initialization or shutdown is conditionally compiled using these #defines. See OSAL\_taskCreate() for an example.

OSAL\_NUCLEUS for Nucleus

OSAL\_VXWORKS for Wind River VxWorks

OSAL\_PTHREAD for Unix Pthread Environments

OSAL\_LKM for Linux Loadable Kernel Environments

OSAL\_ECOS for ECOS Environments

OSAL\_RTK for RTK Environments

NOTE: Conditional compilation is an older method and should no longer be used.

## OSAL Standard Include Files

OSAL public header files do not include any standard C header files.

## Types

These types are defined in the file: osal\_types.h

typedef enum OSAL\_Status {

OSAL\_FAIL = 0,

OSAL\_SUCCESS = 1

} OSAL\_Status;

typedef enum OSAL\_Boolean {

OSAL\_FALSE = 0,

OSAL\_TRUE = 1

} OSAL\_Boolean;

typedef void (\*OSAL\_FuncPtr)(...); /\* Void Function Ptr \*/

Modules uses the following type values:

vint this type holds a variable-width integer (the native int size)

uvint this type holds a variable-width unsigned integer (the native unsigned int

size)

uint8 this type holds precisely an unsigned 8-bit quantity

uint16 this type holds precisely an unsigned 16-bit quantity

uint32 this type holds precisely an unsigned 32-bit quantity

uint64 this type holds precisely an unsigned 64-bit quantity

int8 this type holds precisely a signed 8-bit quantity

int16 this type holds precisely a signed 16-bit quantity

int32 this type holds precisely a signed 32-bit quantity

int64 this type holds precisely a signed 64-bit quantity

Modules do not include “int” or “unsigned” in it’s Modules interface. If the portable module does not care about the width of the integer or unsigned, then the module uses “vint” for “int” and “uvint” for “unsigned” or “unsigned int”. The only exception to this is when A module is accessing a third-party software interface which specifically requires these types.

## Blocking and Non-Blocking Timeout Constants

For time-based routines, the following two definitions are available:

#define OSAL\_NO\_WAIT 0

#define OSAL\_WAIT\_FOREVER (~0) /\* portable to 16-bit architectures \*/

## Module ID Constants

This define all vport modules ID in osal\_constant.h. This is used in OSAL\_msgQCreate()

typedef enum {

OSAL\_MODULE\_NONE,

OSAL\_MODULE\_REV\_ONLY,

OSAL\_MODULE\_AUDIO\_VE,

OSAL\_MODULE\_VIDEO\_VE,

OSAL\_MODULE\_CSM\_PUBLIC, /\* Other modules to CSM and CSM output event \*/

OSAL\_MODULE\_CSM\_PRIVATE, /\* CSM internal \*/

OSAL\_MODULE\_ISIM,

OSAL\_MODULE\_MGT,

OSAL\_MODULE\_ISI,

OSAL\_MODULE\_ISI\_MSG\_RECV,

OSAL\_MODULE\_ISI\_RPC\_CLIENT,

OSAL\_MODULE\_ISI\_RPC\_SERVER,

OSAL\_MODULE\_MC,

OSAL\_MODULE\_GAPP,

OSAL\_MODULE\_GAPP\_GSM,

OSAL\_MODULE\_SAPP,

OSAL\_MODULE\_MSRP,

OSAL\_MODULE\_RIR,

OSAL\_MODULE\_SR,

OSAL\_MODULE\_VIER,

OSAL\_MODULE\_VPR,

OSAL\_MODULE\_VPR\_KERNEL,

OSAL\_MODULE\_SIP,

OSAL\_MODULE\_GBAM,

OSAL\_MODULE\_SIT,

OSAL\_MODULE\_XCAP,

OSAL\_MODULE\_VAPP,

OSAL\_MODULE\_CSM\_UT\_CLI,

OSAL\_MODULE\_OSAL\_UT,

OSAL\_MODULE\_TAPP,

OSAL\_MODULE\_MM,

OSAL\_MODULE\_MM\_UT,

} OSAL\_MODULE\_ID;

## Message ID Constants

This define all vport message data struture ID in osal\_constant.h. This is used in OSAL\_msgQCreate()

typedef enum {

OSAL\_DATA\_STRUCT\_UNKONOWN,

OSAL\_DATA\_STRUCT\_CSM\_InputEvent,

OSAL\_DATA\_STRUCT\_CSM\_PrivateInputEvt,

OSAL\_DATA\_STRUCT\_CSM\_OutputEvent,

OSAL\_DATA\_STRUCT\_ISIP\_Message,

OSAL\_DATA\_STRUCT\_ISI\_EventMessage,

OSAL\_DATA\_STRUCT\_tSipIpcMsg,

OSAL\_DATA\_STRUCT\_SAPP\_TmrEvent,

OSAL\_DATA\_STRUCT\_MSRP\_CmdTaskTmr,

OSAL\_DATA\_STRUCT\_MSRP\_Event,

OSAL\_DATA\_STRUCT\_MSRP\_Cmd,

OSAL\_DATA\_STRUCT\_MSRP\_CmdHdr,

OSAL\_DATA\_STRUCT\_GSM\_Event,

OSAL\_DATA\_STRUCT\_ISI\_ServerEvent,

OSAL\_DATA\_STRUCT\_ISI\_DATA,

OSAL\_DATA\_STRUCT\_XCAP\_Cmd,

OSAL\_DATA\_STRUCT\_XCAP\_Evt,

OSAL\_DATA\_STRUCT\_VPR\_Comm,

OSAL\_DATA\_STRUCT\_RIR\_EventMsg,

OSAL\_DATA\_STRUCT\_\_VTSP\_CmdMsg,

OSAL\_DATA\_STRUCT\_VTSP\_EventMsg,

OSAL\_DATA\_STRUCT\_\_VTSP\_RtcpCmdMsg,

OSAL\_DATA\_STRUCT\_\_VTSP\_RtcpEventMsg,

OSAL\_DATA\_STRUCT\_\_VTSP\_CIDData,

OSAL\_DATA\_STRUCT\_\_VTSP\_FlowMsg,

OSAL\_DATA\_STRUCT\_\_VTSPR\_MultiPktMsg,

OSAL\_DATA\_STRUCT\_\_VTSPR\_MultiDecodedMsg,

OSAL\_DATA\_STRUCT\_\_VTSPR\_MultiRawMsg,

OSAL\_DATA\_STRUCT\_\_VTSPR\_MultiCodedMsg,

OSAL\_DATA\_STRUCT\_VTSP\_Stun,

OSAL\_DATA\_STRUCT\_\_VC\_AppEventMsg,

OSAL\_DATA\_STRUCT\_JVHW\_Command,

OSAL\_DATA\_STRUCT\_JVHW\_PCM\_BUFFER,

OSAL\_DATA\_STRUCT\_JVHW\_Message,

OSAL\_DATA\_STRUCT\_VAPP\_Msg,

OSAL\_DATA\_STRUCT\_VAPP\_MgmtEvent,

OSAL\_DATA\_STRUCT\_SIT\_QueueMsg,

OSAL\_DATA\_STRUCT\_GBA\_Command,

OSAL\_DATA\_STRUCT\_GBA\_Event,

OSAL\_DATA\_STRUCT\_MM\_UtMsg,

OSAL\_DATA\_STRUCT\_String,

OSAL\_DATA\_STRUCT\_int,

} OSAL\_DATA\_STRUCT\_ID;

## Tasks

As part of the OSAL software interface, modules are given the ability to create and delete tasks, suspend them, and perform various management on tasks. OSAL also gives the user the ability to monitor for the demise of a specific task by registering a handler to catch OSAL\_taskExit() invocations from a OSAL-created task as well as an indication handler to receive indications that a OSAL-created task has taken an exception.

If task exit callbacks are registered, an exit code is specified on the OSAL\_taskExit() and this is conveyed to the callback in order to determine what to do. If an exception is caught, a reason is specified along with system-specific information such as the address of the instruction that generated the exception, the exception type etc.

Protected-mode OS generally cleanup a task and its runtime-allocated resources whenever an exception/abort/or exit is invoked from that task’s context. Un-protected mode systems generally move the task to a SUSPEND state for a post-mortem analysis using development tools. All resources in un-protected mode systems are usually left dangling. So the exception handler or the task exit handler may be an appropriate point at which to place clean-up code with conditionally compiled RTOS-specific procedures. The sections below on Task Exit, Task Exit Callback Registration and Deregistration as well as the section on Shutdown provide more details on recommended procedures during un-anticipated events such as these.

### Task Creation

Modules may call this routine in order to create a task for a specific purpose.

***Function Binding***:

OSAL\_TaskId OSAL\_taskCreate(name\_ptr,priority,stackByteSize,entryPt,arg0)

***Parameters***:

char \*name\_ptr - C-string uniquely identifying this task

OSAL\_TaskPrio priority - priority of the task to be created.

uint32 stackByteSize - size of task stack to be allocated

OSAL\_TaskPtr fx\_ptr - identifier of C routine that is the main entry for the task

OSAL\_TaskArg arg0 - optional argument to the task (entryPt)(arg0)

***Returns***:

OSAL\_TaskId taskId - which is to be used for subsequent task-related calls. On success, this is non-zero, on failure, this is NULL.

***Code Sample***:

myTaskId = OSAL\_taskCreate(“tVe\_Proc”, OSAL\_TASK\_PRIO\_APP, (uint32) 10000, ve\_process, /\* unused arg0:\*/ NULL);

if(NULL == myTaskId) {

…

Code Example 1: Portable Task Creation

Modules need to take care with priority and stack size. On some RTOS, the user should allow for interrupt context in the user’s stack. If nested interrupts are allowed (and most RTOS allow them), then the user must allow for the task use plus space for the maximum interrupt nesting stack requirements. In other RTOS, there may be a separate interrupt stack and if there is, then this might not be an issue. (These are precisely the reasons why RTOS abstraction leads inherently to non-Modules.)

Priority levels are defined as type OSAL\_TaskPrio, and are mapped to RTOS values based on specific tasks. The priority levels are as follows:

OSAL\_TASK\_PRIO\_VTSPR – For use when creating vtspr task

OSAL\_TASK\_PRIO\_DEC10 – For use when creating 10ms decode task

OSAL\_TASK\_PRIO\_DEC20 – For use when creating 20ms decode task

OSAL\_TASK\_PRIO\_DEC30 – For use when creating 30ms decode task

OSAL\_TASK\_PRIO\_ENC10 – For use when creating 10ms encode task

OSAL\_TASK\_PRIO\_ENC20 – For use when creating 20ms encode task

OSAL\_TASK\_PRIO\_ENC30 – For use when creating 30ms encode task

Code Example 2: OSAL Task Priorities

### Task Deletion

Modules may call this routine in order to delete a task.

***Function Binding***:

OSAL\_Status OSAL\_taskDelete(OSAL\_TaskId taskId)

***Parameters***:

OSAL\_TaskId taskId - the id returned from OSAL\_taskCreate()

***Returns***:

OSAL\_Status - to indicate Success or Failure.

***Code Sample***:

if( OSAL\_FAIL == OSAL\_taskDelete(ve\_proc)) {

…

Code Example 3: Task Deletion

### Task Delay

Modules may call this routine in order to delay the calling task by a requested clock delay. The delay is implemented using the standard OS delay routines and will cause the calling task to block for the specified HZ (ticks) of the RTOS clock.

***Function Binding***:

OSAL\_Status OSAL\_taskDelay(uint32 msTimeout)

***Parameters***:

uint32 msTimeout - the maximum number of milliseconds to delay. To invoke the OS scheduler, set the parameter msTimeout to 0 (which is equivalent to OSAL\_NO\_WAIT).

***Returns***:

OSAL\_Status - to indicate Success or Failure.

***Code Sample***:

/\* to delay a maximum of 1/2 second \*/

OSAL\_taskDelay((uint32) 500 );

Code Example 4: Task Delay

## Inter-Process Communication

The following provides for a generic IPC model. There is one portable IPC method defined by this specification: message queues which provide blocking and non-blocking features.

The message queues specify that messages may be fixed or variable length, there is no QOS in the message and all queues are FIFO queues. This is a copy-in/copy-out queue mechanism. The queue model is a single producer/consumer. Multiple consumers yield unexpected results. For example, Figure 4 illustrates what might happen in a single producer, multiple consumer model:

Fig4

Figure 4: Single Producer, Multiple Consumer Queue Model

In the above model, task n puts several sequential messages on the queue. However, it does not know which of tasks a, b, or c will actually receive it.

This software interface also supports the notion of a Message Queue select()-like function whereby a single task can select on a set of queues. Figure 5, below, illustrates this. In this scenario, task n is receiving messages from multiple sources (in this example they are three tasks: a, b, and c). Task n is pended with a OSAL\_WAIT\_FOREVER until a message arrives on one of the queues.

This is accomplished using the OSAL\_msgQGrpCreate(), OSAL\_msgQGrpAdd() and OSAL\_msgQGrpRecv() routines below. There is no routine to send a single message to a set of queues (copying the message).

OSAL messages are implemented as fixed-length buffer-based messages. After a queue is created which can hold N messages of max message length M bytes, tasks can send messages on this queue whose length is 0 to M bytes long. Suppose a task sends a set of 3 messages of sizes 5, 10 and 15 bytes respectively and the max message size is set to 20. A separate task that requests a message of size 20 on 3 successive message queue receives will receive 5 bytes on the first, 10 on the second, and 15 on the third as expected.

Suppose that a task is receiving messages and the 5, 10 and 15 byte messages are queued in that order. If a task first requests to receive a message of length 3 bytes, then the first message would be received of size 3 bytes (even though the first message contained 5 bytes.) The remaining 2 bytes of the first message would be discarded. Suppose a second message queue receive were requested on the queue with a size of 2 bytes. Instead of receiving the remnant of the first message, the user would receive the first 2 bytes of the second message and the residual 8 bytes on the second message would be discarded. This illustrates why it is important to request the maxMessageLen bytes on all requests.

When operating in environments where there is a well-defined kernel-user domain, there should be some form of synchronization between the two domains for inter-process communication to occur between threads running in the kernel and those running in user-space. The above message queue model can suffice but with some extensions to help OSAL deal with the boundary issues. OSAL allows several forms of inter-process communication and synchronization. These are limited to OSAL normal and group message queues, semaphores and timers. Unix in general and Linux in particular does not allow the user to specifically control kernel resources unless the user is root.

Unix kernel device drivers do not push data out to user space but buffer data when it comes available. It’s up to user tasks to request data (typicially via a system call which results in a read on a device). The kernel can block the user until data is ready if the user requests a wait. But it is always at the request of user space code (with the exception of signals which may be delivered by the kernel at any time).

In Linux, it is not allowed for the user to perform a semaphore operation on a kernel semaphore, nor set a timer to be executed in the kernel context. Similarly, the kernel should not take user semaphore resources or request timers to be executed in user-space. For this reason, OSAL kernel-user IPC boundary code excludes the use of OSAL semaphores and timers as forms of inter-process synchronization.

Therefore, user-space code will focus on user-process/kernel-thread synchronization via the message queues offered by OSAL.

### Message Queue Creation

Modules may call this routine in order to create a message queue for IPC.

***Function Binding***:

OSAL\_MsgQId OSAL\_msgQCreate(char \*name\_ptr, vint srcModId, vint dstModId, vint msgStructId, uint32 maxMsgs, uint32 lenOfMsg, uint32 flags)

***Parameters***:

Char \*name\_ptr - C-string uniquely identifying this queue

vint srcModId - This identify source module id. This parameter is

defined in osal\_constant.h and refer to [section 4.5](#_Module_ID_Constants).

vint dstModId - This identify destination module id. This parameter is

defined in osal\_constant.h and refer to [section 4.5](#_Module_ID_Constants).

vint msgStructId - This identify message data structure id which is

transmitted in this message queue. This parameter is defined in osal\_constant.h and refer to [section 4.6](#_Message_ID_Constants).

uint32 maxMsgs – that the queue can hold until sender’s block on queue full

uint32 lenOfMsg – of each message in the queue

uint32 flags – a bit mask which is currently unused:

In Linux, there are two ports of OSAL: one for the user space (which is based on Pthreads) and one for the kernel-space (which is based on kernel-threads). When a task in user space wishes to communicate with a thread in kernel space via an OSAL message queue, both task and threads must perform an OSAL\_msgQCreate() specifying the same name in the name\_ptr argument.

***Returns***:

OSAL\_MsgQId msgQId - which is to be used for subsequent msgQ-related calls.

On success, this is non-zero. On failure, this is NULL.

***Code Sample For Creation of a Message Queue***:

OSAL\_MsgQId mq;

mq = OSAL\_msgQCreate(“queue”, OSAL\_MODULE\_NONE, OSAL\_MODULE\_NONE,

OSAL\_DATA\_STRUCT\_UNKONOWN, (uint32) MAX\_MOD\_MSGS,

(uint32) MAX\_MOD\_MSG\_LEN, (uint32) 0);

if(NULL == mq) {

…

Code Example 5: Message Queue Creation

### Message Queue Deletion

Modules may call this routine in order to delete a message queue. Message queues may not be deleted while bound to a message group.

***Function Binding***:

OSAL\_Status OSAL\_msgQDelete(OSAL\_MsgQId msgQId)

***Parameters***:

OSAL\_MsgQId msgQId - the id returned from OSAL\_msgQCreate()

***Returns***:

OSAL\_Status - to indicate Success or Failure

***Code Sample***:

if( OSAL\_FAIL == OSAL\_msgQDelete(mq)) {

…

Code Example 6: Message Queue Deletion

### Message Queue Send

Modules may call this routine in order to send a message to a message queue for IPC.

***Function Binding***:

OSAL\_Status OSAL\_msgQSend(OSAL\_MsgQId msgQId, void \*buf\_ptr,

uint32 nBytes, uint32 msTimeout, OSAL\_Boolean \*timeout\_ptr)

***Parameters***:

OSAL\_MsgQId msgQId - the id returned from OSAL\_msgQCreate()

void \*buf\_ptr - pointer to buffer to send

uint32 nBytes - number of bytes to send which must be <= maxMessageLen

that was specified on the OSAL\_msgQCreate() routine for

this message queue.

uint32 msTimeout - this is the maximum number of milliseconds that this task

will pend for if the queue is full on send. This value may be

OSAL\_NO\_WAIT, OSAL\_WAIT\_FOREVER, or a positive value.

OSAL\_Boolean \*timeout\_ptr -set to OSAL\_TRUE if the request timed out.

***Returns***:

OSAL\_Status - to indicate Success or Failure

***Code Sample***:

OSAL\_MsgQId mq;

…

/\* a blocking wait \*/

success = OSAL\_msgQSend(mq, myBuf, sizeof(myBuf), (uint32) OSAL\_WAIT\_FOREVER, sendStatus);

Code Example 7: Message Queue Send

### Message Queue Receive

Modules may call this routine in order to receive a message from a message queue.

***Function Binding***:

int32 OSAL\_msgQRecv(OSAL\_MsgQId msgQId, void \*buf\_ptr, uint32 maxBytes,

uint32 msTimeout, OSAL\_Boolean \*timeout\_ptr)

***Parameters***:

OSAL\_MsgQId msgQId - the id returned from OSAL\_msgQCreate()

void \*buf\_ptr - pointer to buffer to read a message into

uint32 maxBytes - number of bytes to receive which must be at least as big as

maxMessageLen that was specified on the

OSAL\_msgQCreate() routine for this message queue.

uint32 msTimeout - this is the maximum number of millisecs that this task will pend for if the queue is empty on receive. This value may be OSAL\_NO\_WAIT, OSAL\_WAITFORVER, or a positive value.

OSAL\_Boolean \*timeout\_ptr -set to OSAL\_TRUE if the request timed out.

***Returns***:

int32 numBytes - to indicate number of bytes actually received on this

message or –1 to indicate failure.

OSAL\_MsgQId mq;

/\* a blocking wait \*/

numBytesRecvd = OSAL\_msgQRecv(mq, myBuf, sizeof(myBuf),

(uint32) OSAL\_WAIT\_FOREVER, recvStatus);

if(sizeof(ExpectedSizeType) != numBytesRecvd) /\* error \*/ {

if((-1) == numBytesRecvd ) {

…

Code Example 8: Message Queue Receive

It is important to always specify a receive buffer big enough to hold a message size of length maxMessageLen that was specified on the OSAL\_msgQCreate() routine for this message queue.

If a task calls a OSAL\_MsgQRecv() specifying a maxBytes that is shorter than the actual message sent in the current message being received from the message queue, then the trailing part of that message being received may not be preserved or conveyed in the subsequent call. That is why it is important to always specify a receive buffer and maxBytes big enough to hold the biggest message.

### Message Queue Group Creation

In order to provide Modules with the ability to select on a specific set of Portable message queues, a group for the queues must be created. Modules may call this routine in order to create a group for a [set of] message queue(s). Groups are defined as priority-based or round robin based. For priority queues, message queues that are added to groups first have higher priority than those added later. High priority queues are always serviced first (which may lead to queue starvation in lower priority queues if not careful.) Round-robin based groups are those groups where all queues share equal weight and queues are serviced in round-robin order fairly.

***Function Binding***:

OSAL\_msgQGrpCreate(OSAL\_MsgQGrpId \*g\_ptr)

***Parameters***:

OSAL\_MsgQGrpId \*g\_ptr -pointer to a OSAL\_MsgQGrpId structure

***Returns***:

OSAL\_Status - to indicate Success or Failure

***Code Sample:***

See below in OSAL\_msgQGrpRecv()

### Message Queue Group Deletion

Modules may call this routine in order to delete a message grup queue.

***Function Binding***:

OSAL\_Status OSAL\_msgQGrpDelete(OSAL\_MsgQGrpId \*g\_ptr)

***Parameters***:

OSAL\_MsgQGrpId g\_ptr - the id returned from OSAL\_msgQGrpCreate()

***Returns***:

OSAL\_Status - to indicate Success or Failure

***Code Sample***:

if( OSAL\_FAIL == OSAL\_msgQGrpDelete(grpQ)) {

…

Code Example 9: Message Queue Group Deletion

### Message Queue Group Add Queue

Modules may call this routine in order to add a message queue to a group. More than one message queue can be added to a group.

***Function Binding***:

OSAL\_Status OSAL\_msgQGrpAddQ(OSAL\_MsgQGrpId \*msgQGId, OSAL\_MsgQId msgQId)

***Parameters***:

OSAL\_MsgQGrpId \*msgQGId - a pointer to the id populated by OSAL\_msgQGrpCreate()

OSAL\_MsgQId msgQId - the id returned from OSAL\_msgQCreate()

***Returns***:

OSAL\_Status - to indicate OSAL\_SUCCESS or OSAL\_FAIL.

***Code Sample:***

See below in OSAL\_msgQGrpRecv()

***Note:***

The order in which queues are added to a group determines the priority of the message queue. For example, if 3 message queues are added to a group, and 2 have them have messages delivered before a call to the group receive function: OSAL\_msgQGrpRecv(), then the one added first will have it’s message delivered to the caller.

### Message Queue Group Receive

Modules may call this routine in order to block on reception from a set of message queues identified by a group ID. The order of processing the queues is specified in the OSAL\_msgQGrpCreate() routine.

***Function Binding***:

int32 OSAL\_msgQGrpRecv(OSAL\_MsgQGrpId \*msgQGId, char \*buf\_ptr,

uint32 maxBytes, uint32 msTimeout, OSAL\_MsgQId \*msgQId\_ptr, OSAL\_Boolean \*timeout\_ptr)

***Parameters***:

OSAL\_MsgQGrpId msgQGId - a pointer to the id returned from OSAL\_msgQGrpCreate()

char \*buf\_ptr - pointer to buffer to read a message into. This should be big

enough to accommodate the maximum message size in all

buffers on the set of queues.

uint32 nBytes - number of bytes to receive into buf\_ptr

uint32 msTimeout - this is the maximum number of millisecs that this task will pend for if the set of queues are empty on receive. This value may be OSAL\_NO\_WAIT, OSAL\_WAITFORVER, or a positive value.

OSAL\_MsgQGrpId \*msgQId\_ptr - an address of where to put the msgQId (returned from OSAL\_msgQCreate()) which indicates from which queue this message was delivered from.

OSAL\_Boolean \*timeout\_ptr -set to OSAL\_TRUE if the request timed out.

***Returns***:

int32 numBytes - to indicate number of bytes actually received on this

message or –1 to indicate failure.

***Code Sample***:

/\* ERROR PROCESSING REMOVED FROM THIS SAMPLE \*/

/\* message queue descriptors \*/

OSAL\_MsgQGrpId mq1, mq2, mq3**;**

/\* message queue descriptors \*/

OSAL\_MsgQGrpId mqGid**;**

/\* create the message queues \*/

mq1 = OSAL\_msgQCreate((int32) MAX\_MOD\_MSGS, (int32) MAX\_MOD\_MSG\_LEN);

mq2 = OSAL\_msgQCreate((int32) MAX\_MOD\_MSGS, (int32) MAX\_MOD\_MSG\_LEN);

mq3 = OSAL\_msgQCreate((int32) MAX\_MOD\_MSGS, (int32) MAX\_MOD\_MSG\_LEN);

/\* establish the group \*/

mqGid = OSAL\_msgQGrpCreate();

/\* add the 3 msg queues to the group \*/

OSAL\_msgQGrpAddQ(&mqGid, mq1);

OSAL\_msgQGrpAddQ(&mqGid, mq2);

OSAL\_msgQGrpAddQ(&mqGid, mq2);

/\* a blocking wait on all three queues. \*/

numBytesRecvd = OSAL\_msgQGrpRecv(&mqGId, myBuf, sizeof(myBuf),

(uint32) OSAL\_WAIT\_FOREVER);

if(sizeof(ExpectedSizeType) != numBytesRecvd) /\* error \*/ {

…

Code Example 10: Message Queue Group Receive

## Network Access

Modules that are destined for use in non-kernel level code may use the following Socket calls.

/\*

\* DNS query types.

\*/

typedef enum {

OSAL\_NET\_RESOLVE\_A = 1,

OSAL\_NET\_RESOLVE\_SRV = 2,

OSAL\_NET\_RESOLVE\_AAAA = 3,

OSAL\_NET\_RESOLVE\_NAPTR = 4,

} OSAL\_NetResolveType;

/\*

\* Socket types.

\*/

typedef enum {

OSAL\_NET\_SOCK\_UDP = 0, /\* Streams \*/

OSAL\_NET\_SOCK\_TCP = 2, /\* Datagrams \*/

OSAL\_NET\_SOCK\_UDP\_V6 = 3,

OSAL\_NET\_SOCK\_TCP\_V6 =4,

} OSAL\_NetSockType;

/\*

\* Allowed SSL methods

\*/

typedef enum {

OSAL\_NET\_SSL\_METHOD\_CLIENT\_SSLV3 = 1,

OSAL\_NET\_SSL\_METHOD\_CLIENT\_TLSV1 = 2,

OSAL\_NET\_SSL\_METHOD\_CLIENT\_ALL = 3,

OSAL\_NET\_SSL\_METHOD\_SERVER\_SSLV3 = 4,

OSAL\_NET\_SSL\_METHOD\_SERVER\_TLSV1 = 5,

OSAL\_NET\_SSL\_METHOD\_SERVER\_ALL = 6,

} OSAL\_NetSslMethod;

typedef enum {

OSAL\_NET\_SOCK\_NONBLOCKING = 1, /\* Non blocking I/O \*/

OSAL\_NET\_SOCK\_REUSE = 2, /\* Reuse ports \*/

OSAL\_NET\_IP\_TOS = 3 /\* Enable TOS \*/

} OSAL\_NetSockopt;

typedef enum {

OSAL\_NET\_SSL\_HASH\_SHA1 = 1,

OSAL\_NET\_SSL\_HASH\_MD5 = 2

} OSAL\_NetSSlHash;

/\*

\* This structure serves as SSL ID.

\*/

typedef struct {

int \*ssl\_ptr;

int \*ctx\_ptr;

} OSAL\_NetSslId;

/\*

\* This serves as socket ID.

\*/

typedef int OSAL\_NetSockId;

/\*

\* Address structure

\*/

typedef struct {

uint32 ipv4;

uint16 ipv6[OSAL\_NET\_IPV6\_WORD\_SZ];

uint16 port;

OSAL\_NetSockType type;

} OSAL\_NetAddress;

Code Example 11: Network Socket Data Structures

### Socket Setup

The socket setup functions that may be used are as follows:

OSAL\_netAddressToString()

OSAL\_netStringToAddress()

**OSAL\_Status OSAL\_netSocket**(

OSAL\_NetSockId \*socket\_ptr,

OSAL\_NetSockType type);

**OSAL\_Status OSAL\_netBindSocket**(

OSAL\_NetSockId \*socket\_ptr,

OSAL\_NetAddress \*address\_ptr);

**OSAL\_Status OSAL\_netGetSocketAddress**(

OSAL\_NetSockId \*socket\_ptr,

OSAL\_NetAddress \*address\_ptr);

**OSAL\_Status OSAL\_netListenOnSocket**(

OSAL\_NetSockId \*socket\_ptr);

**OSAL\_Status OSAL\_netAcceptOnSocket**(

OSAL\_NetSockId \*socket\_ptr,

OSAL\_NetSockId \*newSocket\_ptr,

OSAL\_NetAddress \*address\_ptr);

OSAL\_Status OSAL\_netIsSocketIdValid(

OSAL\_NetSockId \*socket\_ptr);

### Socket Communication

The socket communication functions that may be used are as follows:

**OSAL\_Status OSAL\_netSocketSend**(

OSAL\_NetSockId \*socket\_ptr,

void \*buf\_ptr,

vint \*size\_ptr);

**OSAL\_Status OSAL\_netSocketSendTo**(

OSAL\_NetSockId \*socket\_ptr,

void \*buf\_ptr,

vint \*size\_ptr,

OSAL\_NetAddress \*address\_ptr);

**OSAL\_Status OSAL\_netSocketReceive**(

OSAL\_NetSockId \*socket\_ptr,

void \*buf\_ptr,

vint \*size\_ptr,

OSAL\_NetReceiveFlags flags);

**OSAL\_Status OSAL\_netSocketReceiveFrom**(

OSAL\_NetSockId \*socket\_ptr,

void \*buf\_ptr,

vint \*size\_ptr,

OSAL\_NetAddress \*address\_ptr);

### Socket Removal

The socket teardown functions that may be used are as follows:

**OSAL\_Status OSAL\_netCloseSocket**(

OSAL\_NetSockId \*socket\_ptr);

### Socket Control

The typical socket control functions that may be used are as follows:

**int getsockopt**(int socket, int level, int optName, void \*optValue,

unsigned int \*optLen);

**int setsockopt**(int socket, int level, int optName, const void \*optValue,

unsigned int optLen);

**OSAL\_Status OSAL\_netSetSocketOptions**(

OSAL\_NetSockId \*socket\_ptr,

OSAL\_NetSockopt option,

OSAL\_Boolean value);

### SSL Functions

**OSAL\_Status OSAL\_netSslConnect**(

OSAL\_NetSslId \*ssl\_ptr);

**OSAL\_Status OSAL\_netSsl**(

OSAL\_NetSslId \*ssl\_ptr,

OSAL\_NetSslMethod method);

**OSAL\_Status OSAL\_netSslClose**(

OSAL\_NetSslId \*ssl\_ptr);

**OSAL\_Status OSAL\_netIsSslIdValid**(

OSAL\_NetSslId \*ssl\_ptr);

**OSAL\_Status OSAL\_netSslSend**(

OSAL\_NetSslId \*ssl\_ptr,

void \*buf\_ptr,

vint \*size\_ptr);

**OSAL\_Status OSAL\_netSslReceive**(

OSAL\_NetSslId \*ssl\_ptr,

void \*buf\_ptr,

vint \*size\_ptr);

**OSAL\_Status OSAL\_netSslSetSocket**(

OSAL\_NetSockId \*socket\_ptr,

OSAL\_NetSslId \*ssl\_ptr);

OSAL\_Status OSAL\_netSslGetFingerprint(

OSAL\_NetSslCert \*cert\_ptr,

OSAL\_NetSslHash hash,

unsigned char \*fingerprint,

vint fpMaxLen,

vint \*fpLen\_ptr);

OSAL\_Status OSAL\_netSslSetCertVerifyCB(

OSAL\_NetSslId \*ssl\_ptr,

void \*certValidateCB);

OSAL\_Status OSAL\_netSslSetCert(

OSAL\_NetSslId \*ssl\_ptr,

OSAL\_NetSslCert \*cert\_ptr);

OSAL\_Status OSAL\_netSslCertDestroy(

OSAL\_NetSslCert \*cert\_ptr);

OSAL\_Status OSAL\_netSslCertGen(

OSAL\_NetSslCert \*cert\_ptr,

vint bits,

vint serial,

vint days);

### Network Address Helpers

The following are helpers routines for translating addresses.

OSAL\_Status OSAL\_netStringToAddress(

int8 \*ddn\_ptr,

OSAL\_NetAddress \*address\_ptr);

OSAL\_Status OSAL\_netAddressToString(

int8 \*ddn\_ptr,

OSAL\_NetAddress \*address\_ptr);

### Network Address Conversions

OSAL provides a number of routines to help convert IP address.

uint32 OSAL\_netNtohl(

uint32 val);

uint32 OSAL\_netHtonl(

uint32 val);

uint16 OSAL\_netNtohs(

uint16 val);

uint16 OSAL\_netHtons(

uint16 val);

void OSAL\_netIpv6Ntoh(

uint16 \*dst\_ptr,

uint16 \*src\_ptr);

void OSAL\_netIpv6Hton(

uint16 \*dst\_ptr,

uint16 \*src\_ptr);

void OSAL\_netAddrHton(

OSAL\_NetAddress \*dst\_ptr,

OSAL\_NetAddress \*src\_ptr);

void OSAL\_netAddrNtoh(

OSAL\_NetAddress \*dst\_ptr,

OSAL\_NetAddress \*src\_ptr);

void OSAL\_netAddrPortHton(

OSAL\_NetAddress \*dst\_ptr,

OSAL\_NetAddress \*src\_ptr);

void OSAL\_netAddrPortNtoh(

OSAL\_NetAddress \*dst\_ptr,

OSAL\_NetAddress \*src\_ptr);

OSAL\_Boolean OSAL\_netIpv6IsAddrZero(

uint16 \*addr\_ptr);

OSAL\_Boolean OSAL\_netIsAddrZero(

OSAL\_NetAddress \*addr\_ptr);

OSAL\_Boolean OSAL\_netIsAddrLoopback(

OSAL\_NetAddress \*addr\_ptr);

void OSAL\_netAddrCpy(

OSAL\_NetAddress \*dst\_ptr,

OSAL\_NetAddress \*src\_ptr);

void OSAL\_netIpv6AddrCpy(

uint16 \*dst\_ptr,

uint16 \*src\_ptr);

void OSAL\_netAddrPortCpy(

OSAL\_NetAddress \*dst\_ptr,

OSAL\_NetAddress \*src\_ptr);

OSAL\_Boolean OSAL\_netIsAddrEqual(

OSAL\_NetAddress \*addrA\_ptr,

OSAL\_NetAddress \*addrB\_ptr);

OSAL\_Boolean OSAL\_netIsAddrPortEqual(

OSAL\_NetAddress \*addrA\_ptr,

OSAL\_NetAddress \*addrB\_ptr);

OSAL\_Boolean OSAL\_netIsAddrIpv6(

OSAL\_NetAddress \*addr\_ptr);

void OSAL\_netAddrClear(

OSAL\_NetAddress \*addr\_ptr);

## Encryption

OSAL provides interfaces to perform common encryption encoding/decoding algorithms. These routines wrap around many popular ‘crypto’ functions in openssl or isc lib (DNS BIND).

#define OSAL\_CRYPTO\_SHA\_LBLOCK 16

#define OSAL\_CRYPTO\_SHA\_CBLOCK (SHA\_LBLOCK\*4)

#define OSAL\_CRYPTO\_SHA256\_CBLOCK (SHA\_LBLOCK\*4)

#define OSAL\_CRYPTO\_SHA512\_CBLOCK (SHA\_LBLOCK\*8)

#define OSAL\_CRYPTO\_SHA\_DIGEST\_LENGTH 20

#define OSAL\_CRYPTO\_SHA224\_DIGEST\_LENGTH 28

#define OSAL\_CRYPTO\_SHA256\_DIGEST\_LENGTH 32

#define OSAL\_CRYPTO\_SHA384\_DIGEST\_LENGTH 48

#define OSAL\_CRYPTO\_SHA512\_DIGEST\_LENGTH 64

/\* message digest algo to be used in HMAC \*/

typedef enum {

OSAL\_CRYPTO\_MD\_ALGO\_NONE = 0,

OSAL\_CRYPTO\_MD\_ALGO\_SHA256,

} OSAL\_CryptoMdAlgo;

typedef void \*OSAL\_CryptoCtxId;

OSAL\_CryptoCtxId OSAL\_cryptoAllocCtx(void);

OSAL\_Status OSAL\_cryptoFreeCtx(

OSAL\_CryptoCtxId ctxId);

/\* SHA256 algo \*/

OSAL\_Status OSAL\_cryptoSha256Init(

OSAL\_CryptoCtxId ctx);

OSAL\_Status OSAL\_cryptoSha256Update(

OSAL\_CryptoCtxId ctx,

const void \*data,

size\_t len);

OSAL\_Status OSAL\_cryptoSha256Final(

OSAL\_CryptoCtxId ctx,

unsigned char \*md);

unsigned char \*OSAL\_cryptoSha256(

const unsigned char \*d,

size\_t n,

unsigned char \*md);

/\* HMAC functions using specified md algo \*/

OSAL\_Status OSAL\_cryptoHmacInit(

OSAL\_CryptoCtxId ctx,

const void \*key,

size\_t keyLen,

OSAL\_CryptoMdAlgo mdAlgo);

OSAL\_Status OSAL\_cryptoHmacUpdate(

OSAL\_CryptoCtxId ctx,

const void \*data,

size\_t len);

OSAL\_Status OSAL\_cryptoHmacFinal(

OSAL\_CryptoCtxId ctx,

unsigned char \*md,

size\_t \*mdLen\_ptr);

unsigned char \*OSAL\_cryptoHmac(

const void \*key,

size\_t keyLen,

OSAL\_CryptoMdAlgo mdAlgo,

const unsigned char \*d,

size\_t n,

unsigned char \*md,

size\_t \*mdLen\_ptr);

/\* base64 encode/decode \*/

int OSAL\_cryptoB64Encode(

char \*s,

char \*t,

int size);

int OSAL\_cryptoB64Decode(

char \*s,

int size,

char \*t);

## IPSec

OSAL provides interfaces to manage secure IP tunnels. These routines wrap PF Key library implementations.

#define OSAL\_IPSEC\_KEY\_LENGTH\_HMAC\_MD5 16 /\* 128-bit \*/

#define OSAL\_IPSEC\_KEY\_LENGTH\_HMAC\_SHA1 20 /\* 160-bit \*/

#define OSAL\_IPSEC\_KEY\_LENGTH\_3DES\_CBC 24 /\* 192-bit \*/

#define OSAL\_IPSEC\_KEY\_LENGTH\_AES\_CBC 16 /\* 128-bit \*/

#define OSAL\_IPSEC\_KEY\_LENGTH\_MAX OSAL\_IPSEC\_KEY\_LENGTH\_3DES\_CBC

/\*

\* Maximum size of PFKEY message

\*/

#define OSAL\_IPSEC\_PFKEY\_MSG\_SIZE\_MAX 0xFFFF

/\* Maximum bytes for pfkey request \*/

#define OSAL\_IPSEC\_PFKEY\_REQUEST\_BYTE\_MAX 64

/\*

\* IPSec direction for security policy

\*/

typedef enum {

OSAL\_IPSEC\_DIR\_IN = 0,

OSAL\_IPSEC\_DIR\_OUT,

OSAL\_PSEC\_DIR\_LAST = OSAL\_IPSEC\_DIR\_OUT,

} OSAL\_IpsecDirection;

/\*

\* IPSec transport for security policy

\*/

typedef enum {

OSAL\_IPSEC\_TRANSPORT\_ANY = 0,

OSAL\_IPSEC\_TRANSPORT\_UDP,

OSAL\_IPSEC\_TRANSPORT\_TCP,

OSAL\_IPSEC\_TRANSPORT\_LAST = OSAL\_IPSEC\_TRANSPORT\_TCP,

} OSAL\_IpsecTransport;

/\*

\* IPSec mode for security policy

\*/

typedef enum {

OSAL\_IPSEC\_SP\_MODE\_TRANSPORT = 0,

OSAL\_IPSEC\_SP\_MODE\_TUNNEL,

OSAL\_IPSEC\_SP\_MODE\_LAST = OSAL\_IPSEC\_SP\_MODE\_TUNNEL,

} OSAL\_IpsecMode;

/\*

\* IPSec level for security policy

\*/

typedef enum {

OSAL\_IPSEC\_SP\_LEVEL\_DEFAULT = 0,

OSAL\_IPSEC\_SP\_LEVEL\_USE,

OSAL\_IPSEC\_SP\_LEVEL\_REQUIRE,

OSAL\_IPSEC\_SP\_LEVEL\_UNIQUE,

OSAL\_IPSEC\_SP\_LEVEL\_LAST = OSAL\_IPSEC\_SP\_LEVEL\_UNIQUE,

} OSAL\_IpsecLevel;

/\*

\* IPSec protocol for security association

\*/

typedef enum {

OSAL\_IPSEC\_PROTOCOL\_ESP = 0, /\* Default \*/

OSAL\_IPSEC\_PROTOCOL\_AH,

OSAL\_IPSEC\_PROTOCOL\_LAST = OSAL\_IPSEC\_PROTOCOL\_AH,

} OSAL\_IpsecProtocol;

/\*

\* IPSec autheticaiton algorithm for security association

\*/

typedef enum {

OSAL\_IPSEC\_AUTH\_ALG\_HMAC\_MD5\_96 = 0,

OSAL\_IPSEC\_AUTH\_ALG\_HMAC\_SHA1\_96,

OSAL\_IPSEC\_AUTH\_ALG\_NULL,

OSAL\_IPSEC\_AUTH\_ALG\_LAST = OSAL\_IPSEC\_AUTH\_ALG\_NULL,

} OSAL\_IpsecAuthAlg;

/\*

\* IPSec encrypt algorithm for security association

\*/

typedef enum {

OSAL\_IPSEC\_ENC\_ALG\_NULL = 0, /\* Default \*/

OSAL\_IPSEC\_ENC\_ALG\_DES\_EDE3\_CBC,

OSAL\_IPSEC\_ENC\_ALG\_AES\_CBC, /\* Requires by TS 33.203 \*/

OSAL\_IPSEC\_ENC\_ALG\_LAST = OSAL\_IPSEC\_ENC\_ALG\_AES\_CBC,

} OSAL\_IpsecEncryptAlg;

/\*

\* IPSEC security policy

\*/

typedef struct {

OSAL\_NetAddress srcAddr; /\* source address \*/

OSAL\_NetAddress dstAddr; /\* destination address \*/

OSAL\_IpsecProtocol protocol; /\* ESP, AH or both \*/

OSAL\_IpsecDirection dir; /\* in or out \*/

OSAL\_IpsecTransport transport; /\* UDP, TCP or any \*/

OSAL\_IpsecMode mode; /\* transport, tunnel or any \*/

OSAL\_IpsecLevel level; /\* default, use, require or unique \*/

uint32 reqId; /\* Request Id \*/

} OSAL\_IpsecSp;

/\*

\* IPSEC security association

\*/

typedef struct {

OSAL\_NetAddress srcAddr; /\* source address \*/

OSAL\_NetAddress dstAddr; /\* destination address \*/

OSAL\_IpsecProtocol protocol; /\* ESP or AH \*/

OSAL\_IpsecMode mode; /\* transport, tunnel or any \*/

uint32 spi; /\* Security Parameter Index \*/

uint32 reqId; /\* Request Id \*/

OSAL\_IpsecAuthAlg algAh;

OSAL\_IpsecEncryptAlg algEsp;

char keyAh[OSAL\_IPSEC\_KEY\_LENGTH\_MAX + 1];/\* Key for AH \*/

char keyEsp[OSAL\_IPSEC\_KEY\_LENGTH\_MAX + 1];/\* Key for ESP \*/

} OSAL\_IpsecSa;

OSAL\_Status OSAL\_ipsecCreateSA(

OSAL\_IpsecSa \*sa\_ptr,

OSAL\_IpsecSp \*sp\_ptr);

OSAL\_Status OSAL\_ipsecDeleteSA(

OSAL\_IpsecSa \*sa\_ptr,

OSAL\_IpsecSp \*sp\_ptr);

## File I/O

Lightweight file I/O is supported for environments with a file system. Below are basic file I/O facilities provided by OSAL.

typedef enum {

OSAL\_FILE\_SEEK\_SET = 0, /\* The offset is set to offset bytes. \*/

OSAL\_FILE\_SEEK\_CUR = 1, /\* The offset is set to its current location plus

\* offset bytes. \*/

OSAL\_FILE\_SEEK\_END = 2, /\* The offset is set to the size of the file plus

\* offset bytes. \*/

} OSAL\_FileSeekType;

typedef enum {

OSAL\_FILE\_O\_CREATE = 0,/\* If the file does not exist it will be created. \*/

OSAL\_FILE\_O\_APPEND = 1,/\* The file is opened in append mode. Before each

\* write the file offset is positioned at the end

\* of the file \*/

OSAL\_FILE\_O\_RDONLY = 2,/\* Open for reading only. \*/

OSAL\_FILE\_O\_WRONLY = 4,/\* Open for writing only. \*/

OSAL\_FILE\_O\_RDWR = 8, /\* Open for reading and writing. The result is

\* undefined if this flag is applied to a FIFO. \*/

} OSAL\_FileFlags;

OSAL\_Status OSAL\_fileOpen(

OSAL\_FileId \*fileId\_ptr,

const uint8 \*pathname\_ptr,

OSAL\_FileFlags flags,

uint32 mode);

OSAL\_Status OSAL\_fileClose(OSAL\_FileId \*fileId\_ptr);

OSAL\_Status OSAL\_fileRead(

OSAL\_FileId \*fileId\_ptr,

void \*buf\_ptr,

vint \*size\_ptr);

OSAL\_Status OSAL\_fileWrite(

OSAL\_FileId \*fileId\_ptr,

void \*buf\_ptr,

vint \*size\_ptr);

OSAL\_Status OSAL\_fileSeek(

OSAL\_FileId \*fileId\_ptr,

vint \*offset\_ptr,

OSAL\_FileSeekType type);

OSAL\_Status OSAL\_fileGetSize(OSAL\_FileId \*fileId\_ptr, vint \*size\_ptr);

OSAL\_Boolean OSAL\_fileExists(const char \*pathname\_ptr);

## Semaphores

This set of routines provide a uniform mechanism for specifying inter-task synchronization primitives using semaphores: mutex and counting.

### Semaphore Creation

Modules may call this routine in order to create a portable counting semaphore.

***Function Binding***:

OSAL\_SemIdOSAL\_semCountCreate(int32 initialCount)

***Parameters***:

int32 initialCount – must be greater than 0

***Returns***:

OSAL\_SemId semId - which is to be used for subsequent OSAL semaphore-related calls. On success, this is non-zero, on failure, this is NULL

***Code Sample***:

OSAL\_SemId cSemId;

cSemId = OSAL\_semCountCreate((int32)3);

if(NULL == cSemId) {

…

Code Example 14: Counting Semaphore Creation

Modules may call this routine in order to create a portable mutex semaphore.

***Function Binding***:

OSAL\_SemIdOSAL\_semMutexCreate()

***Parameters***:

void

***Returns***:

OSAL\_SemId semId - which is to be used for subsequent OSAL semaphore-related calls. On success, this is non-zero, on failure, this is NULL

***Code Sample***:

OSAL\_SemId mSemId;

mSemId = OSAL\_semMutexCreate();

if(NULL == mSemId) {

…

Code Example 15: Mutex Semaphore Creation

Modules may call this routine in order to create a portable binary semaphore.

***Function Binding***:

OSAL\_SemIdOSAL\_semBinaryCreate(OSAL\_SemBState initState)

***Parameters***:

OSAL\_SemBState initState – 0 to make binary semaphore initially unavailable, 1 to make binary semaphore initially available

***Returns***:

OSAL\_SemId semId - which is to be used for subsequent OSAL semaphore-related calls. On success, this is non-zero, on failure, this is NULL

***Code Sample***:

OSAL\_SemId mSemId;

mSemId = OSAL\_semBinaryCreate(OSAL\_SEM\_EMPTY);

if(NULL == mSemId) {

…

Code Example 16: Binary Semaphore Creation

### Semaphore Deletion

Modules may call this routine in order to delete a portable binary, counting, or mutex semaphore.

***Function Binding***:

OSAL\_StatusOSAL\_semDelete(OSAL\_SemId sId)

***Parameters***:

OSAL\_SemId sId – returned from OSAL\_semBinaryCreate(), OSAL\_semCountCreate(), or OSAL\_semMutexCreate().

***Returns***:

OSAL\_Status - success or failure

***Code Sample***:

OSAL\_SemId cSemId;

cSemId = OSAL\_semCountCreate(3);

…

if(OSAL\_semDelete(cSemId) != OSAL\_SUCCESS)

{

….

Code Example 16: Semaphore Deletion

### Semaphore Acquisition

Modules may call this routine in order to acquire a portable binary, counting, or mutex semaphore.

***Function Binding***:

OSAL\_StatusOSAL\_semAcquire(OSAL\_SemId sId, uint32 msTimeout)

***Parameters***:

OSAL\_SemId sId – returned from OSAL\_semBinaryCreate(), OSAL\_semCountCreate(), or OSAL\_semMutexCreate().

uint32 timeout OSAL\_NO\_WAIT, OSAL\_WAIT\_FOREVER, or a positive number representing the maximum millisecond timeout the caller is willing to wait. See Section 3 above for a discussion of time.

***Returns***:

OSAL\_Status - success or failure

***Code Sample***:

OSAL\_SemId cSemId;

cSemId = OSAL\_semCountCreate(3);

…

if(OSAL\_SUCCESS != OSAL\_semAcquire(cSemId, OSAL\_NO\_WAIT)) {

/\* semaphore not acquired \*/

Code Example 17: Semaphore Acquisition

### Semaphore Giving

Modules may call this routine in order to give a portable binary, counting, or mutex semaphore.

***Function Binding***:

OSAL\_StatusOSAL\_semGive(OSAL\_SemId sId)

***Parameters***:

OSAL\_SemId sId – returned from OSAL\_semBinaryCreate(), OSAL\_semCountCreate(), or OSAL\_semMutexCreate().

***Returns***:

OSAL\_Status - success or failure

***Code Sample***:

OSAL\_SemId cSemId;

cSemId = OSAL\_semCountCreate(3);

…

if(OSAL\_semGive(cSemId) != OSAL\_SUCCESS) {

/\* semaphore not given \*/

Code Example 18: Semaphore Release

### Conditional Application Exiting

Modules may call this routine in order to setup conditional flow of a process based on the termination or interruption of the calling processes. Processes that will be controlled conditionally must use OSAL\_condApplicationExitWaitForCondition(),and will be blocked from continuation until the process that previously called OSAL\_condApplicationExitRegister() receives a termination or interrupt signal.

***Function Binding***:

OSAL\_StatusOSAL\_condApplicationExitRegister()

***Parameters***:

void

***Returns***:

OSAL\_Status - success or failure

***Code Sample***:

if(OSAL\_condApplicationExitRegister() != OSAL\_SUCCESS) {

/\* conditional exiting not successfully registered \*/

Code Example 19: Registering Conditional Application Exiting

Modules may call this routine in order to halt the flow of the current process until another process has exited or been terminated. Processes that will be controlled conditionally must use OSAL\_condApplicationExitWaitForCondition(),and will be blocked from continuation until the process that previously called OSAL\_condApplicationExitRegister() receives a termination or interrupt signal.

***Function Binding***:

OSAL\_StatusOSAL\_condApplicationExitWaitForCondition()

***Parameters***:

void

***Returns***:

OSAL\_Status - success or failure

***Code Sample***:

/\* Block until previously registered process exits \*/

if(OSAL\_condApplicationExitWaitForCondition() != OSAL\_SUCCESS) {

/\* could not wait on process \*/

Code Example 20: Conditional Application Exiting

Modules may call this routine in order to disable conditional flow of a process based on the termination or interruption of the calling processes. Processes that will be controlled conditionally must use OSAL\_condApplicationExitWaitForCondition(),and will be blocked from continuation until the process that previously called OSAL\_condApplicationExitRegister() receives a termination or interrupt signal. To stop the controller process from raising a condition upon exit or termination it must call OSAL\_condApplicationExitUnregister().

***Function Binding***:

OSAL\_Status OSAL\_condApplicationExitUnregister()

***Parameters***:

void

***Returns***:

OSAL\_Status - success or failure

***Code Sample***:

/\* Block until previously registered process exits \*/

if(OSAL\_condApplicationExitUnregister () != OSAL\_SUCCESS) {

/\* could not unregister conditional application exiting \*/

Code Example 21: Unregistering Conditional Application Exiting

## Timer Management

This timer specification is not POSIX timer compliant.

The program flow is typically as follows: a user task creates a timer to obtain a OSAL\_TmrId. Using this id, code can then schedule a one-shot timer, a periodic timer, or may cancel an outstanding timer that has been already started. A handler routine is called at timer expiration. In some RTOS, the handler will be executed at interrupt context while in other RTOS, the handler will be scheduled in either kernel-context or a timer task context. Therefore, modules must limit the scope of the handler to functions available in interrupt context. This means they cannot take semaphores but rather only give them. They may also perform interprocess communication if the timeouts are set to OSAL\_NO\_WAIT.

All timers that are started are executed by the RTOS when the closest approximation to RTOS ticks is reached. Refer to the documentation on arbitrary timeouts for more information.

### Timer Creation

Modules may call this routine in order to create a timer which, if it expires, will cause a C routine to be executed.

***Function Binding***:

OSAL\_TmrIdOSAL\_tmrCreate(void)

***Parameters***:

void

***Returns***:

OSAL\_TmrId - an id to a timer for deleting, starting or canceling

***Code Sample***:

OSAL\_TmrId tmrId;

tmrId = OSAL\_tmrCreate();

if(NULL == tmrId) {

…

Code Example 22: Timer Creation

Modules may call this routine in order to delete a timer previously created using OSAL\_tmrCreate().

### Timer Deletion

***Function Binding***:

OSAL\_StatusOSAL\_tmrDelete(OSAL\_TmrId tmrId)

***Parameters***:

tmrId - a timer previously returned from OSAL\_tmrCreate()

***Returns***:

OSAL\_Status - to indicate success or failure

***Code Sample***:

OSAL\_TmrId tmrId;

…

If(OSAL\_SUCCESS != (OSAL\_tmrDelete(tmrId)) {

/\* error processing \*/

Code Example 23: Timer Deletion

**Notes**: This may not be called within an IRQ Service routine.

### Timer Status – One-Shot and Periodic Timer Stop

Modules may call this routine in order to see if a timer is currently running stop.

***Function Binding***:

OSAL\_StatusOSAL\_tmrIsRunning(OSAL\_TmrId tmrId)

***Parameters***:

tmrId - a timer previously returned from OSAL\_tmrCreate()

***Returns***:

OSAL\_Status - to indicate whether or not a timer is running

### Timer Control – One-Shot Timer Start

Modules may call this routine in order to start a timer previously created using OSAL\_tmrCreate(). It binds timer expiration to a C-routine. If the timer has been previously been started, the user code must first call OSAL\_tmrCancel() before calling this routine.

***Function Binding***:

OSAL\_StatusOSAL\_tmrStart(OSAL\_TmrId tmrId, OSAL\_FuncPtr myRoutine,

uint32 arg, uint32 msTimeout)

***Parameters***:

tmrId - a timer previously returned from OSAL\_tmrCreate()

myRoutine - A C routine.

arg - a User’s argument to be supplied to the routine on expiration.

msTimeout - OSAL\_NO\_WAIT, OSAL\_WAIT\_FOREVER, or a positive number representing the maximum millisecond timeout the caller is willing to wait. See Section 3 above for a discussion of time.

***Returns***:

OSAL\_Status - to indicate success or failure

***Code Sample***:

OSAL\_TmrId tmrId;

tmrId = OSAL\_tmrCreate();

…

/\* start a 250 ms timer \*/

if( OSAL\_SUCCESS != OSAL\_tmrStart(tmrId, myFx, (uint32) myArg, 250))

{

…

Code Example 24: Starting a Timer

**Notes**: This may not be called within an IRQ Service routine.

### Timer Control – Periodic Timer Start

Modules may call this routine in order to start a periodic timer previously created using OSAL\_tmrCreate(). It binds timer expiration to a C-routine. If the timer has been previously been started, the user code must first call OSAL\_tmrCancel() before calling this routine.

***Function Binding***:

OSAL\_Status OSAL\_tmrPeriodicStart(OSAL\_TmrId tmrId,

OSAL\_FuncPtr myRoutine, uint32 arg, uint32 msTimeout)

***Parameters***:

tmrId - a timer previously returned from OSAL\_tmrCreate()

myRoutine - A C routine.

arg - a User’s argument to be supplied to the routine on timer trigger.

msTimeout - The max number of milliseconds between periodic invocations of

of the C-routine pointed to by myRoutine.

***Returns***:

OSAL\_Status - to indicate success or failure

***Code Sample***:

OSAL\_TmrId tmrId;

tmrId = OSAL\_tmrCreate();

…

/\* start a 250 ms timer \*/

if( OSAL\_SUCCESS != OSAL\_tmrPeriodicStart(tmrId, myFx,

(uint32) myArg, 250)) {

…

Code Example 25: Starting a Periodic Timer

**Notes**: This may not be called within an IRQ Service routine.

### Timer Control – One-Shot and Periodic Timer Stop

Modules may call this routine in order to stop (cancel) a timer previously started using OSAL\_tmrStart() or OSAL\_tmrPeriodicStart(). Note that if a clock tick occurs during this call, the routine may actually be scheduled one final time. This routine may be called from ISR context.

***Function Binding***:

OSAL\_StatusOSAL\_tmrStop(OSAL\_TmrId tmrId)

***Parameters***:

tmrId - a timer previously returned from OSAL\_tmrCreate()

***Returns***:

OSAL\_Status - to indicate success or failure

***Code Sample***:

OSAL\_TmrId tmrId;

tmrId = OSAL\_tmrCreate();

…

if( OSAL\_SUCCESS != OSAL\_tmrStart(tmrId, myFx, (uint32) myArg, 250))

…

if( OSAL\_SUCCESS != OSAL\_tmrStop(tmrId))

…

Code Example 26: Stopping (Canceling) a Timer

### Timer Control – One-Shot and Periodic Timer Interupt

Module can wake up a timer if it's been sleeping and either fires, if the time has elasped, or reset and sleeps until the specified expiration. This routine is typically called when a mobile device is placed into suspend mode and then awakens from suspend mode. Calling this routine is used in the case to wake up and reset timers that were totally idle during suspend mode.

***Function Binding***:

OSAL\_StatusOSAL\_tmrInterrupt(OSAL\_TmrId tmrId)

***Parameters***:

tmrId - a timer previously returned from OSAL\_tmrCreate()

***Returns***:

OSAL\_Status - to indicate success or failure.

## Time Retrieval

Modules can ask OSAL for the current time. Two facilities are defined below to get the time of day.

### Time Since Epoch Fine

This function gets the number of seconds and microseconds since the Epoch. The timeVal\_ptr argument is a pointer to a struct OSAL\_TimeVal (as specified in osal\_time.h

***Function Binding***:

OSAL\_StatusOSAL\_timeGetTimeOfDay(OSAL\_TimeVal \*timeVal\_ptr)

***Parameters***:

timeVal\_ptr - a pointer to a OSAL\_TimeVal object

***Returns***:

OSAL\_Status - to indicate success or failure

### Time Since Epoch Formatted as ISO 8601

This function gets the time of day in the time format defined in ISO 8601 and then writes a NULL terminated string to the buffer indicated by the buff\_ptr parameter. The size\_ptr parameter indicates the max size of the buffer pointed to by buff\_ptr. Upon successful completion the size of the formatted ISO 8601 string will be written to the size\_ptr parameter.

***Function Binding***:

OSAL\_StatusOSAL\_timeGetISO8601(uint8 \*buff\_ptr, vint \*size\_ptr)

***Parameters***:

buff\_ptr - a pointer to a buffer where the time will be written

size\_ptr - a pointer to a value that holds the size of of the buff\_ptr

***Returns***:

OSAL\_Status - to indicate success or failure

### Time Local

This function gets the number of seconds, minutes, hours, day, month, year, day of week, day in year, & daylight savings time since the Epoch. The timeLocal\_ptr argument is a pointer to a struct OSAL\_TimeLocal (as specified in osal\_time.h

***Function Binding***:

OSAL\_StatusOSAL\_timeLocalTime(OSAL\_TimeLocal \*timeLocal\_ptr)

***Parameters***:

timeLocal\_ptr - a pointer to a OSAL\_TimeLocal object

***Returns***:

OSAL\_Status - to indicate success or failure

## Memory Management

Note that these routines will use the memory services of the RTOS. It is not a memory allocator replacement. The intent is to facilitate memory corruption and leak-detection within the memory system.

For future use, memory functions use a uint32 memArg parameter that is defined in osal\_mem.h.

#define OSAL\_MEM\_ARG\_STATIC\_ALLOC (0) // For static alloc

#define OSAL\_MEM\_ARG\_DYNAMIC\_ALLOC (1) // For dynamic alloc

### Memory Allocation

Modules may call this routine in order to create a non-shared memory buffer.

It is strongly recommended to use OSAL\_memCalloc( ) instead of the generic OSAL\_memAlloc( ). Intializing the memory to zero may reducing programming errors and imposes little penalty if used when the module initializes.

***Function Binding***:

void \*OSAL\_memAlloc(int32 nBytes, int32 memArg)

***Parameters***:

int32 nBytes – to allocate

int32 memArg – Arg for static or dynamic allocation. OSAL\_MEM\_ARG\_STATIC\_ALLOC for static allocate, OSAL\_MEM\_ARG\_DYNAMIC\_ALLOC for dynamic allocate

***Returns***:

void \*buffer\_ptr - pointer to the block or NULL on failure

***Error Codes****:*

OSAL\_E\_NO\_MEM Not enough memory to complete the request

OSAL\_E\_INVAL nBytes must be >= 0

OSAL\_E\_NOT\_ISR\_CALLABLE Cannot call this operation from IRQ context

***Code Sample***:

struct MyStruct \*buf;

buf = (struct MyStruct \*) OSAL\_memAlloc((int32) sizeof(struct MyStruct), OSAL\_MEM\_ARG\_STATIC\_ALLOC);

if(NULL == buf) {

switch (OSAL\_errorcodeGet()) {

case OSAL\_E\_NO\_MEM: …

Code Example 28: Allocating Local Memory

***Notes***: This may not be called within an IRQ Service routine.

The actual number of bytes allocated may be greater than numElements\*elementSize and alignment is guaranteed.

Modules may call this routine in order to create space for a non-shared memory array.

On return, the buffer (if non-NULL) is set to all 0’s.

***Function Binding***:

void \*OSAL\_memCalloc(int32 numElements, int32 elementSize, int32 memArg)

***Parameters***:

int32 numElements – to allocate

int32 elementSize – size of each array element.

int32 memArg – unused presently, set this to 0.

***Returns***:

void \*buffer - pointer to the block or NULL on failure

***Error Codes****:*

OSAL\_E\_NO\_MEM Not enough memory to complete the request

OSAL\_E\_INVAL parameters must be >= 0

OSAL\_E\_NOT\_ISR\_CALLABLE Cannot call this operation from IRQ context

***Code Sample*:**

struct MyStruct \*\*buf;

buf = (struct MyStruct \*\*) OSAL\_memCalloc((int32) 20, (int32)

sizeof(struct MyStruct), 0);

if(NULL == buf) {

switch (OSAL\_errorcodeGet()) {

case OSAL\_E\_NO\_MEM: …

Code Example 29: Allocating Array (Cleared) Memory

**Notes**: This may not be called within an IRQ Service routine.

The actual number of bytes allocated may be greater than numElements\*elementSize and alignment is guaranteed.

### Memory De-allocation

Modules may call this routine in order to free a non-shared memory array which was previously allocated using OSAL\_memAlloc() or OSAL\_memCalloc(). It has some protection in it to detect double frees, buffer over-runs/under-runs and memory utilization on a per-task basis.

***Function Binding***:

OSAL\_StatusOSAL\_memFree(void \*vAddr\_ptr, int32 memArg)

***Parameters*:**

void \*vAddr\_ptr – buffer previously allocated using OSAL\_memAlloc() or OSAL\_memCalloc().

int32 memArg – Arg for static or dynamic allocation. OSAL\_MEM\_ARG\_STATIC\_ALLOC for static allocate, OSAL\_MEM\_ARG\_DYNAMIC\_ALLOC for dynamic allocate

***Returns***:

OSAL\_Status - to indicate Success or Failure. On OSAL\_FAIL, the error code returned by OSAL\_errorcodeGet() will be set to an appropriate value as follows:

***Error Codes:***

OSAL\_E\_NO\_MEM Not enough memory to complete the request

OSAL\_E\_INVAL Bad vAddr buffer pointer

OSAL\_E\_NOT\_ISR\_CALLABLE Cannot call this operation from IRQ context

***Code Sample*:**

struct MyStruct \*buf;

buf = (struct MyStruct \*) OSAL\_memAlloc((int32) sizeof(struct MyStruct), OSAL\_MEM\_ARG\_STATIC\_ALLOC);

…

OSAL\_memFree((void \*) buf);

Code Example 30: Memory De-allocation

Notes: This may not be called within an IRQ Service routine.

### Memory Copying, Comparing, Initializing, Move, Reallocate

The following routines are provided to perform copying, Comparing, initializing, move, and reallocate of memory.

void OSAL\_memSet(

void \*mem\_ptr,

int32 value,

int32 size);

void OSAL\_memCpy(

void \*dest\_ptr,

const void \*src\_ptr,

int32 size);

int OSAL\_memCmp(

const void \*mem1\_ptr,

const void \*mem2\_ptr,

int32 size);

void OSAL\_memMove(

void \*dest\_ptr,

const void \*src\_ptr,

int32 size)

void \* OSAL\_memReAlloc(

void \*mem\_ptr,

int32 numBytes,

int32 memArg)

## Memory Monitor

vPort 1.3 removes memory monitor operations from OSAL.

## Portable Message Logging

This routine provides a logging capability to the standard output in the system.

***Function Binding***:

OSAL\_Status OSAL\_logMsg(const char \*msg\_ptr, …)

OSAL uses a variable number of arguments in vPort. Modules must be careful to match the number of format string %-arguments to the number of arguments passed; otherwise, stack overflow may occur.

***Parameters*:**

char \*msg\_ptr – pointer to a printf formatted string.

***Returns***:

OSAL\_Status - Always returns

***Code Sample*:**

OSAL\_logMsg(“Invalid State Condition %d, event %s.\n”, state,

“Successful RCS Feature”);

Code Example 31: Message Logging

# Interrupt Context Requirements

No OSAL functions may be called from an IRQ context.

# Running The OSAL Unit Tests

## Test Setup

The OSAL unit tests are broken into two major categories: OSAL user space unit tests and OSAL kernel space user tests.

**User Space**

The user space tests are compiled into an executable named “osal\_ut\_user” that must be transferred to the board and run from there.

**Kernel Space**

The kernel space tests are compiled into a kernel module named “osal\_ut\_kernel.ko” wish must be transferred to the board and inserted.

## Test Commands

Both user and kernel space unit tests execute specific tests based on a single hexadecimal bitmask argument.

**User Space**

User space unit tests are run by executing the command (in the directory where ‘osal\_ut\_user’ is located) “osal\_ut\_user [mask]” where ‘mask’ is a decimal integer that represents a bitwise or of the following:

0x01 – run memory tests

0x02 – run task tests

0x04 – run message tests

0x10 – run timer tests

0x20 – run semaphore tests

0x40 – run kernel-user IPC tests

0x80 – run task stress tests

Note that the mask must be entered in decimal, such that if you were to execute the memory and timer tests (which would equate to ‘0x01 or 0x10’ = ‘0x11’) you would need to type:

“osal\_ut\_user 17”

due to the fact that 0x11 in hexadecimal equates to 17 in decimal.

One can also simply type “osal\_ut\_user” with no argument and a usage message will be printed to the screen.

The unit tests will run and print various useful messages, along with a total number of errors that occurred during the length of the test.

**Kernel Space**

Kernel space unit tests are run by writing to a special proc file named “/proc/d2/osalut”

This is accomplished by typing “echo [mask] > /proc/d2/osalut” where ‘mask’ a decimal integer that represents a bitwise or of the following:

0x000001 – run memory tests

0x000002 – run task tests

0x000004 – run message queue tests

0x000080 – run semaphore tests

0x000100 – run real-time verification tests

0x001000 – run net loopback on a single IP port tests

0x002000 – run net loopbacks on multiple IP ports tests

0x008000 – run net reusing bindings tests

0x010000 – run net polling for packets tests

0x020000 – run net socket open/close stress tests

Note that the mask must be entered in decimal, such that if you were to execute the memory and net loopback tests (which would equate to ‘0x01 or 0x1000’ = ‘0x1001’) you would need to type:

“echo 4097 > /proc/d2/osalut”

due to the fact that 0x1001 in hexadecimal equates to 4097 in decimal.

One can also simply type “cat /proc/d2/osalut” and a usage message will be printed to the screen.

The unit tests will run and print various useful messages, along with a total number of errors that occurred during the length of the test.

## Step by Step Instructions for Running the Tests

The following instructions list how to run all the tests, with an example shown.

1. Create OSAL tmp directory:

mkdir -p /var/tmp/osal

1. Go to directory containing OSAL modules:

cd /pub/cdecuir/m6406

1. Insert OSAL modules:

insmod extern\_osal\_kernel\_gpl.ko

insmod osal\_kernel.ko

insmod osal\_ut\_kernel.ko

1. Run all the osal kernel unit tests as follows:

echo 242055 > /proc/d2/osalut

1. Kernel unit tests take about 44 seconds to complete
2. Check the final console output for test results:

10/10 tests passed

1. If some tests fail, examine remainder of the console output to find point of failure
2. Run all the OSAL user unit tests as follows:

./osal\_ut\_user 1015

1. User unit tests take about 76 seconds to complete
2. Check the final console output for test results:

7/7 tests passed

1. If some tests fail, examine remainder of the console output to find point of failure

Note that the OSAL tmp directory must be created, and the osal\_ut\_kernel.ko module must be inserted before running either kernel or user unit tests.