OSAL User's Guide

Generated by Doxygen 1.8.13

Contents

1 Osal API Documentation

2 OSAL Introduction

The goal of this library 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.

The OS Application Program Interfaces (APIs) are broken up into core, file system, loader, network, and timer APIs. See the related document sections for full descriptions.

Note

The majority of these APIs should be called from a task running in the context of an OSAL application and in general should not be called from an ISR. There are a few exceptions, such as the ability to give a binary semaphore from an ISR.

3 File System Overview

The File System API is a thin wrapper around a selection of POSIX file APIs. In addition the File System API presents a common directory structure and volume view regardless of the underlying system type. For example, vxWorks uses MS-DOS style volume names and directories where a vxWorks RAM disk might have the volume “RAM:0”. With this File System API, volumes are represented as Unix-style paths where each volume is mounted on the root file system:

- RAM:0/file1.dat becomes /mnt/ram/file1.dat
- FL:0/file2.dat becomes /mnt/fl/file2.dat

This abstraction allows the applications to use the same paths regardless of the implementation and it also allows file systems to be simulated on a desktop system for testing. On a desktop Linux system, the file system abstraction can be set up to map virtual devices to a regular directory. This is accomplished through the OS_mkfs call, OS_mount call, and a BSP specific volume table that maps the virtual devices to real devices or underlying file systems.

In order to make this file system volume abstraction work, a “Volume Table” needs to be provided in the Board Support Package of the application. The table has the following fields:

- Device Name: This is the name of the virtual device that the Application uses. Common names are “ramdisk1”, “flash1”, or “volatile1” etc. But the name can be any unique string.
- Physical Device Name: This is an implementation specific field. For vxWorks it is not needed and can be left blank. For a File system based implementation, it is the “mount point” on the root file system where all of the volume will be mounted. A common place for this on Linux could be a user’s home directory, “/tmp”, or even the current working directory “.”. In the example of “/tmp” all of the directories created for the volumes would be under “/tmp” on the Linux file system. For a real disk device in Linux, such as a RAM disk, this field is the device name “/dev/ram0”.
- Volume Type: This field defines the type of volume. The types are: FS_BASED which uses the existing file system, RAM_DISK which uses a RAM_DISK device in vxWorks, RTEMS, or Linux, FLASH_DISK_FORMAT which uses a flash disk that is to be formatted before use, FLASH_DISK_INIT which uses a flash disk with an existing format that is just to be initialized before it’s use, EEPROM which is for an EEPROM or PROM based system.
- Volatile Flag: This flag indicates that the volume or disk is a volatile disk (RAM disk) or a non-volatile disk, that retains its contents when the system is rebooted. This should be set to TRUE or FALSE.
- Free Flag: This is an internal flag that should be set to FALSE or zero.
- Is Mounted Flag: This is an internal flag that should be set to FALSE or zero. Note that a “pre-mounted” FS_B↩ ASED path can be set up by setting this flag to one.
- Volume Name: This is an internal field and should be set to a space character “ “.
- Mount Point Field: This is an internal field and should be set to a space character “ “.
- Block Size Field: This is used to record the block size of the device and does not need to be set by the user.

4 File Descriptors In Osal

The OSAL uses abstracted file descriptors. This means that the file descriptors passed back from the OS_open and OS_creat calls will only work with other OSAL OS_∗ calls. The reasoning for this is as follows:

Because the OSAL now keeps track of all file descriptors, OSAL specific information can be associated with a specific file descriptor in an OS independent way. For instance, the path of the file that the file descriptor points to can be easily retrieved. Also, the OSAL task ID of the task that opened the file can also be retrieved easily. Both of these pieces of information are very useful when trying to determine statistics for a task, or the entire system. This information can all be retrieved with a single API, OS_FDGetInfo.

All of possible file system calls are not implemented. "Special" files requiring OS specific control/operations are by nature not portable. Abstraction in this case is is not possible, so the raw OS calls should be used (including open/close/etc). Mixing with OSAL calls is not supported for such cases. OS_TranslatePath is available to support using open directly by an app and maintain abstraction on the file system.

There are some small drawbacks with the OSAL file descriptors. Because the related information is kept in a table, there is a define called OS_MAX_NUM_OPEN_FILES that defines the maximum number of file descriptors available. This is a configuration parameter, and can be changed to fit your needs.

Also, if you open or create a file not using the OSAL calls (OS_open or OS_creat) then none of the other OS_∗ calls that accept a file descriptor as a parameter will work (the results of doing so are undefined). Therefore, if you open a file with the underlying OS's open call, you must continue to use the OS's calls until you close the file descriptor. Be aware that by doing this your software may no longer be OS agnostic.

5 Timer Overview

The timer API is a generic interface to the OS timer facilities. It is implemented using the POSIX timers on Linux and vxWorks and the native timer API on RTEMS. The number of timers supported is controlled by the configuration parameter OS_MAX_TIMERS.

6 Deprecated List

Global boolean

Use bool

Global FALSE

Use false

Global os_dirp_t

Global OS_ExcAttachHandler (uint32 ExceptionNumber, void(∗ExceptionHandler)(uint32, const void ∗, uint32), int32 parameter)

Planning move to PSP due to platform dependencies

Global OS_ExcDisable (int32 ExceptionNumber)

Planning move to PSP due to platform dependencies

Global OS_ExcEnable (int32 ExceptionNumber)

Planning move to PSP due to platform dependencies

Global OS_FDTableEntry

Use OS_file_prop_t

Global OS_FPUExcAttachHandler (uint32 ExceptionNumber, osal_task_entry ExceptionHandler, int32 parameter)

Planning move to PSP due to platform dependencies

Global OS_FPUExcDisable (int32 ExceptionNumber)

Planning move to PSP due to platform dependencies

Global OS_FPUExcEnable (int32 ExceptionNumber)

Planning move to PSP due to platform dependencies

Global OS_FPUExcGetMask (uint32 ∗mask)

Planning move to PSP due to platform dependencies

Global OS_FPUExcSetMask (uint32 mask)

Planning move to PSP due to platform dependencies

Global os_fshealth_t

type no longer used

Global OS_module_record_t

Use OS_module_prop_t

Global OS_opendir (const char ∗path)

Replaced by OS_DirectoryOpen()

Global OS_ShMemAttach (cpuaddr ∗Address, uint32 Id)

Never implemented

Global OS_ShMemCreate (uint32 *Id, uint32 NBytes, const char *SegName)
Never implemented

Global OS_ShMemGetIdByName (uint32 *ShMemId, const char *SegName)
Never implemented

Global OS_ShMemInit (void)
Never implemented

Global OS_ShMemSemGive (uint32 Id)
Never implemented

Global OS_ShMemSemTake (uint32 Id)
Never implemented

Global OS_TaskRegister (void)
Explicit registration call no longer needed

Global osalbool
Use bool

Module OSAPIExc
Planning move to PSP due to platform dependencies

Module OSAPIFPUExc
Planning move to PSP due to platform dependencies

Module OSAPIShMem
Not in current implementations

Global TRUE
Use true

7 Module Index

7.1 Modules

Here is a list of all modules:

8 Data Structure Index

8.1 Data Structures

Here are the data structures with brief descriptions:

9 File Index

9.1 File List

Here is a list of all files with brief descriptions:

10 Module Documentation

10.1 OSAL Object Type Defines

Macros

- #define OS_OBJECT_TYPE_UNDEFINED 0x00
  Object type undefined.
- #define OS_OBJECT_TYPE_OS_TASK 0x01
  Object task type.
- #define OS_OBJECT_TYPE_OS_QUEUE 0x02
  Object queue type.
- #define OS_OBJECT_TYPE_OS_COUNTSEM 0x03
  Object counting semaphore type.
- #define OS_OBJECT_TYPE_OS_BINSEM 0x04
  Object binary semaphore type.
- #define OS_OBJECT_TYPE_OS_MUTEX 0x05
  Object mutex type.
- #define OS_OBJECT_TYPE_OS_STREAM 0x06
  Object stream type.
- #define OS_OBJECT_TYPE_OS_DIR 0x07
  Object directory type.
- #define OS_OBJECT_TYPE_OS_TIMEBASE 0x08
  Object timebase type.
- #define OS_OBJECT_TYPE_OS_TIMECB 0x09
  Object timer callback type.
- #define OS_OBJECT_TYPE_OS_MODULE 0x0A
  Object module type.

- #define OS_OBJECT_TYPE_OS_FILESYS 0x0B
  Object file system type.
- #define OS_OBJECT_TYPE_OS_CONSOLE 0x0C
  Object console type.
- #define OS_OBJECT_TYPE_USER 0x10
  Object user type.

10.1.1 Detailed Description

10.1.2 Macro Definition Documentation

10.1.2.1 OS_OBJECT_TYPE_OS_BINSEM

```
#define OS_OBJECT_TYPE_OS_BINSEM 0x04
```

Object binary semaphore type.

Definition at line 35 of file osapi-os-core.h.

10.1.2.2 OS_OBJECT_TYPE_OS_CONSOLE

```
#define OS_OBJECT_TYPE_OS_CONSOLE 0x0C
```

Object console type.

Definition at line 43 of file osapi-os-core.h.

10.1.2.3 OS_OBJECT_TYPE_OS_COUNTSEM

```
#define OS_OBJECT_TYPE_OS_COUNTSEM 0x03
```

Object counting semaphore type.

Definition at line 34 of file osapi-os-core.h.

10.1.2.4 OS_OBJECT_TYPE_OS_DIR

```
#define OS_OBJECT_TYPE_OS_DIR 0x07
```

Object directory type.

Definition at line 38 of file osapi-os-core.h.

10.1.2.5 OS_OBJECT_TYPE_OS_FILESYS

```
#define OS_OBJECT_TYPE_OS_FILESYS 0x0B
```

Object file system type.

Definition at line 42 of file osapi-os-core.h.

10.1.2.6 OS_OBJECT_TYPE_OS_MODULE

```
#define OS_OBJECT_TYPE_OS_MODULE 0x0A
```

Object module type.

Definition at line 41 of file osapi-os-core.h.

10.1.2.7 OS_OBJECT_TYPE_OS_MUTEX

```
#define OS_OBJECT_TYPE_OS_MUTEX 0x05
```

Object mutex type.

Definition at line 36 of file osapi-os-core.h.

10.1.2.8 OS_OBJECT_TYPE_OS_QUEUE

```
#define OS_OBJECT_TYPE_OS_QUEUE 0x02
```

Object queue type.

Definition at line 33 of file osapi-os-core.h.

10.1.2.9 OS_OBJECT_TYPE_OS_STREAM

```
#define OS_OBJECT_TYPE_OS_STREAM 0x06
```

Object stream type.

Definition at line 37 of file osapi-os-core.h.

10.1.2.10 OS_OBJECT_TYPE_OS_TASK

```
#define OS_OBJECT_TYPE_OS_TASK 0x01
```

Object task type.

Definition at line 32 of file osapi-os-core.h.

10.1.2.11 OS_OBJECT_TYPE_OS_TIMEBASE

```
#define OS_OBJECT_TYPE_OS_TIMEBASE 0x08
```

Object timebase type.

Definition at line 39 of file osapi-os-core.h.

10.1.2.12 OS_OBJECT_TYPE_OS_TIMECB

```
#define OS_OBJECT_TYPE_OS_TIMECB 0x09
```

Object timer callback type.

Definition at line 40 of file osapi-os-core.h.

10.1.2.13 OS_OBJECT_TYPE_UNDEFINED

```
#define OS_OBJECT_TYPE_UNDEFINED 0x00
```

Object type undefined.

Definition at line 31 of file osapi-os-core.h.

10.1.2.14 OS_OBJECT_TYPE_USER

```
#define OS_OBJECT_TYPE_USER 0x10
```

Object user type.

Definition at line 44 of file osapi-os-core.h.

10.2 OSAL Semaphore State Defines

Macros

- #define OS_SEM_FULL 1

  Semaphore full state.
- #define OS_SEM_EMPTY 0

  Semaphore empty state.

10.2.1 Detailed Description

10.2.2 Macro Definition Documentation

10.2.2.1 OS_SEM_EMPTY

```
#define OS_SEM_EMPTY 0
```

Semaphore empty state.

Definition at line 54 of file osapi-os-core.h.

10.2.2.2 OS_SEM_FULL

```
#define OS_SEM_FULL 1
```

Semaphore full state.

Definition at line 53 of file osapi-os-core.h.

10.3 OSAL Core Operation APIs

Functions

- void OS_Application_Startup (void)

  Application startup.
- void OS_Application_Run (void)

  Application run.
- int32 OS_API_Init (void)

  Initialization of API.
- void OS_IdleLoop (void)

  Background thread implementation - waits forever for events to occur.
- void OS_DeleteAllObjects (void)

  delete all resources created in OSAL.
- void OS_ApplicationShutdown (uint8 flag)

  Initiate orderly shutdown.
- void OS_ApplicationExit (int32 Status)

  Exit/Abort the application.

10.3.1 Detailed Description

These are for OSAL core operations for startup/initialization, running, and shutdown. Typically only used in bsps, unit tests, psps, etc.

Not intended for user application use

10.3.2 Function Documentation

10.3.2.1 OS_API_Init()

```
int32 OS_API_Init (
            void  )
```

Initialization of API.

This function returns initializes the internal data structures of the OS Abstraction Layer. It must be called in the application startup code before calling any other OS routines.

Returns

Execution status, see OSAL Return Code Defines. Any error code (negative) means the OSAL can not be initialized. Typical platform specific response is to abort since additional OSAL calls will have undefined behavior.

Return values

OS_SUCCESS	Successful execution.
OS_ERROR	Failed execution.

10.3.2.2 OS_Application_Run()

```
void OS_Application_Run (
            void  )
```

Application run.

Run abstraction such that the same BSP can be used for operations and testing.

10.3.2.3 OS_Application_Startup()

```
void OS_Application_Startup (
            void  )
```

Application startup.

Startup abstraction such that the same BSP can be used for operations and testing.

10.3.2.4 OS_ApplicationExit()

```
void OS_ApplicationExit (
            int32 Status )
```

Exit/Abort the application.

Indicates that the OSAL application should exit and return control to the OS This is intended for e.g. scripted unit testing where the test needs to end without user intervention.

This function does not return. Production code typically should not ever call this.

Note

This exits the entire process including tasks that have been created.

10.3.2.5 OS_ApplicationShutdown()

```
void OS_ApplicationShutdown (
            uint8 flag )
```

Initiate orderly shutdown.

Indicates that the OSAL application should perform an orderly shutdown of ALL tasks, clean up all resources, and exit the application.

This allows the task currently blocked in OS_IdleLoop() to wake up, and for that function to return to its caller.

This is preferred over e.g. OS_ApplicationExit() which exits immediately and does not provide for any means to clean up first.

Parameters

in	*flag*	set to true to initiate shutdown, false to cancel

10.3.2.6 OS_DeleteAllObjects()

```
void OS_DeleteAllObjects (
            void  )
```

delete all resources created in OSAL.

provides a means to clean up all resources allocated by this instance of OSAL. It would typically be used during an orderly shutdown but may also be helpful for testing purposes.

10.3.2.7 OS_IdleLoop()

```
void OS_IdleLoop (
            void  )
```

Background thread implementation - waits forever for events to occur.

This should be called from the BSP main routine or initial thread after all other board and application initialization has taken place and all other tasks are running.

Typically just waits forever until "OS_shutdown" flag becomes true.

10.4 OSAL Object Utility APIs

Functions

- uint32 OS_IdentifyObject (uint32 object_id)

  Obtain the type of an object given an arbitrary object ID.
- int32 OS_ConvertToArrayIndex (uint32 object_id, uint32 ∗ArrayIndex)

  Converts an abstract ID into a number suitable for use as an array index.
- void OS_ForEachObject (uint32 creator_id, OS_ArgCallback_t callback_ptr, void ∗callback_arg)

  call the supplied callback function for all valid object IDs

10.4.1 Detailed Description

10.4.2 Function Documentation

10.4.2.1 OS_ConvertToArrayIndex()

```
int32 OS_ConvertToArrayIndex (
            uint32 object_id,
            uint32 * ArrayIndex )
```

Converts an abstract ID into a number suitable for use as an array index.

This will return a unique zero-based integer number in the range of [0,MAX) for any valid object ID. This may be used by application code as an array index for indexing into local tables.

Note

This does NOT verify the validity of the ID, that is left to the caller. This is only the conversion logic.

Parameters

in	*object_id*	The object ID to operate on
out	∗*ArrayIndex*	The Index to return

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INCORRECT_OBJ_TYPE	Incorrect object type.

10.4.2.2 OS_ForEachObject()

```
void OS_ForEachObject (
             uint32 creator_id,
             OS_ArgCallback_t callback_ptr,
             void * callback_arg )
```

call the supplied callback function for all valid object IDs

Loops through all defined OSAL objects and calls callback_ptr on each one If creator_id is nonzero then only objects with matching creator id are processed.

10.4.2.3 OS_IdentifyObject()

```
uint32 OS_IdentifyObject (
             uint32 object_id )
```

Obtain the type of an object given an arbitrary object ID.

Given an arbitrary object ID, get the type of the object

Parameters

in	*object_id*	The object ID to operate on

Returns

The object type portion of the object_id, see OSAL Object Type Defines for expected values

10.5 OSAL Task APIs

Functions

- int32 OS_TaskCreate (uint32 *task_id, const char *task_name, osal_task_entry function_pointer, uint32 *stack_pointer, uint32 stack_size, uint32 priority, uint32 flags)

  Creates a task and starts running it.
- int32 OS_TaskDelete (uint32 task_id)

  Deletes the specified Task.
- void OS_TaskExit (void)

  Exits the calling task.
- int32 OS_TaskInstallDeleteHandler (osal_task_entry function_pointer)

  Installs a handler for when the task is deleted.
- int32 OS_TaskDelay (uint32 millisecond)

  Delay a task for specified amount of milliseconds.
- int32 OS_TaskSetPriority (uint32 task_id, uint32 new_priority)

  Sets the given task to a new priority.
- int32 OS_TaskRegister (void)

  Obsolete.
- uint32 OS_TaskGetId (void)

  Obtain the task id of the calling task.
- int32 OS_TaskGetIdByName (uint32 *task_id, const char *task_name)

  Find an existing task ID by name.
- int32 OS_TaskGetInfo (uint32 task_id, OS_task_prop_t *task_prop)

  Fill a property object buffer with details regarding the resource.

10.5.1 Detailed Description

10.5.2 Function Documentation

10.5.2.1 OS_TaskCreate()

```
int32 OS_TaskCreate (
            uint32 * task_id,
            const char * task_name,
            osal_task_entry function_pointer,
            uint32 * stack_pointer,
            uint32 stack_size,
            uint32 priority,
            uint32 flags )
```

Creates a task and starts running it.

Creates a task and passes back the id of the task created. Task names must be unique; if the name already exists this function fails. Names cannot be NULL.

Parameters

out	*task_id*	will be set to the ID of the newly-created resource
in	*task_name*	the name of the new resource to create
in	*function_pointer*	the entry point of the new task
in	*stack_pointer*	pointer to the stack for the task, or NULL to allocate a stack from the system memory heap
in	*stack_size*	the size of the stack, or 0 to use a default stack size.
in	*priority*	initial priority of the new task
in	*flags*	initial options for the new task

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if any of the necessary pointers are NULL
OS_ERR_NAME_TOO_LONG	if the name of the task is too long to be copied
OS_ERR_INVALID_PRIORITY	if the priority is bad
OS_ERR_NO_FREE_IDS	if there can be no more tasks created
OS_ERR_NAME_TAKEN	if the name specified is already used by a task
OS_ERROR	if an unspecified/other error occurs

10.5.2.2 OS_TaskDelay()

```
int32 OS_TaskDelay (
            uint32 millisecond )
```

Delay a task for specified amount of milliseconds.

Causes the current thread to be suspended from execution for the period of millisecond.

Parameters

in	*millisecond*	Amount of time to delay

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERROR	if sleep fails or millisecond = 0

10.5.2.3 OS_TaskDelete()

```
int32 OS_TaskDelete (
            uint32 task_id )
```

Deletes the specified Task.

The task will be removed from the local tables. and the OS will be configured to stop executing the task at the next opportunity.

Parameters

in	*task_id*	The object ID to operate on

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the ID given to it is invalid
OS_ERROR	if the OS delete call fails

10.5.2.4 OS_TaskExit()

```
void OS_TaskExit (
            void  )
```

Exits the calling task.

The calling thread is terminated. This function does not return.

10.5.2.5 OS_TaskGetId()

```
uint32 OS_TaskGetId (
            void  )
```

Obtain the task id of the calling task.

This function returns the task id of the calling task

Returns

Task ID, or zero if the operation failed (zero is never a valid task ID)

10.5.2.6 OS_TaskGetIdByName()

```
int32 OS_TaskGetIdByName (
            uint32 * task_id,
            const char * task_name )
```

Find an existing task ID by name.

This function tries to find a task Id given the name of a task

Parameters

out	*task_id*	will be set to the ID of the existing resource
in	*task_name*	the name of the existing resource to find

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if the pointers passed in are NULL
OS_ERR_NAME_TOO_LONG	if the name to found is too long to begin with
OS_ERR_NAME_NOT_FOUND	if the name wasn't found in the table

10.5.2.7 OS_TaskGetInfo()

```
int32 OS_TaskGetInfo (
            uint32 task_id,
            OS_task_prop_t * task_prop )
```

Fill a property object buffer with details regarding the resource.

This function will pass back a pointer to structure that contains all of the relevant info (creator, stack size, priority, name) about the specified task.

Parameters

in	*task_id*	The object ID to operate on
out	*task_prop*	The property object buffer to fill

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the ID passed to it is invalid
OS_INVALID_POINTER	if the task_prop pointer is NULL

10.5.2.8 OS_TaskInstallDeleteHandler()

```
int32 OS_TaskInstallDeleteHandler (
            osal_task_entry function_pointer )
```

Installs a handler for when the task is deleted.

This function is used to install a callback that is called when the task is deleted. The callback is called when OS_Task↩ Delete is called with the task ID. A task delete handler is useful for cleaning up resources that a task creates, before the task is removed from the system.

Parameters

in	*function_pointer*	function to be called when task exits

Returns

Execution status, see OSAL Return Code Defines

10.5.2.9 OS_TaskRegister()

```
int32 OS_TaskRegister (
            void  )
```

Obsolete.

Deprecated Explicit registration call no longer needed

Obsolete function retained for compatibility purposes. Does Nothing in the current implementation.

Returns

OS_SUCCESS (always), see OSAL Return Code Defines

10.5.2.10 OS_TaskSetPriority()

```
int32 OS_TaskSetPriority (
            uint32 task_id,
            uint32 new_priority )
```

Sets the given task to a new priority.

Parameters

in	*task_id*	The object ID to operate on
in	*new_priority*	Set the new priority

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the ID passed to it is invalid
OS_ERR_INVALID_PRIORITY	if the priority is greater than the max allowed
OS_ERROR	if the OS call to change the priority fails

10.6 OSAL Message Queue APIs

Functions

- int32 OS_QueueCreate (uint32 *queue_id, const char *queue_name, uint32 queue_depth, uint32 data_size, uint32 flags)

  Create a message queue.
- int32 OS_QueueDelete (uint32 queue_id)

  Deletes the specified message queue.
- int32 OS_QueueGet (uint32 queue_id, void *data, uint32 size, uint32 *size_copied, int32 timeout)

  Receive a message on a message queue.
- int32 OS_QueuePut (uint32 queue_id, const void *data, uint32 size, uint32 flags)

  Put a message on a message queue.
- int32 OS_QueueGetIdByName (uint32 *queue_id, const char *queue_name)

  Find an existing queue ID by name.
- int32 OS_QueueGetInfo (uint32 queue_id, OS_queue_prop_t *queue_prop)

  Fill a property object buffer with details regarding the resource.

10.6.1 Detailed Description

10.6.2 Function Documentation

10.6.2.1 OS_QueueCreate()

```
int32 OS_QueueCreate (
            uint32 * queue_id,
            const char * queue_name,
            uint32 queue_depth,
            uint32 data_size,
            uint32 flags )
```

Create a message queue.

This is the function used to create a queue in the operating system. Depending on the underlying operating system, the memory for the queue will be allocated automatically or allocated by the code that sets up the queue. Queue names must be unique; if the name already exists this function fails. Names cannot be NULL.

Parameters

out	*queue_id*	will be set to the ID of the newly-created resource
in	*queue_name*	the name of the new resource to create
in	*queue_depth*	the maximum depth of the queue
in	*data_size*	the size of each entry in the queue
in	*flags*	options for the queue (reserved for future use, pass as 0)

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if a pointer passed in is NULL
OS_ERR_NAME_TOO_LONG	if the name passed in is too long
OS_ERR_NO_FREE_IDS	if there are already the max queues created
OS_ERR_NAME_TAKEN	if the name is already being used on another queue
OS_ERROR	if the OS create call fails

10.6.2.2 OS_QueueDelete()

```
int32 OS_QueueDelete (
            uint32 queue_id )
```

Deletes the specified message queue.

This is the function used to delete a queue in the operating system. This also frees the respective queue_id to be used again when another queue is created.

Note

If There are messages on the queue, they will be lost and any subsequent calls to QueueGet or QueuePut to this queue will result in errors

Parameters

in	*queue_id*	The object ID to delete

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the id passed in does not exist
OS_ERROR	if the OS call to delete the queue fails

10.6.2.3 OS_QueueGet()

```
int32 OS_QueueGet (
            uint32 queue_id,
            void * data,
            uint32 size,
            uint32 * size_copied,
            int32 timeout )
```

Receive a message on a message queue.

If a message is pending, it is returned immediately. Otherwise the calling task will block until a message arrives or the timeout expires.

Parameters

in	*queue_id*	The object ID to operate on
out	*data*	The buffer to store the received message
in	*size*	The size of the data buffer
out	*size_copied*	Set to the actual size of the message
in	*timeout*	The maximum amount of time to block, or OS_PEND to wait forever

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the given ID does not exist
OS_INVALID_POINTER	if a pointer passed in is NULL
OS_QUEUE_EMPTY	if the Queue has no messages on it to be recieved
OS_QUEUE_TIMEOUT	if the timeout was OS_PEND and the time expired
OS_QUEUE_INVALID_SIZE	if the size copied from the queue was not correct

10.6.2.4 OS_QueueGetIdByName()

```
int32 OS_QueueGetIdByName (
            uint32 * queue_id,
            const char * queue_name )
```

Find an existing queue ID by name.

This function tries to find a queue Id given the name of the queue. The id of the queue is passed back in queue_id.

Parameters

out	*queue_id*	will be set to the ID of the existing resource
in	*queue_name*	the name of the existing resource to find

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if the name or id pointers are NULL
OS_ERR_NAME_TOO_LONG	the name passed in is too long
OS_ERR_NAME_NOT_FOUND	the name was not found in the table

10.6.2.5 OS_QueueGetInfo()

```
int32 OS_QueueGetInfo (
            uint32 queue_id,
            OS_queue_prop_t * queue_prop )
```

Fill a property object buffer with details regarding the resource.

This function will pass back a pointer to structure that contains all of the relevant info (name and creator) about the specified queue.

Parameters

in	*queue_id*	The object ID to operate on
out	*queue_prop*	The property object buffer to fill

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if queue_prop is NULL
OS_ERR_INVALID_ID	if the ID given is not a valid queue
OS_SUCCESS	if the info was copied over correctly

10.6.2.6 OS_QueuePut()

```
int32 OS_QueuePut (
            uint32 queue_id,
            const void * data,
            uint32 size,
            uint32 flags )
```

Put a message on a message queue.

Parameters

in	*queue_id*	The object ID to operate on
in	*data*	The buffer containing the message to put
in	*size*	The size of the data buffer
in	*flags*	Currently reserved/unused, should be passed as 0

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the queue id passed in is not a valid queue
OS_INVALID_POINTER	if the data pointer is NULL
OS_QUEUE_FULL	if the queue cannot accept another message
OS_ERROR	if the OS call returns an error

10.7 OSAL Semaphore APIs

Functions

- int32 OS_BinSemCreate (uint32 *sem_id, const char *sem_name, uint32 sem_initial_value, uint32 options)

  Creates a binary semaphore.
- int32 OS_BinSemFlush (uint32 sem_id)

  Unblock all tasks pending on the specified semaphore.
- int32 OS_BinSemGive (uint32 sem_id)

  Increment the semaphore value.
- int32 OS_BinSemTake (uint32 sem_id)

  Decrement the semaphore value.
- int32 OS_BinSemTimedWait (uint32 sem_id, uint32 msecs)

  Decrement the semaphore value with a timeout.
- int32 OS_BinSemDelete (uint32 sem_id)

  Deletes the specified Binary Semaphore.
- int32 OS_BinSemGetIdByName (uint32 *sem_id, const char *sem_name)

  Find an existing semaphore ID by name.
- int32 OS_BinSemGetInfo (uint32 sem_id, OS_bin_sem_prop_t *bin_prop)

  Fill a property object buffer with details regarding the resource.
- int32 OS_CountSemCreate (uint32 *sem_id, const char *sem_name, uint32 sem_initial_value, uint32 options)

  Creates a counting semaphore.
- int32 OS_CountSemGive (uint32 sem_id)

  Increment the semaphore value.
- int32 OS_CountSemTake (uint32 sem_id)

  Decrement the semaphore value.
- int32 OS_CountSemTimedWait (uint32 sem_id, uint32 msecs)

  Decrement the semaphore value with timeout.
- int32 OS_CountSemDelete (uint32 sem_id)

  Deletes the specified counting Semaphore.
- int32 OS_CountSemGetIdByName (uint32 *sem_id, const char *sem_name)

  Find an existing semaphore ID by name.
- int32 OS_CountSemGetInfo (uint32 sem_id, OS_count_sem_prop_t *count_prop)

  Fill a property object buffer with details regarding the resource.
- int32 OS_MutSemCreate (uint32 *sem_id, const char *sem_name, uint32 options)

  Creates a mutex semaphore.
- int32 OS_MutSemGive (uint32 sem_id)

  Releases the mutex object referenced by sem_id.
- int32 OS_MutSemTake (uint32 sem_id)

  Acquire the mutex object referenced by sem_id.
- int32 OS_MutSemDelete (uint32 sem_id)

  Deletes the specified Mutex Semaphore.
- int32 OS_MutSemGetIdByName (uint32 *sem_id, const char *sem_name)

  Find an existing mutex ID by name.
- int32 OS_MutSemGetInfo (uint32 sem_id, OS_mut_sem_prop_t *mut_prop)

  Fill a property object buffer with details regarding the resource.

10.7.1 Detailed Description

10.7.2 Function Documentation

10.7.2.1 OS_BinSemCreate()

```
int32 OS_BinSemCreate (
            uint32 * sem_id,
            const char * sem_name,
            uint32 sem_initial_value,
            uint32 options )
```

Creates a binary semaphore.

Creates a binary semaphore with initial value specified by sem_initial_value and name specified by sem_name. sem_id will be returned to the caller

Parameters

out	*sem_id*	will be set to the ID of the newly-created resource
in	*sem_name*	the name of the new resource to create
in	*sem_initial_value*	the initial value of the binary semaphore
in	*options*	Reserved for future use, should be passed as 0.

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if sen name or sem_id are NULL
OS_ERR_NAME_TOO_LONG	if the name given is too long
OS_ERR_NO_FREE_IDS	if all of the semaphore ids are taken
OS_ERR_NAME_TAKEN	if this is already the name of a binary semaphore
OS_SEM_FAILURE	if the OS call failed

10.7.2.2 OS_BinSemDelete()

```
int32 OS_BinSemDelete (
            uint32 sem_id )
```

Deletes the specified Binary Semaphore.

This is the function used to delete a binary semaphore in the operating system. This also frees the respective sem_id to be used again when another semaphore is created.

Parameters

in	*sem↩ _id*	The object ID to delete

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the id passed in is not a valid binary semaphore
OS_SEM_FAILURE	the OS call failed

10.7.2.3 OS_BinSemFlush()

```
int32 OS_BinSemFlush (
            uint32 sem_id )
```

Unblock all tasks pending on the specified semaphore.

The function unblocks all tasks pending on the specified semaphore. However, this function does not change the state of the semaphore.

Parameters

in	*sem↩ _id*	The object ID to operate on

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the id passed in is not a binary semaphore
OS_SEM_FAILURE	if an unspecified failure occurs

10.7.2.4 OS_BinSemGetIdByName()

```
int32 OS_BinSemGetIdByName (
```

```
            uint32 * sem_id,
            const char * sem_name )
```

Find an existing semaphore ID by name.

This function tries to find a binary sem Id given the name of a bin_sem The id is returned through sem_id

Parameters

out	*sem_id*	will be set to the ID of the existing resource
in	*sem_name*	the name of the existing resource to find

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	is semid or sem_name are NULL pointers
OS_ERR_NAME_TOO_LONG	if the name given is to long to have been stored
OS_ERR_NAME_NOT_FOUND	if the name was not found in the table

10.7.2.5 OS_BinSemGetInfo()

```
int32 OS_BinSemGetInfo (
            uint32 sem_id,
            OS_bin_sem_prop_t * bin_prop )
```

Fill a property object buffer with details regarding the resource.

This function will pass back a pointer to structure that contains all of the relevant info(name and creator) about the specified binary semaphore.

Parameters

in	*sem_id*	The object ID to operate on
out	*bin_prop*	The property object buffer to fill

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the id passed in is not a valid semaphore
OS_INVALID_POINTER	if the bin_prop pointer is null

10.7.2.6 OS_BinSemGive()

```
int32 OS_BinSemGive (
            uint32 sem_id )
```

Increment the semaphore value.

The function unlocks the semaphore referenced by sem_id by performing a semaphore unlock operation on that semaphore. If the semaphore value resulting from this operation is positive, then no threads were blocked waiting for the semaphore to become unlocked; the semaphore value is simply incremented for this semaphore.

Parameters

in	*sem_id*	The object ID to operate on

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_SEM_FAILURE	the semaphore was not previously initialized or is not in the array of semaphores defined by the system
OS_ERR_INVALID_ID	if the id passed in is not a binary semaphore

10.7.2.7 OS_BinSemTake()

```
int32 OS_BinSemTake (
            uint32 sem_id )
```

Decrement the semaphore value.

The locks the semaphore referenced by sem_id by performing a semaphore lock operation on that semaphore. If the semaphore value is currently zero, then the calling thread shall not return from the call until it either locks the semaphore or the call is interrupted.

Parameters

in	*sem↩ _id*	The object ID to operate on

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	the Id passed in is not a valid binary semaphore
OS_SEM_FAILURE	if the OS call failed

10.7.2.8 OS_BinSemTimedWait()

```
int32 OS_BinSemTimedWait (
            uint32 sem_id,
            uint32 msecs )
```

Decrement the semaphore value with a timeout.

The function locks the semaphore referenced by sem_id. However, if the semaphore cannot be locked without waiting for another process or thread to unlock the semaphore, this wait shall be terminated when the specified timeout, msecs, expires.

Parameters

in	*sem↩ _id*	The object ID to operate on
in	*msecs*	The maximum amount of time to block, in milliseconds

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_SEM_TIMEOUT	if semaphore was not relinquished in time
OS_SEM_FAILURE	the semaphore was not previously initialized or is not in the array of semaphores defined by the system
OS_ERR_INVALID_ID	if the ID passed in is not a valid semaphore ID

10.7.2.9 OS_CountSemCreate()

```
int32 OS_CountSemCreate (
            uint32 * sem_id,
            const char * sem_name,
            uint32 sem_initial_value,
            uint32 options )
```

Creates a counting semaphore.

Creates a counting semaphore with initial value specified by sem_initial_value and name specified by sem_name. sem_id will be returned to the caller

Parameters

out	*sem_id*	will be set to the ID of the newly-created resource
in	*sem_name*	the name of the new resource to create
in	*sem_initial_value*	the initial value of the counting semaphore
in	*options*	Reserved for future use, should be passed as 0.

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if sen name or sem_id are NULL
OS_ERR_NAME_TOO_LONG	if the name given is too long
OS_ERR_NO_FREE_IDS	if all of the semaphore ids are taken
OS_ERR_NAME_TAKEN	if this is already the name of a counting semaphore
OS_SEM_FAILURE	if the OS call failed
OS_INVALID_SEM_VALUE	if the semaphore value is too high

10.7.2.10 OS_CountSemDelete()

```
int32 OS_CountSemDelete (
            uint32 sem_id )
```

Deletes the specified counting Semaphore.

Parameters

in	*sem↩ _id*	The object ID to delete

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the id passed in is not a valid counting semaphore
OS_SEM_FAILURE	the OS call failed

10.7.2.11 OS_CountSemGetIdByName()

```
int32 OS_CountSemGetIdByName (
            uint32 * sem_id,
            const char * sem_name )
```

Find an existing semaphore ID by name.

This function tries to find a counting sem Id given the name of a count_sem The id is returned through sem_id

Parameters

out	*sem_id*	will be set to the ID of the existing resource
in	*sem_name*	the name of the existing resource to find

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	is semid or sem_name are NULL pointers
OS_ERR_NAME_TOO_LONG	if the name given is to long to have been stored
OS_ERR_NAME_NOT_FOUND	if the name was not found in the table

10.7.2.12 OS_CountSemGetInfo()

```
int32 OS_CountSemGetInfo (
            uint32 sem_id,
            OS_count_sem_prop_t * count_prop )
```

Fill a property object buffer with details regarding the resource.

This function will pass back a pointer to structure that contains all of the relevant info(name and creator) about the specified counting semaphore.

Parameters

in	*sem_id*	The object ID to operate on
out	*count_prop*	The property object buffer to fill

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the id passed in is not a valid semaphore
OS_INVALID_POINTER	if the count_prop pointer is null

10.7.2.13 OS_CountSemGive()

```
int32 OS_CountSemGive (
            uint32 sem_id )
```

Increment the semaphore value.

The function unlocks the semaphore referenced by sem_id by performing a semaphore unlock operation on that semaphore. If the semaphore value resulting from this operation is positive, then no threads were blocked waiting for the semaphore to become unlocked; the semaphore value is simply incremented for this semaphore.

Parameters

in	*sem_id*	The object ID to operate on

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_SEM_FAILURE	the semaphore was not previously initialized or is not in the array of semaphores defined by the system
OS_ERR_INVALID_ID	if the id passed in is not a counting semaphore

10.7.2.14 OS_CountSemTake()

```
int32 OS_CountSemTake (
            uint32 sem_id )
```

Decrement the semaphore value.

The locks the semaphore referenced by sem_id by performing a semaphore lock operation on that semaphore. If the semaphore value is currently zero, then the calling thread shall not return from the call until it either locks the semaphore or the call is interrupted.

Parameters

in	*sem_id*	The object ID to operate on

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	the Id passed in is not a valid counting semaphore
OS_SEM_FAILURE	if the OS call failed

10.7.2.15 OS_CountSemTimedWait()

```
int32 OS_CountSemTimedWait (
            uint32 sem_id,
            uint32 msecs )
```

Decrement the semaphore value with timeout.

The function locks the semaphore referenced by sem_id. However, if the semaphore cannot be locked without waiting for another process or thread to unlock the semaphore, this wait shall be terminated when the specified timeout, msecs, expires.

Parameters

in	*sem_id*	The object ID to operate on
in	*msecs*	The maximum amount of time to block, in milliseconds

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_SEM_TIMEOUT	if semaphore was not relinquished in time
OS_SEM_FAILURE	the semaphore was not previously initialized or is not in the array of semaphores defined by the system
OS_ERR_INVALID_ID	if the ID passed in is not a valid semaphore ID

10.7.2.16 OS_MutSemCreate()

```
int32 OS_MutSemCreate (
            uint32 * sem_id,
            const char * sem_name,
            uint32 options )
```

Creates a mutex semaphore.

Mutex semaphores are always created in the unlocked (full) state.

Parameters

out	*sem_id*	will be set to the ID of the newly-created resource
in	*sem_name*	the name of the new resource to create
in	*options*	reserved for future use. Should be passed as 0.

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if sem_id or sem_name are NULL
OS_ERR_NAME_TOO_LONG	if the sem_name is too long to be stored

Return values

OS_ERR_NO_FREE_IDS	if there are no more free mutex Ids
OS_ERR_NAME_TAKEN	if there is already a mutex with the same name
OS_SEM_FAILURE	if the OS call failed

10.7.2.17 OS_MutSemDelete()

```
int32 OS_MutSemDelete (
            uint32 sem_id )
```

Deletes the specified Mutex Semaphore.

Delete the semaphore. This also frees the respective sem_id such that it can be used again when another is created.

Parameters

in	*sem_id*	The object ID to delete

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the id passed in is not a valid mutex
OS_SEM_FAILURE	if the OS call failed

10.7.2.18 OS_MutSemGetIdByName()

```
int32 OS_MutSemGetIdByName (
            uint32 * sem_id,
            const char * sem_name )
```

Find an existing mutex ID by name.

This function tries to find a mutex sem Id given the name of a mut_sem. The id is returned through sem_id

Parameters

out	*sem_id*	will be set to the ID of the existing resource
in	*sem_name*	the name of the existing resource to find

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	is semid or sem_name are NULL pointers
OS_ERR_NAME_TOO_LONG	if the name given is to long to have been stored
OS_ERR_NAME_NOT_FOUND	if the name was not found in the table

10.7.2.19 OS_MutSemGetInfo()

```
int32 OS_MutSemGetInfo (
            uint32 sem_id,
            OS_mut_sem_prop_t * mut_prop )
```

Fill a property object buffer with details regarding the resource.

This function will pass back a pointer to structure that contains all of the relevant info(name and creator) about the specified mutex semaphore.

Parameters

in	*sem_id*	The object ID to operate on
out	*mut_prop*	The property object buffer to fill

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the id passed in is not a valid semaphore
OS_INVALID_POINTER	if the mut_prop pointer is null

10.7.2.20 OS_MutSemGive()

```
int32 OS_MutSemGive (
            uint32 sem_id )
```

Releases the mutex object referenced by sem_id.

If there are threads blocked on the mutex object referenced by mutex when this function is called, resulting in the mutex becoming available, the scheduling policy shall determine which thread shall acquire the mutex.

Parameters

in	*sem↩ _id*	The object ID to operate on

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the id passed in is not a valid mutex
OS_SEM_FAILURE	if an unspecified error occurs

10.7.2.21 OS_MutSemTake()

```
int32 OS_MutSemTake (
            uint32 sem_id )
```

Acquire the mutex object referenced by sem_id.

If the mutex is already locked, the calling thread shall block until the mutex becomes available. This operation shall return with the mutex object referenced by mutex in the locked state with the calling thread as its owner.

Parameters

in	*sem↩ _id*	The object ID to operate on

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_SEM_FAILURE	if the semaphore was not previously initialized or is not in the array of semaphores defined by the system
OS_ERR_INVALID_ID	the id passed in is not a valid mutex

10.8 OSAL Time/Tick APIs

Functions

- int32 OS_Milli2Ticks (uint32 milli_seconds)

  Convert time units from milliseconds to system ticks.
- int32 OS_Tick2Micros (void)

  Get the system tick size, in microseconds.
- int32 OS_GetLocalTime (OS_time_t *time_struct)

  Get the local time.
- int32 OS_SetLocalTime (OS_time_t *time_struct)

  Set the local time.

10.8.1 Detailed Description

10.8.2 Function Documentation

10.8.2.1 OS_GetLocalTime()

```
int32 OS_GetLocalTime (
            OS_time_t * time_struct )
```

Get the local time.

This function gets the local time from the underlying OS.

Note

Mission time management typically uses the cFE Time Service

Parameters

out	*time_struct*	An OS_time_t that will be set to the current time

Returns

Get local time status, see OSAL Return Code Defines

10.8.2.2 OS_Milli2Ticks()

```
int32 OS_Milli2Ticks (
            uint32 milli_seconds )
```

Convert time units from milliseconds to system ticks.

This function accepts a time interval in milliseconds and returns the tick equivalent. If the result is not an exact number of system ticks, the result will be rounded up to the nearest tick.

Parameters

in	*milli_seconds*	the number of milliseconds

Returns

The number of ticks

10.8.2.3 OS_SetLocalTime()

```
int32 OS_SetLocalTime (
            OS_time_t * time_struct )
```

Set the local time.

This function sets the local time on the underlying OS.

Note

Mission time management typically uses the cFE Time Services

Parameters

in	*time_struct*	An OS_time_t containing the current time

Returns

Set local time status, see OSAL Return Code Defines

10.8.2.4 OS_Tick2Micros()

```
int32 OS_Tick2Micros (
            void  )
```

Get the system tick size, in microseconds.

This function returns the duration of a system tick in micro seconds

Note

care is taken to ensure this does not return "0" since it is often used as the divisor in mathematical operations

Returns

Duration of a system tick in microseconds

10.9 OSAL Exception APIs

Functions

- int32 OS_ExcAttachHandler (uint32 ExceptionNumber, void(*ExceptionHandler)(uint32, const void *, uint32), int32 parameter)
  placeholder; not currently implemented
- int32 OS_ExcEnable (int32 ExceptionNumber)
  placeholder; not currently implemented
- int32 OS_ExcDisable (int32 ExceptionNumber)
  placeholder; not currently implemented

10.9.1 Detailed Description

Note

Not implemented in current OSAL version

Deprecated Planning move to PSP due to platform dependencies

10.9.2 Function Documentation

10.9.2.1 OS_ExcAttachHandler()

```
int32 OS_ExcAttachHandler (
            uint32 ExceptionNumber,
            void(*)(uint32, const void *, uint32) ExceptionHandler,
            int32 parameter )
```

placeholder; not currently implemented

Deprecated Planning move to PSP due to platform dependencies

10.9.2.2 OS_ExcDisable()

```
int32 OS_ExcDisable (
            int32 ExceptionNumber )
```

placeholder; not currently implemented

Deprecated Planning move to PSP due to platform dependencies

10.9.2.3 OS_ExcEnable()

```
int32 OS_ExcEnable (
            int32 ExceptionNumber )
```

placeholder; not currently implemented

Deprecated Planning move to PSP due to platform dependencies

10.10 OSAL Floating Point Unit Exception APIs

Functions

- int32 OS_FPUExcAttachHandler (uint32 ExceptionNumber, osal_task_entry ExceptionHandler, int32 parameter)

  Set an FPU exception handler function.
- int32 OS_FPUExcEnable (int32 ExceptionNumber)

  Enable FPU exceptions.
- int32 OS_FPUExcDisable (int32 ExceptionNumber)

  Disable FPU exceptions.
- int32 OS_FPUExcSetMask (uint32 mask)

  Sets the FPU exception mask.
- int32 OS_FPUExcGetMask (uint32 *mask)

  Gets the FPU exception mask.

10.10.1 Detailed Description

Deprecated Planning move to PSP due to platform dependencies

10.10.2 Function Documentation

10.10.2.1 OS_FPUExcAttachHandler()

```
int32 OS_FPUExcAttachHandler (
            uint32 ExceptionNumber,
            osal_task_entry ExceptionHandler,
            int32 parameter )
```

Set an FPU exception handler function.

The call associates a specified C routine to a specified FPU exception number. When the specified FPU Exception occurs , the ExceptionHandler routine will be called and passed the parameter.

Deprecated Planning move to PSP due to platform dependencies

Parameters

in	*ExceptionNumber*	The exception number to attach to
in	*ExceptionHandler*	Pointer to handler function
in	*parameter*	Argument to pass to handler

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_NOT_IMPLEMENTED	Not implemented.

10.10.2.2 OS_FPUExcDisable()

```
int32 OS_FPUExcDisable (
            int32 ExceptionNumber )
```

Disable FPU exceptions.

Deprecated Planning move to PSP due to platform dependencies

Parameters

in	*ExceptionNumber*	The exception number to disable

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_NOT_IMPLEMENTED	Not implemented.

10.10.2.3 OS_FPUExcEnable()

```
int32 OS_FPUExcEnable (
            int32 ExceptionNumber )
```

Enable FPU exceptions.

Deprecated Planning move to PSP due to platform dependencies

Parameters

in	*ExceptionNumber*	The exception number to enable

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_NOT_IMPLEMENTED	Not implemented.

10.10.2.4 OS_FPUExcGetMask()

```
int32 OS_FPUExcGetMask (
            uint32 * mask )
```

Gets the FPU exception mask.

Deprecated Planning move to PSP due to platform dependencies

This function gets the FPU exception mask

Note

The exception environment is local to each task Therefore this must be called for each task that that wants to do floating point and catch exceptions.

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_NOT_IMPLEMENTED	Not implemented.

10.10.2.5 OS_FPUExcSetMask()

```
int32 OS_FPUExcSetMask (
```

```
            uint32 mask )
```

Sets the FPU exception mask.

Deprecated Planning move to PSP due to platform dependencies

This function sets the FPU exception mask

Note

The exception environment is local to each task Therefore this must be called for each task that that wants to do floating point and catch exceptions.

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_NOT_IMPLEMENTED	Not implemented.

10.11 OSAL Interrupt APIs

Functions

- int32 OS_IntAttachHandler (uint32 InterruptNumber, osal_task_entry InterruptHandler, int32 parameter)

  Associate an interrupt number to a specified handler routine.
- int32 OS_IntUnlock (int32 IntLevel)

  Enable interrupts.
- int32 OS_IntLock (void)

  Disable interrupts.
- int32 OS_IntEnable (int32 Level)

  Enables interrupts through Level.
- int32 OS_IntDisable (int32 Level)

  Disable interrupts through Level.
- int32 OS_IntSetMask (uint32 mask)

  Set the CPU interrupt mask register.
- int32 OS_IntGetMask (uint32 *mask)

  Get the CPU interrupt mask register.
- int32 OS_IntAck (int32 InterruptNumber)

  Acknowledge the corresponding interrupt number.

10.11.1 Detailed Description

10.11.2 Function Documentation

10.11.2.1 OS_IntAck()

```
int32 OS_IntAck (
            int32 InterruptNumber )
```

Acknowledge the corresponding interrupt number.

Note

: placeholder; not currently implemented in sample implementations

Parameters

in	*InterruptNumber*	The interrupt number to be acknowledged.

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_INT_NUM	Invalid Interrupt number.
OS_ERR_NOT_IMPLEMENTED	Not implemented.

10.11.2.2 OS_IntAttachHandler()

```
int32 OS_IntAttachHandler (
            uint32 InterruptNumber,
            osal_task_entry InterruptHandler,
            int32 parameter )
```

Associate an interrupt number to a specified handler routine.

The call associates a specified C routine to a specified interrupt number. Upon occurring of the InterruptNumber, the InerruptHandler routine will be called and passed the parameter.

Parameters

in	*InterruptNumber*	The Interrupt Number that will cause the start of the ISR
in	*InterruptHandler*	The ISR associated with this interrupt
in	*parameter*	Argument that is passed to the ISR

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	The Interrupt handler pointer is NULL
OS_ERR_NOT_IMPLEMENTED	Not implemented.

10.11.2.3 OS_IntDisable()

```
int32 OS_IntDisable (
            int32 Level )
```

Disable interrupts through Level.

Parameters

in	*Level*	the interrupts to disable

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_NOT_IMPLEMENTED	Not implemented.

10.11.2.4 OS_IntEnable()

```
int32 OS_IntEnable (
            int32 Level )
```

Enables interrupts through Level.

Parameters

in	*Level*	the interrupts to enable

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_NOT_IMPLEMENTED	Not implemented.

10.11.2.5 OS_IntGetMask()

```
int32 OS_IntGetMask (
            uint32 * mask )
```

Get the CPU interrupt mask register.

Note

The interrupt bits are architecture-specific.

Parameters

out	*mask*	The register value will be stored to this location

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_NOT_IMPLEMENTED	Not implemented.

10.11.2.6 OS_IntLock()

```
int32 OS_IntLock (
            void  )
```

Disable interrupts.

Returns

An key value to be passed to OS_IntUnlock() to restore interrupts or error status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_NOT_IMPLEMENTED	Not implemented.

10.11.2.7 OS_IntSetMask()

```
int32 OS_IntSetMask (
            uint32 mask )
```

Set the CPU interrupt mask register.

Note

The interrupt bits are architecture-specific.

Parameters

in	*mask*	The value to set in the register

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_NOT_IMPLEMENTED	Not implemented.

10.11.2.8 OS_IntUnlock()

```
int32 OS_IntUnlock (
            int32 IntLevel )
```

Enable interrupts.

Parameters

in	*IntLevel*	value from previous call to OS_IntLock()

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_NOT_IMPLEMENTED	Not implemented.

10.12 OSAL Shared memory APIs

Functions

- int32 OS_ShMemInit (void)

  placeholder; not currently implemented
- int32 OS_ShMemCreate (uint32 ∗Id, uint32 NBytes, const char ∗SegName)

  placeholder; not currently implemented
- int32 OS_ShMemSemTake (uint32 Id)

  placeholder; not currently implemented
- int32 OS_ShMemSemGive (uint32 Id)

  placeholder; not currently implemented
- int32 OS_ShMemAttach (cpuaddr ∗Address, uint32 Id)

  placeholder; not currently implemented
- int32 OS_ShMemGetIdByName (uint32 ∗ShMemId, const char ∗SegName)

  placeholder; not currently implemented

10.12.1 Detailed Description

Deprecated Not in current implementations

10.12.2 Function Documentation

10.12.2.1 OS_ShMemAttach()

```
int32 OS_ShMemAttach (
            cpuaddr * Address,
            uint32 Id )
```

placeholder; not currently implemented

Deprecated Never implemented

10.12.2.2 OS_ShMemCreate()

```
int32 OS_ShMemCreate (
            uint32 * Id,
            uint32 NBytes,
            const char * SegName )
```

placeholder; not currently implemented

Deprecated Never implemented

10.12.2.3 OS_ShMemGetIdByName()

```
int32 OS_ShMemGetIdByName (
            uint32 * ShMemId,
            const char * SegName )
```

placeholder; not currently implemented

Deprecated Never implemented

10.12.2.4 OS_ShMemInit()

```
int32 OS_ShMemInit (
            void  )
```

placeholder; not currently implemented

Deprecated Never implemented

10.12.2.5 OS_ShMemSemGive()

```
int32 OS_ShMemSemGive (
            uint32 Id )
```

placeholder; not currently implemented

Deprecated Never implemented

10.12.2.6 OS_ShMemSemTake()

```
int32 OS_ShMemSemTake (
            uint32 Id )
```

placeholder; not currently implemented

Deprecated Never implemented

10.13 OSAL Heap APIs

Functions

- int32 OS_HeapGetInfo (OS_heap_prop_t *heap_prop)

  Return current info on the heap.

10.13.1 Detailed Description

10.13.2 Function Documentation

10.13.2.1 OS_HeapGetInfo()

```
int32 OS_HeapGetInfo (
            OS_heap_prop_t * heap_prop )
```

Return current info on the heap.

Parameters

out	*heap_prop*	Storage buffer for heap info

Returns

Execution status, see OSAL Return Code Defines

10.14 OSAL Error Info APIs

Functions

- int32 OS_GetErrorName (int32 error_num, os_err_name_t *err_name)

  Convert an error number to a string.

10.14.1 Detailed Description

10.14.2 Function Documentation

10.14.2.1 OS_GetErrorName()

```
int32 OS_GetErrorName (
            int32 error_num,
            os_err_name_t * err_name )
```

Convert an error number to a string.

Parameters

in	*error_num*	Error number to convert
out	*err_name*	Buffer to store error string

Returns

Execution status, see OSAL Return Code Defines

10.15 OSAL Select APIs

Functions

- int32 OS_SelectMultiple (OS_FdSet *ReadSet, OS_FdSet *WriteSet, int32 msecs)

  Wait for events across multiple file handles.
- int32 OS_SelectSingle (uint32 objid, uint32 *StateFlags, int32 msecs)

  Wait for events on a single file handle.
- int32 OS_SelectFdZero (OS_FdSet *Set)

  Clear a FdSet structure.
- int32 OS_SelectFdAdd (OS_FdSet *Set, uint32 objid)

  Add an ID to an FdSet structure.
- int32 OS_SelectFdClear (OS_FdSet *Set, uint32 objid)

  Clear an ID from an FdSet structure.
- bool OS_SelectFdIsSet (OS_FdSet *Set, uint32 objid)

  Check if an FdSet structure contains a given ID.

10.15.1 Detailed Description

10.15.2 Function Documentation

10.15.2.1 OS_SelectFdAdd()

```
int32 OS_SelectFdAdd (
            OS_FdSet * Set,
            uint32 objid )
```

Add an ID to an FdSet structure.

After this call the set will contain the given OSAL ID

Returns

Execution status, see OSAL Return Code Defines

10.15.2.2 OS_SelectFdClear()

```
int32 OS_SelectFdClear (
            OS_FdSet * Set,
            uint32 objid )
```

Clear an ID from an FdSet structure.

After this call the set will no longer contain the given OSAL ID

Returns

Execution status, see OSAL Return Code Defines

10.15.2.3 OS_SelectFdIsSet()

```
bool OS_SelectFdIsSet (
            OS_FdSet * Set,
            uint32 objid )
```

Check if an FdSet structure contains a given ID.

Returns

Boolean set status

Return values

true	FdSet structure contains ID
false	FDSet structure does not contain ID

10.15.2.4 OS_SelectFdZero()

```
int32 OS_SelectFdZero (
            OS_FdSet * Set )
```

Clear a FdSet structure.

After this call the set will contain no OSAL IDs

Returns

Execution status, see OSAL Return Code Defines

10.15.2.5 OS_SelectMultiple()

```
int32 OS_SelectMultiple (
            OS_FdSet * ReadSet,
            OS_FdSet * WriteSet,
            int32 msecs )
```

Wait for events across multiple file handles.

Wait for any of the given sets of IDs to be become readable or writable

This function will block until any of the following occurs:

- At least one OSAL ID in the ReadSet is readable
- At least one OSAL ID in the WriteSet is writable
- The timeout has elapsed

The sets are input/output parameters. On entry, these indicate the file handle(s) to wait for. On exit, these are set to the actual file handle(s) that have activity.

If the timeout occurs this returns an error code and all output sets should be empty.

Note

This does not lock or otherwise protect the file handles in the given sets. If a filehandle supplied via one of the FdSet arguments is closed or modified by another while this function is in progress, the results are undefined. Because of this limitation, it is recommended to use OS_SelectSingle() whenever possible.

Returns

Execution status, see OSAL Return Code Defines

10.15.2.6 OS_SelectSingle()

```
int32 OS_SelectSingle (
            uint32 objid,
            uint32 * StateFlags,
            int32 msecs )
```

Wait for events on a single file handle.

Wait for a single OSAL filehandle to change state

This function can be used to wait for a single OSAL stream ID to become readable or writable. On entry, the "State↩ Flags" parameter should be set to the desired state (readble or writable) and upon return the flags will be set to the state actually detected.

As this operates on a single ID, the filehandle is protected during this call, such that another thread accessing the same handle will return an error. However, it is important to note that once the call returns then other threads may then also read/write and affect the state before the current thread can service it.

To mitigate this risk the application may prefer to use the OS_TimedRead/OS_TimedWrite calls.

Returns

Execution status, see OSAL Return Code Defines

10.16 OSAL Printf APIs

Functions

- void OS_printf (const char *string,...) OS_PRINTF(1

  Abstraction for the system printf() call.
- void void OS_printf_disable (void)

  This function disables the output from OS_printf.
- void OS_printf_enable (void)

  This function enables the output from OS_printf.

10.16.1 Detailed Description

10.16.2 Function Documentation

10.16.2.1 OS_printf()

```
void OS_printf (
            const char * string,
             ...  )
```

Abstraction for the system printf() call.

This function abstracts out the printf type statements. This is useful for using OS- specific thats that will allow non-polled print statements for the real time systems.

Operates in a manner similar to the printf() call defined by the standard C library and takes all the parameters and formatting options of printf. This abstraction may implement additional buffering, if necessary, to improve the real-time performance of the call.

The output of this routine also may be dynamically enabled or disabled by the OS_printf_enable() and OS_printf_↩ disable() calls, respectively.

Parameters

in	*string*	Format string, followed by additional arguments

10.16.2.2 OS_printf_disable()

```
void void OS_printf_disable (
            void  )
```

This function disables the output from OS_printf.

10.16.2.3 OS_printf_enable()

```
void OS_printf_enable (
             void  )
```

This function enables the output from OS_printf.

10.17 OSAL File Access Option Defines

Macros

- #define OS_READ_ONLY 0
- #define OS_WRITE_ONLY 1
- #define OS_READ_WRITE 2

10.17.1 Detailed Description

10.17.2 Macro Definition Documentation

10.17.2.1 OS_READ_ONLY

```
#define OS_READ_ONLY 0
```

Read only file access

Definition at line 25 of file osapi-os-filesys.h.

10.17.2.2 OS_READ_WRITE

```
#define OS_READ_WRITE 2
```

Read write file access

Definition at line 27 of file osapi-os-filesys.h.

10.17.2.3 OS_WRITE_ONLY

```
#define OS_WRITE_ONLY 1
```

Write only file access

Definition at line 26 of file osapi-os-filesys.h.

10.18 OSAL Refernce Point For Seek Offset Defines

Macros

- #define OS_SEEK_SET 0
- #define OS_SEEK_CUR 1
- #define OS_SEEK_END 2

10.18.1 Detailed Description

10.18.2 Macro Definition Documentation

10.18.2.1 OS_SEEK_CUR

```
#define OS_SEEK_CUR 1
```

Seek offset current

Definition at line 34 of file osapi-os-filesys.h.

10.18.2.2 OS_SEEK_END

```
#define OS_SEEK_END 2
```

Seek offset end

Definition at line 35 of file osapi-os-filesys.h.

10.18.2.3 OS_SEEK_SET

```
#define OS_SEEK_SET 0
```

Seek offset set

Definition at line 33 of file osapi-os-filesys.h.

10.19 OSAL Volume Type Defines

Macros

- #define FS_BASED 0
- #define RAM_DISK 1
- #define EEPROM_DISK 2
- #define ATA_DISK 3

10.19.1 Detailed Description

10.19.2 Macro Definition Documentation

10.19.2.1 ATA_DISK

```
#define ATA_DISK 3
```

Volume type ATA disk

Definition at line 47 of file osapi-os-filesys.h.

10.19.2.2 EEPROM_DISK

```
#define EEPROM_DISK 2
```

Volume type EEPROM disk

Definition at line 46 of file osapi-os-filesys.h.

10.19.2.3 FS_BASED

```
#define FS_BASED 0
```

Volume type FS based

Definition at line 44 of file osapi-os-filesys.h.

10.19.2.4 RAM_DISK

```
#define RAM_DISK 1
```

Volume type RAM disk

Definition at line 45 of file osapi-os-filesys.h.

10.20 OSAL Standard File APIs

Functions

- int32 OS_creat (const char *path, int32 access)

  Creates a file specified by path.
- int32 OS_open (const char *path, int32 access, uint32 mode)

  Opens a file.
- int32 OS_close (uint32 filedes)

  Closes an open file handle.
- int32 OS_read (uint32 filedes, void *buffer, uint32 nbytes)

  Read from a file handle.
- int32 OS_write (uint32 filedes, const void *buffer, uint32 nbytes)

  Write to a file handle.
- int32 OS_TimedRead (uint32 filedes, void *buffer, uint32 nbytes, int32 timeout)

  File/Stream input read with a timeout.
- int32 OS_TimedWrite (uint32 filedes, const void *buffer, uint32 nbytes, int32 timeout)

  File/Stream output write with a timeout.
- int32 OS_chmod (const char *path, uint32 access)

  Changes the permissions of a file.
- int32 OS_stat (const char *path, os_fstat_t *filestats)

  Obtain information about a file or directory.
- int32 OS_lseek (uint32 filedes, int32 offset, uint32 whence)

  Seeks to the specified position of an open file.
- int32 OS_remove (const char *path)

  Removes a file from the file system.
- int32 OS_rename (const char *old_filename, const char *new_filename)

  Renames a file.
- int32 OS_cp (const char *src, const char *dest)

  Copies a single file from src to dest.
- int32 OS_mv (const char *src, const char *dest)

  Move a single file from src to dest.
- int32 OS_FDGetInfo (uint32 filedes, OS_file_prop_t *fd_prop)

  Obtain information about an open file.
- int32 OS_FileOpenCheck (const char *Filename)

  Checks to see if a file is open.
- int32 OS_CloseAllFiles (void)

  Close all open files.
- int32 OS_CloseFileByName (const char *Filename)

  Close a file by filename.

10.20.1 Detailed Description

10.20.2 Function Documentation

10.20.2.1 OS_chmod()

```
int32 OS_chmod (
            const char * path,
            uint32 access )
```

Changes the permissions of a file.

Parameters

in	*path*	File to change
in	*access*	Desired access mode - see OSAL File Access Option Defines

Note

Some file systems do not implement permissions

Returns

Execution status, see OSAL Return Code Defines

10.20.2.2 OS_close()

```
int32 OS_close (
            uint32 filedes )
```

Closes an open file handle.

This closes regular file handles and any other file-like resource, such as network streams or pipes.

Parameters

in	*filedes*	The handle ID to operate on

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERROR	if file descriptor could not be closed
OS_ERR_INVALID_ID	if the file descriptor passed in is invalid

10.20.2.3 OS_CloseAllFiles()

```
int32 OS_CloseAllFiles (
            void  )
```

Close all open files.

Closes All open files that were opened through the OSAL

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERROR	if one or more file close returned an error

10.20.2.4 OS_CloseFileByName()

```
int32 OS_CloseFileByName (
            const char * Filename )
```

Close a file by filename.

Allows a file to be closed by name. This will only work if the name passed in is the same name used to open the file.

Parameters

in	*Filename*	The file to close

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_FS_ERR_PATH_INVALID	if the file is not found
OS_ERROR	if the file close returned an error

10.20.2.5 OS_cp()

```
int32 OS_cp (
            const char * src,
            const char * dest )
```

Copies a single file from src to dest.

Parameters

in	*src*	The source file to operate on
in	*dest*	The destination file

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERROR	if the file could not be accessed
OS_INVALID_POINTER	if src or dest are NULL
OS_FS_ERR_PATH_INVALID	if path cannot be parsed
OS_FS_ERR_PATH_TOO_LONG	if the paths given are too long to be stored locally
OS_FS_ERR_NAME_TOO_LONG	if the dest name is too long to be stored locally

10.20.2.6 OS_creat()

```
int32 OS_creat (
            const char * path,
            int32 access )
```

Creates a file specified by path.

Creates a file specified by const char ∗path, with read/write permissions by access. The file is also automatically opened by the create call.

Parameters

in	*path*	File name to create
in	*access*	Intended access mode - see OSAL File Access Option Defines

Note

Valid handle IDs are never negative. Failure of this call can be checked by testing if the result is less than 0.

Returns

A file handle ID or appropriate error code, see OSAL Return Code Defines

Return values

OS_INVALID_POINTER	if path is NULL
OS_FS_ERR_PATH_TOO_LONG	if path exceeds the maximum number of chars
OS_FS_ERR_PATH_INVALID	if path cannot be parsed
OS_FS_ERR_NAME_TOO_LONG	if the name of the file is too long
OS_ERROR	if permissions are unknown or OS call fails
OS_ERR_NO_FREE_IDS	if there are no free file descriptors left

10.20.2.7 OS_FDGetInfo()

```
int32 OS_FDGetInfo (
            uint32 filedes,
            OS_file_prop_t * fd_prop )
```

Obtain information about an open file.

Copies the information of the given file descriptor into a structure passed in

Parameters

in	*filedes*	The handle ID to operate on
out	*fd_prop*	Storage buffer for file information

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the file descriptor passed in is invalid

10.20.2.8 OS_FileOpenCheck()

```
int32 OS_FileOpenCheck (
            const char * Filename )
```

Checks to see if a file is open.

This function takes a filename and determines if the file is open. The function will return success if the file is open.

Parameters

in	*Filename*	The file to operate on

Returns

OS_SUCCESS if the file is open, or appropriate error code

Return values

OS_ERROR	if the file is not open

10.20.2.9 OS_lseek()

```
int32 OS_lseek (
            uint32 filedes,
            int32 offset,
            uint32 whence )
```

Seeks to the specified position of an open file.

Sets the read/write pointer to a specific offset in a specific file.

Parameters

in	*filedes*	The handle ID to operate on
in	*offset*	The file offset to seek to
in	*whence*	The reference point for offset, see OSAL Refernce Point For Seek Offset Defines

Returns

Byte offset from the beginning of the file or appropriate error code, see OSAL Return Code Defines

Return values

OS_ERR_INVALID_ID	if the file descriptor passed in is invalid
OS_ERROR	if OS call failed

10.20.2.10 OS_mv()

```
int32 OS_mv (
            const char * src,
            const char * dest )
```

Move a single file from src to dest.

This first attempts to rename the file, which is faster if the source and destination reside on the same file system.

If this fails, it falls back to copying the file and removing the original.

Parameters

in	*src*	The source file to operate on
in	*dest*	The destination file

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERROR	if the file could not be renamed.
OS_INVALID_POINTER	if src or dest are NULL
OS_FS_ERR_PATH_INVALID	if path cannot be parsed
OS_FS_ERR_PATH_TOO_LONG	if the paths given are too long to be stored locally
OS_FS_ERR_NAME_TOO_LONG	if the dest name is too long to be stored locally

10.20.2.11 OS_open()

```
int32 OS_open (
            const char * path,
            int32 access,
            uint32 mode )
```

Opens a file.

Opens a file.

Parameters

in	*path*	File name to create
in	*access*	Intended access mode - see OSAL File Access Option Defines
in	*mode*	The file permissions. This parameter is passed through to the native open call, but will be ignored. The file mode (or permissions) are ignored by the POSIX open call when the O_CREAT access flag is not passed in.

Note

Valid handle IDs are never negative. Failure of this call can be checked by testing if the result is less than 0.

Returns

A file handle ID or appropriate error code, see OSAL Return Code Defines

Return values

OS_INVALID_POINTER	if path is NULL
OS_FS_ERR_PATH_TOO_LONG	if path exceeds the maximum number of chars
OS_FS_ERR_PATH_INVALID	if path cannot be parsed
OS_FS_ERR_NAME_TOO_LONG	if the name of the file is too long
OS_ERROR	if permissions are unknown or OS call fails
OS_ERR_NO_FREE_IDS	if there are no free file descriptors left

10.20.2.12 OS_read()

```
int32 OS_read (
            uint32 filedes,
            void * buffer,
            uint32 nbytes )
```

Read from a file handle.

Reads up to nbytes from a file, and puts them into buffer.

Parameters

in	*filedes*	The handle ID to operate on
out	*buffer*	Storage location for file data
in	*nbytes*	Maximum number of bytes to read

Note

All OSAL error codes are negative int32 values. Failure of this call can be checked by testing if the result is less than 0.

Returns

A non-negative byte count or appropriate error code, see OSAL Return Code Defines

Return values

OS_INVALID_POINTER	if buffer is a null pointer
OS_ERROR	if OS call failed
OS_ERR_INVALID_ID	if the file descriptor passed in is invalid

10.20.2.13 OS_remove()

```
int32 OS_remove (
            const char * path )
```

Removes a file from the file system.

Removes a given filename from the drive

Note

Some file systems permit removal of open files while others do not. For portability, it is recommended that applications ensure the file is closed prior to removal.

Parameters

in	*path*	The file to operate on

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERROR	if there is no device or the driver returns error
OS_INVALID_POINTER	if path is NULL
OS_FS_ERR_PATH_TOO_LONG	if path is too long to be stored locally
OS_FS_ERR_PATH_INVALID	if path cannot be parsed
OS_FS_ERR_NAME_TOO_LONG	if the name of the file to remove is too long

10.20.2.14 OS_rename()

```
int32 OS_rename (
            const char * old_filename,
            const char * new_filename )
```

Renames a file.

Changes the name of a file, where the source and destination reside on the same file system.

Note

Some file systems permit renaming of open files while others do not. For portability, it is recommended that applications ensure the file is closed prior to rename.

Parameters

in	*old_filename*	The original filename
in	*new_filename*	The desired filename

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERROR	if the file could not be opened or renamed.
OS_INVALID_POINTER	if old or new are NULL
OS_FS_ERR_PATH_INVALID	if path cannot be parsed
OS_FS_ERR_PATH_TOO_LONG	if the paths given are too long to be stored locally
OS_FS_ERR_NAME_TOO_LONG	if the new name is too long to be stored locally

10.20.2.15 OS_stat()

```
int32 OS_stat (
            const char * path,
            os_fstat_t * filestats )
```

Obtain information about a file or directory.

Returns information about a file or directory in a os_fstat_t structure

Parameters

in	*path*	The file to operate on
out	*filestats*	Buffer to store file information

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if path or filestats is NULL
OS_FS_ERR_PATH_TOO_LONG	if the path is too long to be stored locally
OS_FS_ERR_NAME_TOO_LONG	if the name of the file is too long to be stored
OS_FS_ERR_PATH_INVALID	if path cannot be parsed
OS_ERROR	if the OS call failed

10.20.2.16 OS_TimedRead()

```
int32 OS_TimedRead (
            uint32 filedes,
            void * buffer,
            uint32 nbytes,
            int32 timeout )
```

File/Stream input read with a timeout.

This implements a time-limited read and is primarily intended for use with sockets but may also work with any other stream-like resource that the underlying OS supports.

If data is immediately available on the file/socket, this will return that data along with the actual number of bytes that were immediately available. It will not block.

If no data is immediately available, this will wait up to the given timeout for data to appear. If no data appears within the timeout period, then this returns an error code (not zero).

In all cases this will return successfully as soon as at least 1 byte of actual data is available. It will not attempt to read the entire input buffer.

If an EOF condition occurs prior to timeout, this function returns zero.

Parameters

in	*filedes*	The handle ID to operate on
in	*buffer*	Source location for file data
in	*nbytes*	Maximum number of bytes to read
in	*timeout*	Maximum time to wait, in milliseconds (OS_PEND = forever)

Returns

Byte count on success, zero for timeout, or appropriate error code, see OSAL Return Code Defines

10.20.2.17 OS_TimedWrite()

```
int32 OS_TimedWrite (
            uint32 filedes,
            const void * buffer,
            uint32 nbytes,
            int32 timeout )
```

File/Stream output write with a timeout.

This implements a time-limited write and is primarily intended for use with sockets but may also work with any other stream-like resource that the underlying OS supports.

If output buffer space is immediately available on the file/socket, this will place data into the buffer and return the actual number of bytes that were queued for output. It will not block.

If no output buffer space is immediately available, this will wait up to the given timeout for space to become available. If no space becomes available within the timeout period, then this returns an error code (not zero).

In all cases this will return successfully as soon as at least 1 byte of actual data is output. It will *not* attempt to write the entire output buffer.

If an EOF condition occurs prior to timeout, this function returns zero.

Parameters

in	*filedes*	The handle ID to operate on
in	*buffer*	Source location for file data
in	*nbytes*	Maximum number of bytes to read
in	*timeout*	Maximum time to wait, in milliseconds (OS_PEND = forever)

Returns

Byte count on success, zero for timeout, or appropriate error code, see OSAL Return Code Defines

10.20.2.18 OS_write()

```
int32 OS_write (
            uint32 filedes,
            const void * buffer,
            uint32 nbytes )
```

Write to a file handle.

Writes to a file. copies up to a maximum of nbytes of buffer to the file described in filedes

Parameters

in	*filedes*	The handle ID to operate on
in	*buffer*	Source location for file data
in	*nbytes*	Maximum number of bytes to read

Note

All OSAL error codes are negative int32 values. Failure of this call can be checked by testing if the result is less than 0.

Returns

A non-negative byte count or appropriate error code, see OSAL Return Code Defines

Return values

OS_INVALID_POINTER	if buffer is NULL
OS_ERROR	if OS call failed
OS_ERR_INVALID_ID	if the file descriptor passed in is invalid

10.21 OSAL Directory APIs

Functions

- os_dirp_t OS_opendir (const char *path)

  Opens a directory for searching.
- int32 OS_closedir (os_dirp_t directory)
- void OS_rewinddir (os_dirp_t directory)
- os_dirent_t * OS_readdir (os_dirp_t directory)
- int32 OS_DirectoryOpen (uint32 *dir_id, const char *path)

  Opens a directory.
- int32 OS_DirectoryClose (uint32 dir_id)

  Closes an open directory.
- int32 OS_DirectoryRewind (uint32 dir_id)

  Rewinds an open directory.
- int32 OS_DirectoryRead (uint32 dir_id, os_dirent_t *dirent)

  Reads the next name in the directory.
- int32 OS_mkdir (const char *path, uint32 access)

  Makes a new directory.
- int32 OS_rmdir (const char *path)

  Removes a directory from the file system.

10.21.1 Detailed Description

10.21.2 Function Documentation

10.21.2.1 OS_closedir()

```
int32 OS_closedir (
            os_dirp_t directory )
```

10.21.2.2 OS_DirectoryClose()

```
int32 OS_DirectoryClose (
            uint32 dir_id )
```

Closes an open directory.

The directory referred to by dir_id will be closed

Parameters

in	*dir↩ _id*	The handle ID of the directory

Returns

Execution status, see OSAL Return Code Defines

10.21.2.3 OS_DirectoryOpen()

```
int32 OS_DirectoryOpen (
            uint32 * dir_id,
            const char * path )
```

Opens a directory.

Prepares for reading the files within a directory

Parameters

out	*dir↩ _id*	The handle ID of the directory
in	*path*	The directory to open

Returns

Execution status, see OSAL Return Code Defines

10.21.2.4 OS_DirectoryRead()

```
int32 OS_DirectoryRead (
            uint32 dir_id,
            os_dirent_t * dirent )
```

Reads the next name in the directory.

Obtains directory entry data for the next file from an open directory

Parameters

in	*dir↩ _id*	The handle ID of the directory
out	*dirent*	Buffer to store directory entry information

Returns

Execution status, see OSAL Return Code Defines

10.21.2.5 OS_DirectoryRewind()

```
int32 OS_DirectoryRewind (
            uint32 dir_id )
```

Rewinds an open directory.

Resets a directory read handle back to the first file.

Parameters

in	*dir_id*	The handle ID of the directory

Returns

Execution status, see OSAL Return Code Defines

10.21.2.6 OS_mkdir()

```
int32 OS_mkdir (
            const char * path,
            uint32 access )
```

Makes a new directory.

Makes a directory specified by path.

Parameters

in	*path*	The new directory name
in	*access*	The permissions for the directory (reserved for future use)

Note

Current implementations do not utilize the "access" parameter. Applications should still pass the intended value (OS_READ_WRITE or OS_READ_ONLY) to be compatible with future implementations.

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if path is NULL
OS_FS_ERR_PATH_TOO_LONG	if the path is too long to be stored locally
OS_FS_ERR_PATH_INVALID	if path cannot be parsed
OS_ERROR	if the OS call fails

10.21.2.7 OS_opendir()

```
os_dirp_t OS_opendir (
              const char * path )
```

Opens a directory for searching.

Deprecated Replaced by OS_DirectoryOpen()

10.21.2.8 OS_readdir()

```
os_dirent_t* OS_readdir (
              os_dirp_t directory )
```

10.21.2.9 OS_rewinddir()

```
void OS_rewinddir (
              os_dirp_t directory )
```

10.21.2.10 OS_rmdir()

```
int32 OS_rmdir (
              const char * path )
```

Removes a directory from the file system.

Removes a directory from the structure. The directory must be empty prior to this operation.

Parameters

in	*path*	The directory to remove

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if path is NULL
OS_FS_ERR_PATH_INVALID	if path cannot be parsed
OS_FS_ERR_PATH_TOO_LONG	
OS_ERROR	if the directory remove operation failed

10.22 OSAL File System Level APIs

Functions

- int32 OS_FileSysAddFixedMap (uint32 *filesys_id, const char *phys_path, const char *virt_path)

  Create a fixed mapping between an existing directory and a virtual OSAL mount point.
- int32 OS_mkfs (char *address, const char *devname, const char *volname, uint32 blocksize, uint32 numblocks)

  Makes a file system on the target.
- int32 OS_mount (const char *devname, const char *mountpoint)

  Mounts a file system.
- int32 OS_initfs (char *address, const char *devname, const char *volname, uint32 blocksize, uint32 numblocks)

  Initializes an existing file system.
- int32 OS_rmfs (const char *devname)

  Removes a file system.
- int32 OS_unmount (const char *mountpoint)

  Unmounts a mounted file system.
- int32 OS_fsBlocksFree (const char *name)

  Obtain number of blocks free.
- int32 OS_fsBytesFree (const char *name, uint64 *bytes_free)

  Obtains the number of free bytes in a volume.
- int32 OS_chkfs (const char *name, bool repair)

  Checks the health of a file system and repairs it if necessary.
- int32 OS_FS_GetPhysDriveName (char *PhysDriveName, const char *MountPoint)

  Obtains the physical drive name associated with a mount point.
- int32 OS_TranslatePath (const char *VirtualPath, char *LocalPath)

  Translates a OSAL Virtual file system path to a host Local path.
- int32 OS_GetFsInfo (os_fsinfo_t *filesys_info)

  Returns information about the file system.

10.22.1 Detailed Description

10.22.2 Function Documentation

10.22.2.1 OS_chkfs()

```
int32 OS_chkfs (
            const char * name,
            bool repair )
```

Checks the health of a file system and repairs it if necessary.

Checks the drives for inconsistencies and optionally also repairs it

Note

not all operating systems implement this function

Parameters

in	*name*	The device/path to operate on
in	*repair*	Whether to also repair inconsistencies

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	Name is NULL
OS_ERR_NOT_IMPLEMENTED	Not implemented.
OS_ERROR	Failed execution.

10.22.2.2 OS_FileSysAddFixedMap()

```
int32 OS_FileSysAddFixedMap (
            uint32 * filesys_id,
            const char * phys_path,
            const char * virt_path )
```

Create a fixed mapping between an existing directory and a virtual OSAL mount point.

This mimics the behavior of a "FS_BASED" entry in the VolumeTable but is registered at runtime. It is intended to be called by the PSP/BSP prior to starting the OSAL.

Parameters

out	*filesys_id*	An OSAL ID reflecting the file system
in	*phys_path*	The native system directory (an existing mount point)
in	*virt_path*	The virtual mount point of this filesystem

Returns

Execution status, see OSAL Return Code Defines

10.22.2.3 OS_FS_GetPhysDriveName()

```
int32 OS_FS_GetPhysDriveName (
            char * PhysDriveName,
            const char * MountPoint )
```

Obtains the physical drive name associated with a mount point.

Returns the name of the physical volume associated with the drive, when given the OSAL mount point of the drive

Parameters

out	*PhysDriveName*	Buffer to store physical drive name
in	*MountPoint*	OSAL mount point

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if either parameter is NULL
OS_ERROR	if the mountpoint could not be found

10.22.2.4 OS_fsBlocksFree()

```
int32 OS_fsBlocksFree (
            const char * name )
```

Obtain number of blocks free.

Returns the number of free blocks in a volume

Parameters

in	*name*	The device/path to operate on

Returns

Block count or appropriate error code, see OSAL Return Code Defines

Return values

OS_INVALID_POINTER	if name is NULL
OS_FS_ERR_PATH_TOO_LONG	if the name is too long
OS_ERROR	if the OS call failed

10.22.2.5 OS_fsBytesFree()

```
int32 OS_fsBytesFree (
            const char * name,
            uint64 * bytes_free )
```

Obtains the number of free bytes in a volume.

Returns the number of free bytes in a volume

Note

uses a 64 bit data type to support filesystems that are greater than 4 Gigabytes

Parameters

in	*name*	The device/path to operate on
out	*bytes_free*	The number of free bytes

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if name is NULL
OS_FS_ERR_PATH_TOO_LONG	if the name is too long
OS_ERROR	if the OS call failed

10.22.2.6 OS_GetFsInfo()

```
int32 OS_GetFsInfo (
            os_fsinfo_t * filesys_info )
```

Returns information about the file system.

Returns information about the file system in an os_fsinfo_t. This includes the number of open files and file systems

Parameters

out	*filesys_info*	Buffer to store filesystem information

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if filesys_info is NULL

10.22.2.7 OS_initfs()

```
int32 OS_initfs (
            char * address,
            const char * devname,
            const char * volname,
            uint32 blocksize,
            uint32 numblocks )
```

Initializes an existing file system.

Initializes a file system on the target.

Parameters

in	*address*	The address at which to start the new disk. If address == 0, then space will be allocated by the OS
in	*devname*	The name of the "generic" drive
in	*volname*	The name of the volume (if needed, used on VxWorks)
in	*blocksize*	The size of a single block on the drive
in	*numblocks*	The number of blocks to allocate for the drive

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if devname or volname are NULL
OS_FS_ERR_PATH_TOO_LONG	if the name is too long
OS_FS_ERR_DEVICE_NOT_FREE	if the volume table is full
OS_FS_ERR_DRIVE_NOT_CREATED	on error

10.22.2.8 OS_mkfs()

```
int32 OS_mkfs (
            char * address,
            const char * devname,
            const char * volname,
            uint32 blocksize,
            uint32 numblocks )
```

Makes a file system on the target.

Makes a file system on the target. Highly dependent on underlying OS and dependent on OS volume table definition.

Parameters

in	*address*	The address at which to start the new disk. If address == 0 space will be allocated by the OS.
in	*devname*	The name of the "generic" drive
in	*volname*	The name of the volume (if needed, used on VxWorks)
in	*blocksize*	The size of a single block on the drive
in	*numblocks*	The number of blocks to allocate for the drive

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if devname is NULL
OS_FS_ERR_DRIVE_NOT_CREATED	if the OS calls to create the the drive failed
OS_FS_ERR_DEVICE_NOT_FREE	if the volume table is full
OS_FS_SUCCESS	on creating the disk

10.22.2.9 OS_mount()

```
int32 OS_mount (
            const char * devname,
            const char * mountpoint )
```

Mounts a file system.

Mounts a file system / block device at the given mount point.

Parameters

in	*devname*	The name of the drive to mount. devname is the same from OS_mkfs
in	*mountpoint*	The name to call this disk from now on

Returns

Execution status, see OSAL Return Code Defines

10.22.2.10 OS_rmfs()

```
int32 OS_rmfs (
            const char * devname )
```

Removes a file system.

This function will remove or un-map the target file system. Note that this is not the same as un-mounting the file system.

Parameters

in	*devname*	The name of the "generic" drive

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if devname is NULL
OS_ERROR	is the drive specified cannot be located

10.22.2.11 OS_TranslatePath()

```
int32 OS_TranslatePath (
            const char * VirtualPath,
            char * LocalPath )
```

Translates a OSAL Virtual file system path to a host Local path.

Translates a virtual path to an actual system path name

Parameters

in	*VirtualPath*	OSAL virtual path name
out	*LocalPath*	Buffer to store native/translated path name

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if either parameter is NULL

10.22.2.12 OS_unmount()

```
int32 OS_unmount (
            const char * mountpoint )
```

Unmounts a mounted file system.

This function will unmount a drive from the file system and make all open file descriptors useless.

Note

Any open file descriptors referencing this file system should be closed prior to unmounting a drive

Parameters

in	*mountpoint*	The mount point to remove from OS_mount

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if name is NULL
OS_FS_ERR_PATH_TOO_LONG	if the absolute path given is too long
OS_ERROR	if the OS calls failed

10.23 OSAL Shell APIs

Functions

- int32 OS_ShellOutputToFile (const char *Cmd, uint32 filedes)

  Executes the command and sends output to a file.

10.23.1 Detailed Description

10.23.2 Function Documentation

10.23.2.1 OS_ShellOutputToFile()

```
int32 OS_ShellOutputToFile (
            const char * Cmd,
            uint32 filedes )
```

Executes the command and sends output to a file.

Takes a shell command in and writes the output of that command to the specified file The output file must be opened previously with write access (OS_WRITE_ONLY or OS_READ_WRITE).

Parameters

in	*Cmd*	Command to pass to shell
in	*filedes*	File to send output to.

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERROR	if the command was not executed properly
OS_ERR_INVALID_ID	if the file descriptor passed in is invalid

10.24 OSAL Dynamic Loader and Symbol APIs

Functions

- int32 OS_SymbolLookup (cpuaddr *symbol_address, const char *symbol_name)

  Find the Address of a Symbol.
- int32 OS_SymbolTableDump (const char *filename, uint32 size_limit)

  Dumps the system symbol table to a file.
- int32 OS_ModuleLoad (uint32 *module_id, const char *module_name, const char *filename)

  Loads an object file.
- int32 OS_ModuleUnload (uint32 module_id)

  Unloads the module file.
- int32 OS_ModuleInfo (uint32 module_id, OS_module_prop_t *module_info)

  Obtain information about a module.

10.24.1 Detailed Description

10.24.2 Function Documentation

10.24.2.1 OS_ModuleInfo()

```
int32 OS_ModuleInfo (
            uint32 module_id,
            OS_module_prop_t * module_info )
```

Obtain information about a module.

Returns information about the loadable module

Parameters

in	*module_id*	OSAL ID of the previously the loaded module
out	*module_info*	Buffer to store module information

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the module id invalid
OS_INVALID_POINTER	if the pointer to the ModuleInfo structure is invalid

10.24.2.2 OS_ModuleLoad()

```
int32 OS_ModuleLoad (
            uint32 * module_id,
            const char * module_name,
            const char * filename )
```

Loads an object file.

Loads an object file into the running operating system

Parameters

out	*module_id*	OSAL ID corresponding to the loaded module
in	*module_name*	Name of module
in	*filename*	File containing the object code to load

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERROR	if the module cannot be loaded
OS_INVALID_POINTER	if one of the parameters is NULL
OS_ERR_NO_FREE_IDS	if the module table is full
OS_ERR_NAME_TAKEN	if the name is in use

10.24.2.3 OS_ModuleUnload()

```
int32 OS_ModuleUnload (
            uint32 module_id )
```

Unloads the module file.

Unloads the module file from the running operating system

Parameters

in	*module_id*	OSAL ID of the previously the loaded module

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERROR	if the module is invalid or cannot be unloaded

10.24.2.4 OS_SymbolLookup()

```
int32 OS_SymbolLookup (
            cpuaddr * symbol_address,
            const char * symbol_name )
```

Find the Address of a Symbol.

This calls to the OS dynamic symbol lookup implementation, and/or checks a static symbol table for a matching symbol name.

The static table is intended to support embedded targets that do not have module loading capability or have it disabled.

Parameters

out	*symbol_address*	Set to the address of the symbol
in	*symbol_name*	Name of the symbol to look up

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERROR	if the symbol could not be found
OS_INVALID_POINTER	if one of the pointers passed in are NULL

10.24.2.5 OS_SymbolTableDump()

```
int32 OS_SymbolTableDump (
            const char * filename,
            uint32 size_limit )
```

Dumps the system symbol table to a file.

Dumps the system symbol table to the specified filename

Parameters

in	*filename*	File to write to
in	*size_limit*	Maximum number of bytes to write

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_NOT_IMPLEMENTED	Not implemented.
OS_ERROR	if the symbol table could not be read or dumped

10.25 OSAL Socket Address APIs

Functions

- int32 OS_SocketAddrInit (OS_SockAddr_t *Addr, OS_SocketDomain_t Domain)

  Initialize a socket address structure to hold an address of the given family.
- int32 OS_SocketAddrToString (char *buffer, uint32 buflen, const OS_SockAddr_t *Addr)

  Get a string representation of a network host address.
- int32 OS_SocketAddrFromString (OS_SockAddr_t *Addr, const char *string)

  Set a network host address from a string representation.
- int32 OS_SocketAddrGetPort (uint16 *PortNum, const OS_SockAddr_t *Addr)

  Get the port number of a network address.
- int32 OS_SocketAddrSetPort (OS_SockAddr_t *Addr, uint16 PortNum)

  Set the port number of a network address.

10.25.1 Detailed Description

These functions provide a means to manipulate network addresses in a manner that is (mostly) agnostic to the actual network address type.

Every network address should be representable as a string (i.e. dotted decimal IP, etc). This can serve as a the "common denominator" to all address types.

10.25.2 Function Documentation

10.25.2.1 OS_SocketAddrFromString()

```
int32 OS_SocketAddrFromString (
            OS_SockAddr_t * Addr,
            const char * string )
```

Set a network host address from a string representation.

The specific format of the output string depends on the address family.

The address structure should have been previously initialized using OS_SocketAddrInit() to set the address family type.

Note

For IPv4, this would typically be the dotted-decimal format (X.X.X.X). It is up to the discretion of the underlying implementation whether to accept hostnames, as this depends on the availability of DNS services. Since many embedded deployments do not have name services, this should not be relied upon.

Parameters

out	*Addr*	The address buffer to initialize
in	*string*	The string to initialize the address from.

Returns

Execution status, see OSAL Return Code Defines

10.25.2.2 OS_SocketAddrGetPort()

```
int32 OS_SocketAddrGetPort (
            uint16 * PortNum,
            const OS_SockAddr_t * Addr )
```

Get the port number of a network address.

For network prototcols that have the concept of a port number (such as TCP/IP and UDP/IP) this function gets the port number from the address structure.

Parameters

out	*PortNum*	Buffer to store the port number
in	*Addr*	The network address buffer

Returns

Execution status, see OSAL Return Code Defines

10.25.2.3 OS_SocketAddrInit()

```
int32 OS_SocketAddrInit (
            OS_SockAddr_t * Addr,
            OS_SocketDomain_t Domain )
```

Initialize a socket address structure to hold an address of the given family.

The address is set to a suitable default value for the family.

Parameters

out	*Addr*	The address buffer to initialize
in	*Domain*	The address family

Returns

Execution status, see OSAL Return Code Defines

10.25.2.4 OS_SocketAddrSetPort()

```
int32 OS_SocketAddrSetPort (
            OS_SockAddr_t * Addr,
            uint16 PortNum )
```

Set the port number of a network address.

For network prototcols that have the concept of a port number (such as TCP/IP and UDP/IP) this function sets the port number from the address structure.

Parameters

in	*PortNum*	The port number to set
out	*Addr*	The network address buffer

Returns

Execution status, see OSAL Return Code Defines

10.25.2.5 OS_SocketAddrToString()

```
int32 OS_SocketAddrToString (
            char * buffer,
            uint32 buflen,
            const OS_SockAddr_t * Addr )
```

Get a string representation of a network host address.

The specific format of the output string depends on the address family.

This string should be suitable to pass back into OS_SocketAddrFromString() which should recreate the same network address, and it should also be meaningful to a user of printed or logged as a C string.

Note

For IPv4, this would typically be the dotted-decimal format (X.X.X.X).

Parameters

out	*buffer*	Buffer to hold the output string
in	*buflen*	Maximum length of the output string
in	*Addr*	The network address buffer to convert

Returns

Execution status, see OSAL Return Code Defines

10.26 OSAL Socket Management APIs

Functions

- int32 OS_SocketOpen (uint32 *sock_id, OS_SocketDomain_t Domain, OS_SocketType_t Type)
  Opens a socket.
- int32 OS_SocketBind (uint32 sock_id, const OS_SockAddr_t *Addr)
  Binds a socket to a given local address.
- int32 OS_SocketConnect (uint32 sock_id, const OS_SockAddr_t *Addr, int32 timeout)
  Connects a socket to a given remote address.
- int32 OS_SocketAccept (uint32 sock_id, uint32 *connsock_id, OS_SockAddr_t *Addr, int32 timeout)
  Waits for and accept the next incoming connection on the given socket.
- int32 OS_SocketRecvFrom (uint32 sock_id, void *buffer, uint32 buflen, OS_SockAddr_t *RemoteAddr, int32 timeout)
  Reads data from a message-oriented (datagram) socket.
- int32 OS_SocketSendTo (uint32 sock_id, const void *buffer, uint32 buflen, const OS_SockAddr_t *RemoteAddr)
  Sends data to a message-oriented (datagram) socket.
- int32 OS_SocketGetIdByName (uint32 *sock_id, const char *sock_name)
  Gets an OSAL ID from a given name.
- int32 OS_SocketGetInfo (uint32 sock_id, OS_socket_prop_t *sock_prop)
  Gets information about an OSAL Socket ID.
- int32 OS_NetworkGetID (void)
  Gets the network ID of the local machine.
- int32 OS_NetworkGetHostName (char *host_name, uint32 name_len)
  Gets the local machine network host name.

10.26.1 Detailed Description

These functions are loosely related to the BSD Sockets API but made to be more consistent with other OSAL API functions. That is, they operate on OSAL IDs (32-bit opaque number values) and return an OSAL error code.

OSAL Socket IDs are very closely related to File IDs and share the same ID number space. Additionally, the file OS_read() / OS_write() / OS_close() calls also work on sockets.

Note that all of functions may return OS_ERR_NOT_IMPLEMENTED if network support is not configured at compile time.

10.26.2 Function Documentation

10.26.2.1 OS_NetworkGetHostName()

```
int32 OS_NetworkGetHostName (
            char * host_name,
            uint32 name_len )
```

Gets the local machine network host name.

If configured in the underlying network stack, this function retrieves the local hostname of the system.

Parameters

out	*host_name*	Buffer to hold name information
in	*name_len*	Maximum length of host name buffer

Returns

Execution status, see OSAL Return Code Defines

10.26.2.2 OS_NetworkGetID()

```
int32 OS_NetworkGetID (
            void  )
```

Gets the network ID of the local machine.

The ID is an implementation-defined value and may not be consistent in meaning across different platform types.

Note

This API may be removed in a future version of OSAL due to inconsistencies between platforms.

Returns

The ID or fixed value of -1 if the host id could not be found. Note it is not possible to differentiate between error codes and valid network IDs here. It is assumed, however, that -1 is never a valid ID.

10.26.2.3 OS_SocketAccept()

```
int32 OS_SocketAccept (
            uint32 sock_id,
            uint32 * connsock_id,
            OS_SockAddr_t * Addr,
            int32 timeout )
```

Waits for and accept the next incoming connection on the given socket.

This is used for sockets operating in a "server" role. The socket must be a stream type (connection-oriented) and previously bound to a local address using OS_SocketBind(). This will block the caller up to the given timeout or until an incoming connection request occurs, whichever happens first.

The new stream connection is then returned to the caller and the original server socket ID can be reused for the next connection.

Parameters

in	*sock_id*	The server socket ID, previously bound using OS_SocketBind()
out	*connsock↩ _id*	The connection socket, a new ID that can be read/written
in	*Addr*	The remote address of the incoming connection
in	*timeout*	The maximum amount of time to wait, or OS_PEND to wait forever

Returns

Execution status, see OSAL Return Code Defines

10.26.2.4 OS_SocketBind()

```
int32 OS_SocketBind (
            uint32 sock_id,
            const OS_SockAddr_t * Addr )
```

Binds a socket to a given local address.

The specified socket will be bound to the local address and port, if available.

If the socket is connectionless, then it only binds to the local address.

If the socket is connection-oriented (stream), then this will also put the socket into a listening state for incoming connections at the local address.

Parameters

in	*sock↩ _id*	The socket ID
in	*Addr*	The local address to bind to

Returns

Execution status, see OSAL Return Code Defines

10.26.2.5 OS_SocketConnect()

```
int32 OS_SocketConnect (
            uint32 sock_id,
            const OS_SockAddr_t * Addr,
            int32 timeout )
```

Connects a socket to a given remote address.

The socket will be connected to the remote address and port, if available. This only applies to stream-oriented sockets. Calling this on a datagram socket will return an error (these sockets should use SendTo/RecvFrom).

Parameters

in	*sock_id*	The socket ID
in	*Addr*	The remote address to connect to
in	*timeout*	The maximum amount of time to wait, or OS_PEND to wait forever

Returns

Execution status, see OSAL Return Code Defines

10.26.2.6 OS_SocketGetIdByName()

```
int32 OS_SocketGetIdByName (
            uint32 * sock_id,
            const char * sock_name )
```

Gets an OSAL ID from a given name.

Note

OSAL Sockets use generated names according to the address and type.

See also

OS_SocketGetInfo()

Parameters

out	*sock_id*	Buffer to hold result
in	*sock_name*	Name of socket to find

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	is id or name are NULL pointers
OS_ERR_NAME_TOO_LONG	if the name given is to long to have been stored
OS_ERR_NAME_NOT_FOUND	if the name was not found in the table

10.26.2.7 OS_SocketGetInfo()

```
int32 OS_SocketGetInfo (
            uint32 sock_id,
            OS_socket_prop_t * sock_prop )
```

Gets information about an OSAL Socket ID.

OSAL Sockets use generated names according to the address and type. This allows applications to find the name of a given socket.

Parameters

in	*sock_id*	The socket ID
out	*sock_prop*	Buffer to hold socket information

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the id passed in is not a valid semaphore
OS_INVALID_POINTER	if the count_prop pointer is null

10.26.2.8 OS_SocketOpen()

```
int32 OS_SocketOpen (
            uint32 * sock_id,
            OS_SocketDomain_t Domain,
            OS_SocketType_t Type )
```

Opens a socket.

A new, unconnected and unbound socket is allocated of the given domain and type.

Parameters

out	*sock_id*	Buffer to hold the OSAL ID
in	*Domain*	The domain / address family of the socket (INET or INET6, etc)
in	*Type*	The type of the socket (STREAM or DATAGRAM)

Returns

Execution status, see OSAL Return Code Defines

10.26.2.9 OS_SocketRecvFrom()

```
int32 OS_SocketRecvFrom (
            uint32 sock_id,
            void * buffer,
            uint32 buflen,
            OS_SockAddr_t * RemoteAddr,
            int32 timeout )
```

Reads data from a message-oriented (datagram) socket.

If a message is already available on the socket, this should immediately return that data without blocking. Otherwise, it may block up to the given timeout.

Parameters

in	*sock_id*	The socket ID, previously bound using OS_SocketBind()
out	*buffer*	Pointer to message data receive buffer
in	*buflen*	The maximum length of the message data to receive
out	*RemoteAddr*	Buffer to store the remote network address (may be NULL)
in	*timeout*	The maximum amount of time to wait, or OS_PEND to wait forever

Returns

Count of actual bytes received or error status, see OSAL Return Code Defines

10.26.2.10 OS_SocketSendTo()

```
int32 OS_SocketSendTo (
            uint32 sock_id,
            const void * buffer,
            uint32 buflen,
            const OS_SockAddr_t * RemoteAddr )
```

Sends data to a message-oriented (datagram) socket.

This sends data in a non-blocking mode. If the socket is not currently able to queue the message, such as if its outbound buffer is full, then this returns an error code.

Parameters

in	*sock_id*	The socket ID, which must be of the datagram type
in	*buffer*	Pointer to message data to send
in	*buflen*	The length of the message data to send
in	*RemoteAddr*	Buffer containing the remote network address to send to

Returns

Count of actual bytes sent or error status, see OSAL Return Code Defines

10.27 OSAL Timer APIs

Functions

- int32 OS_TimeBaseCreate (uint32 *timebase_id, const char *timebase_name, OS_TimerSync_t external_sync)

  Create an abstract Time Base resource.

- int32 OS_TimeBaseSet (uint32 timebase_id, uint32 start_time, uint32 interval_time)

  Sets the tick period for simulated time base objects.

- int32 OS_TimeBaseDelete (uint32 timebase_id)

  Deletes a time base object.

- int32 OS_TimeBaseGetIdByName (uint32 *timebase_id, const char *timebase_name)

  Find the ID of an existing time base resource.

- int32 OS_TimeBaseGetInfo (uint32 timebase_id, OS_timebase_prop_t *timebase_prop)

  Obtain information about a timebase resource.

- int32 OS_TimeBaseGetFreeRun (uint32 timebase_id, uint32 *freerun_val)

  Read the value of the timebase free run counter.

- int32 OS_TimerCreate (uint32 *timer_id, const char *timer_name, uint32 *clock_accuracy, OS_TimerCallback_t callback_ptr)

  Create a timer object.

- int32 OS_TimerAdd (uint32 *timer_id, const char *timer_name, uint32 timebase_id, OS_ArgCallback_t callback_ptr, void *callback_arg)

  Add a timer object based on an existing TimeBase resource.

- int32 OS_TimerSet (uint32 timer_id, uint32 start_time, uint32 interval_time)

  Configures a periodic or one shot timer.

- int32 OS_TimerDelete (uint32 timer_id)

  Deletes a timer resource.

- int32 OS_TimerGetIdByName (uint32 *timer_id, const char *timer_name)

  Locate an existing timer resource by name.

- int32 OS_TimerGetInfo (uint32 timer_id, OS_timer_prop_t *timer_prop)

  Gets information about an existing timer.

10.27.1 Detailed Description

10.27.2 Function Documentation

10.27.2.1 OS_TimeBaseCreate()

```
int32 OS_TimeBaseCreate (
            uint32 * timebase_id,
            const char * timebase_name,
            OS_TimerSync_t external_sync )
```

Create an abstract Time Base resource.

An OSAL time base is an abstraction of a "timer tick" that can, in turn, be used for measurement of elapsed time between events.

Time bases can be simulated by the operating system using the OS kernel-provided timing facilities, or based on a hardware timing source if provided by the BSP.

A time base object has a servicing task associated with it, that runs at elevated priority and will thereby interrupt user-level tasks when timing ticks occur.

If the external_sync function is passed as NULL, the operating system kernel timing resources will be utilized for a simulated timer tick.

If the external_sync function is not NULL, this should point to a BSP-provided function that will block the calling task until the next tick occurs. This can be used for synchronizing with hardware events.

Note

When provisioning a tunable RTOS kernel, such as RTEMS, the kernel should be configured to support at least (OS_MAX_TASKS + OS_MAX_TIMEBASES) threads, to account for the helper threads associated with time base objects.

Parameters

out	*timebase_id*	An identifier corresponding to the timebase resource
in	*timebase_name*	The name of the time base
in	*external_sync*	A synchronization function for BSP hardware-based timer ticks

Returns

Execution status, see OSAL Return Code Defines

10.27.2.2 OS_TimeBaseDelete()

```
int32 OS_TimeBaseDelete (
            uint32 timebase_id )
```

Deletes a time base object.

The helper task and any other resources associated with the time base abstraction will be freed.

Parameters

in	*timebase_id*	The timebase resource to delete

Returns

Execution status, see OSAL Return Code Defines

10.27.2.3 OS_TimeBaseGetFreeRun()

```
int32 OS_TimeBaseGetFreeRun (
            uint32 timebase_id,
            uint32 * freerun_val )
```

Read the value of the timebase free run counter.

Poll the timer free-running time counter in a lightweight fashion.

The free run count is a monotonically increasing value reflecting the total time elapsed since the timebase inception. Units are the same as the timebase itself, usually microseconds.

Applications may quickly and efficiently calculate relative time differences by polling this value and subtracting the previous counter value.

The absolute value of this counter is not relevant, because it will "roll over" after 2^32 units of time. For a timebase with microsecond units, this occurs approximately every 4294 seconds, or about 1.2 hours.

Note

To ensure consistency of results, the application should sample the value at a minimum of two times the roll over frequency, and calculate the difference between the consecutive samples.

Parameters

in	*timebase_id*	The timebase to operate on
out	*freerun_val*	Buffer to store the free run counter

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the id passed in is not a valid timebase

10.27.2.4 OS_TimeBaseGetIdByName()

```
int32 OS_TimeBaseGetIdByName (
            uint32 * timebase_id,
            const char * timebase_name )
```

Find the ID of an existing time base resource.

Given a time base name, find and output the ID associated with it.

Parameters

out	*timebase_id*	The timebase resource ID
in	*timebase_name*	The name of the timebase resource to find

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if timebase_id or timebase_name are NULL pointers
OS_ERR_NAME_TOO_LONG	if the name given is to long to have been stored
OS_ERR_NAME_NOT_FOUND	if the name was not found in the table

10.27.2.5 OS_TimeBaseGetInfo()

```
int32 OS_TimeBaseGetInfo (
            uint32 timebase_id,
            OS_timebase_prop_t * timebase_prop )
```

Obtain information about a timebase resource.

Fills the buffer referred to by the timebase_prop parameter with relevant information about the time base resource.

This function will pass back a pointer to structure that contains all of the relevant info(name and creator) about the specified timebase.

Parameters

in	*timebase_id*	The timebase resource ID
out	*timebase_prop*	Buffer to store timebase properties

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the id passed in is not a valid timebase
OS_INVALID_POINTER	if the timebase_prop pointer is null

10.27.2.6 OS_TimeBaseSet()

```
int32 OS_TimeBaseSet (
            uint32 timebase_id,
            uint32 start_time,
            uint32 interval_time )
```

Sets the tick period for simulated time base objects.

This sets the actual tick period for timing ticks that are simulated by the RTOS kernel (i.e. the "external_sync" parameter on the call to OS_TimeBaseCreate() is NULL).

The RTOS will be configured to wake up the helper thread at the requested interval.

This function has no effect for time bases that are using a BSP-provided external_sync function.

Parameters

in	*timebase_id*	The timebase resource to configure
in	*start_time*	The amount of delay for the first tick, in microseconds.
in	*interval_time*	The amount of delay between ticks, in microseconds.

Returns

Execution status, see OSAL Return Code Defines

10.27.2.7 OS_TimerAdd()

```
int32 OS_TimerAdd (
            uint32 * timer_id,
            const char * timer_name,
            uint32 timebase_id,
            OS_ArgCallback_t callback_ptr,
            void * callback_arg )
```

Add a timer object based on an existing TimeBase resource.

A timer object is a resource that invokes the specified application-provided function upon timer expiration. Timers may be one-shot or periodic in nature.

This function uses an existing time base object to service this timer, which must exist prior to adding the timer. The precision of the timer is the same as that of the underlying time base object. Multiple timer objects can be created referring to a single time base object.

This routine also uses a different callback function prototype from OS_TimerCreate(), allowing a single opaque argument to be passed to the callback routine. The OSAL implementation does not use this parameter, and may be set NULL.

Warning

Depending on the OS, the callback_ptr function may be similar to an interrupt service routine. Calls that cause the code to block or require an application context (like sending events) are generally not supported.

Parameters

out	*timer_id*	The resource ID of the timer object
in	*timer_name*	Name of the timer object
in	*timebase_id*	The time base resource to use as a reference
in	*callback_ptr*	Application-provided function to invoke
in	*callback_arg*	Opaque argument to pass to callback function

Returns

Execution status, see OSAL Return Code Defines

10.27.2.8 OS_TimerCreate()

```
int32 OS_TimerCreate (
            uint32 * timer_id,
            const char * timer_name,
            uint32 * clock_accuracy,
            OS_TimerCallback_t callback_ptr )
```

Create a timer object.

A timer object is a resource that invokes the specified application-provided function upon timer expiration. Timers may be one-shot or periodic in nature.

This function creates a dedicated (hidden) time base object to service this timer, which is created and deleted with the timer object itself. The internal time base is configured for an OS simulated timer tick at the same interval as the timer.

Note

clock_accuracy comes from the underlying OS tick value. The nearest integer microsecond value is returned, so may not be exact.

Warning

Depending on the OS, the callback_ptr function may be similar to an interrupt service routine. Calls that cause the code to block or require an application context (like sending events) are generally not supported.

Parameters

`out`	*timer_id*	The resource ID of the timer object
`in`	*timer_name*	Name of the timer object
`out`	*clock_accuracy*	Expected precision of the timer, in microseconds. This is the underlying tick value rounded to the nearest microsecond integer.
`in`	*callback_ptr*	The function pointer of the timer callback or ISR that will be called by the timer. The user's function is declared as follows: `void timer_callback(uint32 timer_id)` Where the timer_id is passed in to the function by the OSAL

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if any parameters are NULL
OS_ERR_NAME_TOO_LONG	if the name parameter is too long.
OS_ERR_NAME_TAKEN	if the name is already in use by another timer.
OS_ERR_NO_FREE_IDS	if all of the timers are already allocated.
OS_TIMER_ERR_INVALID_ARGS	if the callback pointer is zero.
OS_TIMER_ERR_UNAVAILABLE	if the timer cannot be created.

10.27.2.9 OS_TimerDelete()

```
int32 OS_TimerDelete (
            uint32 timer_id )
```

Deletes a timer resource.

The application callback associated with the timer will be stopped, and the resources freed for future use.

Parameters

in	*timer↩ _id*	The timer ID to operate on

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the timer_id is invalid.
OS_TIMER_ERR_INTERNAL	if there was a problem deleting the timer in the host OS.

10.27.2.10 OS_TimerGetIdByName()

```
int32 OS_TimerGetIdByName (
            uint32 * timer_id,
            const char * timer_name )
```

Locate an existing timer resource by name.

Outputs the ID associated with the given timer, if it exists.

Parameters

out	*timer_id*	The timer ID corresponding to the name
in	*timer_name*	The timer name to find

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if timer_id or timer_name are NULL pointers
OS_ERR_NAME_TOO_LONG	if the name given is to long to have been stored
OS_ERR_NAME_NOT_FOUND	if the name was not found in the table

10.27.2.11 OS_TimerGetInfo()

```
int32 OS_TimerGetInfo (
            uint32 timer_id,
            OS_timer_prop_t * timer_prop )
```

Gets information about an existing timer.

This function takes timer_id, and looks it up in the OS table. It puts all of the information known about that timer into a structure pointer to by timer_prop.

Parameters

in	*timer_id*	The timer ID to operate on
out	*timer_prop*	Buffer containing timer properties • creator: the OS task ID of the task that created this timer • name: the string name of the timer • start_time: the start time in microseconds, if any • interval_time: the interval time in microseconds, if any • accuracy: the accuracy of the timer in microseconds

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the id passed in is not a valid timer
OS_INVALID_POINTER	if the timer_prop pointer is null

10.27.2.12 OS_TimerSet()

```
int32 OS_TimerSet (
            uint32 timer_id,
            uint32 start_time,
            uint32 interval_time )
```

Configures a periodic or one shot timer.

This function programs the timer with a start time and an optional interval time. The start time is the time in microseconds when the user callback function will be called. If the interval time is non-zero, the timer will be reprogrammed with that interval in microseconds to call the user callback function periodically. If the start time and interval time are zero, the function will return an error.

For a "one-shot" timer, the start_time configures the expiration time, and the interval_time should be passed as zero to indicate the timer is not to be automatically reset.

Note

The resolution of the times specified is limited to the clock accuracy returned in the OS_TimerCreate call. If the times specified in the start_msec or interval_msec parameters are less than the accuracy, they will be rounded up to the accuracy of the timer.

Parameters

in	*timer_id*	The timer ID to operate on
in	*start_time*	Time in microseconds to the first expiration
in	*interval_time*	Time in microseconds between subsequent intervals, value of zero will only call the user callback function once after the start_msec time.

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the timer_id is not valid.
OS_TIMER_ERR_INTERNAL	if there was an error programming the OS timer.
OS_ERROR	if both start time and interval time are zero.

10.28 OSAL Return Code Defines

Macros

- #define OS_FS_ERR_PATH_TOO_LONG (-103)

  FS path too long.
- #define OS_FS_ERR_NAME_TOO_LONG (-104)

  FS name too long.
- #define OS_FS_ERR_DRIVE_NOT_CREATED (-106)

  FS drive not created.
- #define OS_FS_ERR_DEVICE_NOT_FREE (-107)

  FS device not free.
- #define OS_FS_ERR_PATH_INVALID (-108)

  FS path invalid.
- #define OS_FS_SUCCESS OS_SUCCESS

  Successful execution.
- #define OS_FS_ERROR OS_ERROR

  Failed execution.
- #define OS_FS_ERR_INVALID_POINTER OS_INVALID_POINTER

  Invalid pointer.
- #define OS_FS_ERR_NO_FREE_FDS OS_ERR_NO_FREE_IDS

  No free IDs.
- #define OS_FS_ERR_INVALID_FD OS_ERR_INVALID_ID

  Invalid ID.
- #define OS_FS_UNIMPLEMENTED OS_ERR_NOT_IMPLEMENTED

  Not implemented.
- #define OS_SUCCESS (0)

  Successful execution.
- #define OS_ERROR (-1)

  Failed execution.
- #define OS_INVALID_POINTER (-2)

  Invalid pointer.
- #define OS_ERROR_ADDRESS_MISALIGNED (-3)

  Address misalignment.
- #define OS_ERROR_TIMEOUT (-4)

  Error timeout.
- #define OS_INVALID_INT_NUM (-5)

  Invalid Interrupt number.
- #define OS_SEM_FAILURE (-6)

  Semaphore failure.
- #define OS_SEM_TIMEOUT (-7)

  Semaphore timeout.
- #define OS_QUEUE_EMPTY (-8)

  Queue empty.
- #define OS_QUEUE_FULL (-9)

  Queue full.
- #define OS_QUEUE_TIMEOUT (-10)

Queue timeout.

- #define OS_QUEUE_INVALID_SIZE (-11)

  Queue invalid size.

- #define OS_QUEUE_ID_ERROR (-12)

  Queue ID error.

- #define OS_ERR_NAME_TOO_LONG (-13)

  Name too long.

- #define OS_ERR_NO_FREE_IDS (-14)

  No free IDs.

- #define OS_ERR_NAME_TAKEN (-15)

  Name taken.

- #define OS_ERR_INVALID_ID (-16)

  Invalid ID.

- #define OS_ERR_NAME_NOT_FOUND (-17)

  Name not found.

- #define OS_ERR_SEM_NOT_FULL (-18)

  Semaphore not full.

- #define OS_ERR_INVALID_PRIORITY (-19)

  Invalid priority.

- #define OS_INVALID_SEM_VALUE (-20)

  Invalid semaphore value.

- #define OS_ERR_FILE (-27)

  File error.

- #define OS_ERR_NOT_IMPLEMENTED (-28)

  Not implemented.

- #define OS_TIMER_ERR_INVALID_ARGS (-29)

  Timer invalid arguments.

- #define OS_TIMER_ERR_TIMER_ID (-30)

  Timer ID error.

- #define OS_TIMER_ERR_UNAVAILABLE (-31)

  Timer unavailable.

- #define OS_TIMER_ERR_INTERNAL (-32)

  Timer internal error.

- #define OS_ERR_OBJECT_IN_USE (-33)

  Object in use.

- #define OS_ERR_BAD_ADDRESS (-34)

  Bad address.

- #define OS_ERR_INCORRECT_OBJ_STATE (-35)

  Incorrect object state.

- #define OS_ERR_INCORRECT_OBJ_TYPE (-36)

  Incorrect object type.

- #define OS_ERR_STREAM_DISCONNECTED (-37)

  Stream disconnected.

10.28.1 Detailed Description

10.28.2 Macro Definition Documentation

10.28.2.1 OS_ERR_BAD_ADDRESS

```
#define OS_ERR_BAD_ADDRESS (-34)
```

Bad address.

Definition at line 76 of file osapi.h.

10.28.2.2 OS_ERR_FILE

```
#define OS_ERR_FILE (-27)
```

File error.

Definition at line 69 of file osapi.h.

10.28.2.3 OS_ERR_INCORRECT_OBJ_STATE

```
#define OS_ERR_INCORRECT_OBJ_STATE (-35)
```

Incorrect object state.

Definition at line 77 of file osapi.h.

10.28.2.4 OS_ERR_INCORRECT_OBJ_TYPE

```
#define OS_ERR_INCORRECT_OBJ_TYPE (-36)
```

Incorrect object type.

Definition at line 78 of file osapi.h.

10.28.2.5 OS_ERR_INVALID_ID

```
#define OS_ERR_INVALID_ID (-16)
```

Invalid ID.

Definition at line 64 of file osapi.h.

10.28.2.6 OS_ERR_INVALID_PRIORITY

```
#define OS_ERR_INVALID_PRIORITY (-19)
```

Invalid priority.

Definition at line 67 of file osapi.h.

10.28.2.7 OS_ERR_NAME_NOT_FOUND

```
#define OS_ERR_NAME_NOT_FOUND (-17)
```

Name not found.

Definition at line 65 of file osapi.h.

10.28.2.8 OS_ERR_NAME_TAKEN

```
#define OS_ERR_NAME_TAKEN (-15)
```

Name taken.

Definition at line 63 of file osapi.h.

10.28.2.9 OS_ERR_NAME_TOO_LONG

```
#define OS_ERR_NAME_TOO_LONG (-13)
```

Name too long.

Definition at line 61 of file osapi.h.

10.28.2.10 OS_ERR_NO_FREE_IDS

```
#define OS_ERR_NO_FREE_IDS (-14)
```

No free IDs.

Definition at line 62 of file osapi.h.

10.28.2.11 OS_ERR_NOT_IMPLEMENTED

```
#define OS_ERR_NOT_IMPLEMENTED (-28)
```

Not implemented.

Definition at line 70 of file osapi.h.

10.28.2.12 OS_ERR_OBJECT_IN_USE

```
#define OS_ERR_OBJECT_IN_USE (-33)
```

Object in use.

Definition at line 75 of file osapi.h.

10.28.2.13 OS_ERR_SEM_NOT_FULL

```
#define OS_ERR_SEM_NOT_FULL (-18)
```

Semaphore not full.

Definition at line 66 of file osapi.h.

10.28.2.14 OS_ERR_STREAM_DISCONNECTED

```
#define OS_ERR_STREAM_DISCONNECTED (-37)
```

Stream disconnected.

Definition at line 79 of file osapi.h.

10.28.2.15 OS_ERROR

```
#define OS_ERROR (-1)
```

Failed execution.

Definition at line 49 of file osapi.h.

10.28.2.16 OS_ERROR_ADDRESS_MISALIGNED

```
#define OS_ERROR_ADDRESS_MISALIGNED (-3)
```

Address misalignment.

Definition at line 51 of file osapi.h.

10.28.2.17 OS_ERROR_TIMEOUT

```
#define OS_ERROR_TIMEOUT (-4)
```

Error timeout.

Definition at line 52 of file osapi.h.

10.28.2.18 OS_FS_ERR_DEVICE_NOT_FREE

```
#define OS_FS_ERR_DEVICE_NOT_FREE (-107)
```

FS device not free.

Definition at line 76 of file osapi-os-filesys.h.

10.28.2.19 OS_FS_ERR_DRIVE_NOT_CREATED

```
#define OS_FS_ERR_DRIVE_NOT_CREATED (-106)
```

FS drive not created.

Definition at line 75 of file osapi-os-filesys.h.

10.28.2.20 OS_FS_ERR_INVALID_FD

```
#define OS_FS_ERR_INVALID_FD OS_ERR_INVALID_ID
```

Invalid ID.

Definition at line 88 of file osapi-os-filesys.h.

10.28.2.21 OS_FS_ERR_INVALID_POINTER

```
#define OS_FS_ERR_INVALID_POINTER OS_INVALID_POINTER
```

Invalid pointer.

Definition at line 86 of file osapi-os-filesys.h.

10.28.2.22 OS_FS_ERR_NAME_TOO_LONG

```
#define OS_FS_ERR_NAME_TOO_LONG (-104)
```

FS name too long.

Definition at line 74 of file osapi-os-filesys.h.

10.28.2.23 OS_FS_ERR_NO_FREE_FDS

```
#define OS_FS_ERR_NO_FREE_FDS OS_ERR_NO_FREE_IDS
```

No free IDs.

Definition at line 87 of file osapi-os-filesys.h.

10.28.2.24 OS_FS_ERR_PATH_INVALID

```
#define OS_FS_ERR_PATH_INVALID (-108)
```

FS path invalid.

Definition at line 77 of file osapi-os-filesys.h.

10.28.2.25 OS_FS_ERR_PATH_TOO_LONG

```
#define OS_FS_ERR_PATH_TOO_LONG (-103)
```

FS path too long.

Definition at line 73 of file osapi-os-filesys.h.

10.28.2.26 OS_FS_ERROR

```
#define OS_FS_ERROR OS_ERROR
```

Failed execution.

Definition at line 85 of file osapi-os-filesys.h.

10.28.2.27 OS_FS_SUCCESS

```
#define OS_FS_SUCCESS OS_SUCCESS
```

Successful execution.

Definition at line 84 of file osapi-os-filesys.h.

10.28.2.28 OS_FS_UNIMPLEMENTED

```
#define OS_FS_UNIMPLEMENTED OS_ERR_NOT_IMPLEMENTED
```

Not implemented.

Definition at line 89 of file osapi-os-filesys.h.

10.28.2.29 OS_INVALID_INT_NUM

```
#define OS_INVALID_INT_NUM (-5)
```

Invalid Interrupt number.

Definition at line 53 of file osapi.h.

10.28.2.30 OS_INVALID_POINTER

```
#define OS_INVALID_POINTER (-2)
```

Invalid pointer.

Definition at line 50 of file osapi.h.

10.28.2.31 OS_INVALID_SEM_VALUE

```
#define OS_INVALID_SEM_VALUE (-20)
```

Invalid semaphore value.

Definition at line 68 of file osapi.h.

10.28.2.32 OS_QUEUE_EMPTY

```
#define OS_QUEUE_EMPTY (-8)
```

Queue empty.

Definition at line 56 of file osapi.h.

10.28.2.33 OS_QUEUE_FULL

```
#define OS_QUEUE_FULL (-9)
```

Queue full.

Definition at line 57 of file osapi.h.

10.28.2.34 OS_QUEUE_ID_ERROR

```
#define OS_QUEUE_ID_ERROR (-12)
```

Queue ID error.

Definition at line 60 of file osapi.h.

10.28.2.35 OS_QUEUE_INVALID_SIZE

```
#define OS_QUEUE_INVALID_SIZE (-11)
```

Queue invalid size.

Definition at line 59 of file osapi.h.

10.28.2.36 OS_QUEUE_TIMEOUT

```
#define OS_QUEUE_TIMEOUT (-10)
```

Queue timeout.

Definition at line 58 of file osapi.h.

10.28.2.37 OS_SEM_FAILURE

```
#define OS_SEM_FAILURE (-6)
```

Semaphore failure.

Definition at line 54 of file osapi.h.

10.28.2.38 OS_SEM_TIMEOUT

```
#define OS_SEM_TIMEOUT (-7)
```

Semaphore timeout.

Definition at line 55 of file osapi.h.

10.28.2.39 OS_SUCCESS

```
#define OS_SUCCESS (0)
```

Successful execution.

Definition at line 48 of file osapi.h.

10.28.2.40 OS_TIMER_ERR_INTERNAL

```
#define OS_TIMER_ERR_INTERNAL (-32)
```

Timer internal error.

Definition at line 74 of file osapi.h.

10.28.2.41 OS_TIMER_ERR_INVALID_ARGS

```
#define OS_TIMER_ERR_INVALID_ARGS (-29)
```

Timer invalid arguments.

Definition at line 71 of file osapi.h.

10.28.2.42 OS_TIMER_ERR_TIMER_ID

```
#define OS_TIMER_ERR_TIMER_ID (-30)
```

Timer ID error.

Definition at line 72 of file osapi.h.

10.28.2.43 OS_TIMER_ERR_UNAVAILABLE

```
#define OS_TIMER_ERR_UNAVAILABLE (-31)
```

Timer unavailable.

Definition at line 73 of file osapi.h.

11 Data Structure Documentation

11.1 OS_bin_sem_prop_t Struct Reference

OSAL binary semaphore properties.

`#include <osapi-os-core.h>`

Data Fields

- char name [OS_MAX_API_NAME]
- uint32 creator
- int32 value

11.1.1 Detailed Description

OSAL binary semaphore properties.

Definition at line 87 of file osapi-os-core.h.

11.1.2 Field Documentation

11.1.2.1 creator

```
uint32 OS_bin_sem_prop_t::creator
```

Definition at line 90 of file osapi-os-core.h.

11.1.2.2 name

```
char OS_bin_sem_prop_t::name[OS_MAX_API_NAME]
```

Definition at line 89 of file osapi-os-core.h.

11.1.2.3 value

```
int32 OS_bin_sem_prop_t::value
```

Definition at line 91 of file osapi-os-core.h.

The documentation for this struct was generated from the following file:

- osal/src/os/inc/osapi-os-core.h

11.2 OS_count_sem_prop_t Struct Reference

OSAL counting semaphore properties.

`#include <osapi-os-core.h>`

Data Fields

- char name [OS_MAX_API_NAME]
- uint32 creator
- int32 value

11.2.1 Detailed Description

OSAL counting semaphore properties.

Definition at line 95 of file osapi-os-core.h.

11.2.2 Field Documentation

11.2.2.1 creator

`uint32 OS_count_sem_prop_t::creator`

Definition at line 98 of file osapi-os-core.h.

11.2.2.2 name

`char OS_count_sem_prop_t::name[OS_MAX_API_NAME]`

Definition at line 97 of file osapi-os-core.h.

11.2.2.3 value

`int32 OS_count_sem_prop_t::value`

Definition at line 99 of file osapi-os-core.h.

The documentation for this struct was generated from the following file:

- osal/src/os/inc/osapi-os-core.h

11.3 os_dirent_t Struct Reference

Directory entry.

`#include <osapi-os-filesys.h>`

Data Fields

- char FileName [OS_MAX_PATH_LEN]

11.3.1 Detailed Description

Directory entry.

Definition at line 190 of file osapi-os-filesys.h.

11.3.2 Field Documentation

11.3.2.1 FileName

`char os_dirent_t::FileName[OS_MAX_PATH_LEN]`

Definition at line 192 of file osapi-os-filesys.h.

The documentation for this struct was generated from the following file:

- osal/src/os/inc/osapi-os-filesys.h

11.4 OS_FdSet Struct Reference

An abstract structure capable of holding several OSAL IDs.

`#include <osapi-os-core.h>`

Data Fields

- uint8 object_ids [(OS_MAX_NUM_OPEN_FILES+7)/8]

11.4.1 Detailed Description

An abstract structure capable of holding several OSAL IDs.

This is part of the select API and is manipulated using the related API calls. It should not be modified directly by applications.

See also

OS_SelectFdZero(), OS_SelectFdAdd(), OS_SelectFdClear(), OS_SelectFdIsSet()

Definition at line 136 of file osapi-os-core.h.

11.4.2 Field Documentation

11.4.2.1 object_ids

```
uint8 OS_FdSet::object_ids[(OS_MAX_NUM_OPEN_FILES+7)/8]
```

Definition at line 138 of file osapi-os-core.h.

The documentation for this struct was generated from the following file:

- osal/src/os/inc/osapi-os-core.h

11.5 OS_file_prop_t Struct Reference

OSAL file properties.

```
#include <osapi-os-filesys.h>
```

Data Fields

- char Path [OS_MAX_PATH_LEN]
- uint32 User
- uint8 IsValid

11.5.1 Detailed Description

OSAL file properties.

Definition at line 136 of file osapi-os-filesys.h.

11.5.2 Field Documentation

11.5.2.1 IsValid

```
uint8 OS_file_prop_t::IsValid
```

Definition at line 140 of file osapi-os-filesys.h.

11.5.2.2 Path

```
char OS_file_prop_t::Path[OS_MAX_PATH_LEN]
```

Definition at line 138 of file osapi-os-filesys.h.

11.5.2.3 User

```
uint32 OS_file_prop_t::User
```

Definition at line 139 of file osapi-os-filesys.h.

The documentation for this struct was generated from the following file:

- osal/src/os/inc/osapi-os-filesys.h

11.6 os_fsinfo_t Struct Reference

OSAL file system info.

```
#include <osapi-os-filesys.h>
```

Data Fields

- uint32 MaxFds

  Total number of file descriptors.
- uint32 FreeFds

  Total number that are free.
- uint32 MaxVolumes

  Maximum number of volumes.
- uint32 FreeVolumes

  Total number of volumes free.

11.6.1 Detailed Description

OSAL file system info.

Definition at line 127 of file osapi-os-filesys.h.

11.6.2 Field Documentation

11.6.2.1 FreeFds

```
uint32 os_fsinfo_t::FreeFds
```

Total number that are free.

Definition at line 130 of file osapi-os-filesys.h.

11.6.2.2 FreeVolumes

```
uint32 os_fsinfo_t::FreeVolumes
```

Total number of volumes free.

Definition at line 132 of file osapi-os-filesys.h.

11.6.2.3 MaxFds

```
uint32 os_fsinfo_t::MaxFds
```

Total number of file descriptors.

Definition at line 129 of file osapi-os-filesys.h.

11.6.2.4 MaxVolumes

```
uint32 os_fsinfo_t::MaxVolumes
```

Maximum number of volumes.

Definition at line 131 of file osapi-os-filesys.h.

The documentation for this struct was generated from the following file:

- osal/src/os/inc/osapi-os-filesys.h

11.7 os_fstat_t Struct Reference

File system status.

`#include <osapi-os-filesys.h>`

Data Fields

- uint32 FileModeBits
- int32 FileTime
- uint32 FileSize

11.7.1 Detailed Description

File system status.

Note

This used to be directly typedef'ed to the "struct stat" from the C library

Some C libraries (glibc in particular) actually define member names to reference into sub-structures, so attempting to reuse a name like "st_mtime" might not work.

Definition at line 151 of file osapi-os-filesys.h.

11.7.2 Field Documentation

11.7.2.1 FileModeBits

`uint32 os_fstat_t::FileModeBits`

Definition at line 153 of file osapi-os-filesys.h.

11.7.2.2 FileSize

`uint32 os_fstat_t::FileSize`

Definition at line 155 of file osapi-os-filesys.h.

11.7.2.3 FileTime

```
int32 os_fstat_t::FileTime
```

Definition at line 154 of file osapi-os-filesys.h.

The documentation for this struct was generated from the following file:

- osal/src/os/inc/osapi-os-filesys.h

11.8 OS_heap_prop_t Struct Reference

OSAL heap properties.

```
#include <osapi-os-core.h>
```

Data Fields

- uint32 free_bytes
- uint32 free_blocks
- uint32 largest_free_block

11.8.1 Detailed Description

OSAL heap properties.

See also

OS_HeapGetInfo()

Definition at line 121 of file osapi-os-core.h.

11.8.2 Field Documentation

11.8.2.1 free_blocks

```
uint32 OS_heap_prop_t::free_blocks
```

Definition at line 124 of file osapi-os-core.h.

11.8.2.2 free_bytes

```
uint32 OS_heap_prop_t::free_bytes
```

Definition at line 123 of file osapi-os-core.h.

11.8.2.3 largest_free_block

```
uint32 OS_heap_prop_t::largest_free_block
```

Definition at line 125 of file osapi-os-core.h.

The documentation for this struct was generated from the following file:

- osal/src/os/inc/osapi-os-core.h

11.9 OS_module_address_t Struct Reference

OSAL module address properties.

```
#include <osapi-os-loader.h>
```

Data Fields

- uint32 valid
- uint32 flags
- cpuaddr code_address
- cpuaddr code_size
- cpuaddr data_address
- cpuaddr data_size
- cpuaddr bss_address
- cpuaddr bss_size

11.9.1 Detailed Description

OSAL module address properties.

Definition at line 32 of file osapi-os-loader.h.

11.9.2 Field Documentation

11.9.2.1 bss_address

```
cpuaddr OS_module_address_t::bss_address
```

Definition at line 40 of file osapi-os-loader.h.

11.9.2.2 bss_size

```
cpuaddr OS_module_address_t::bss_size
```

Definition at line 41 of file osapi-os-loader.h.

11.9.2.3 code_address

```
cpuaddr OS_module_address_t::code_address
```

Definition at line 36 of file osapi-os-loader.h.

11.9.2.4 code_size

```
cpuaddr OS_module_address_t::code_size
```

Definition at line 37 of file osapi-os-loader.h.

11.9.2.5 data_address

```
cpuaddr OS_module_address_t::data_address
```

Definition at line 38 of file osapi-os-loader.h.

11.9.2.6 data_size

```
cpuaddr OS_module_address_t::data_size
```

Definition at line 39 of file osapi-os-loader.h.

11.9.2.7 flags

```
uint32 OS_module_address_t::flags
```

Definition at line 35 of file osapi-os-loader.h.

11.9.2.8 valid

```
uint32 OS_module_address_t::valid
```

Definition at line 34 of file osapi-os-loader.h.

The documentation for this struct was generated from the following file:

- osal/src/os/inc/osapi-os-loader.h

11.10 OS_module_prop_t Struct Reference

OSAL module properties.

```
#include <osapi-os-loader.h>
```

Data Fields

- cpuaddr entry_point
- cpuaddr host_module_id
- char filename [OS_MAX_PATH_LEN]
- char name [OS_MAX_API_NAME]
- OS_module_address_t addr

11.10.1 Detailed Description

OSAL module properties.

Definition at line 45 of file osapi-os-loader.h.

11.10.2 Field Documentation

11.10.2.1 addr

```
OS_module_address_t OS_module_prop_t::addr
```

Definition at line 51 of file osapi-os-loader.h.

11.10.2.2 entry_point

```
cpuaddr OS_module_prop_t::entry_point
```

Definition at line 47 of file osapi-os-loader.h.

11.10.2.3 filename

```
char OS_module_prop_t::filename[OS_MAX_PATH_LEN]
```

Definition at line 49 of file osapi-os-loader.h.

11.10.2.4 host_module_id

```
cpuaddr OS_module_prop_t::host_module_id
```

Definition at line 48 of file osapi-os-loader.h.

11.10.2.5 name

```
char OS_module_prop_t::name[OS_MAX_API_NAME]
```

Definition at line 50 of file osapi-os-loader.h.

The documentation for this struct was generated from the following file:

- osal/src/os/inc/osapi-os-loader.h

11.11 OS_mut_sem_prop_t Struct Reference

OSAL mutexe properties.

```
#include <osapi-os-core.h>
```

Data Fields

- char name [OS_MAX_API_NAME]
- uint32 creator

11.11.1 Detailed Description

OSAL mutexe properties.

Definition at line 103 of file osapi-os-core.h.

11.11.2 Field Documentation

11.11.2.1 creator

```
uint32 OS_mut_sem_prop_t::creator
```

Definition at line 106 of file osapi-os-core.h.

11.11.2.2 name

```
char OS_mut_sem_prop_t::name[OS_MAX_API_NAME]
```

Definition at line 105 of file osapi-os-core.h.

The documentation for this struct was generated from the following file:

- osal/src/os/inc/osapi-os-core.h

11.12 OS_queue_prop_t Struct Reference

OSAL queue properties.

```
#include <osapi-os-core.h>
```

Data Fields

- char name [OS_MAX_API_NAME]
- uint32 creator

11.12.1 Detailed Description

OSAL queue properties.

Definition at line 80 of file osapi-os-core.h.

11.12.2 Field Documentation

11.12.2.1 creator

```
uint32 OS_queue_prop_t::creator
```

Definition at line 83 of file osapi-os-core.h.

11.12.2.2 name

```
char OS_queue_prop_t::name[OS_MAX_API_NAME]
```

Definition at line 82 of file osapi-os-core.h.

The documentation for this struct was generated from the following file:

- osal/src/os/inc/osapi-os-core.h

11.13 OS_SockAddr_t Struct Reference

Encapsulates a generic network address.

```
#include <osapi-os-net.h>
```

Data Fields

- uint32 ActualLength

  Length of the actual address data.
- OS_SockAddrData_t AddrData

  Abstract Address data.

11.13.1 Detailed Description

Encapsulates a generic network address.

This is just an abstract buffer type that holds a network address. It is allocated for the worst-case size defined by OS_SOCKADDR_MAX_LEN, and the real size is stored within.

Definition at line 92 of file osapi-os-net.h.

11.13.2 Field Documentation

11.13.2.1 ActualLength

`uint32 OS_SockAddr_t::ActualLength`

Length of the actual address data.

Definition at line 94 of file osapi-os-net.h.

11.13.2.2 AddrData

`OS_SockAddrData_t OS_SockAddr_t::AddrData`

Abstract Address data.

Definition at line 95 of file osapi-os-net.h.

The documentation for this struct was generated from the following file:

- osal/src/os/inc/osapi-os-net.h

11.14 OS_SockAddrData_t Union Reference

Storage buffer for generic network address.

`#include <osapi-os-net.h>`

Data Fields

- uint8 Buffer [OS_SOCKADDR_MAX_LEN]

  Ensures length of at least OS_SOCKADDR_MAX_LEN.
- uint32 AlignU32

  Ensures uint32 alignment.
- void * AlignPtr

  Ensures pointer alignment.

11.14.1 Detailed Description

Storage buffer for generic network address.

This is a union type that helps to ensure a minimum alignment value for the data storage, such that it can be cast to the system-specific type without increasing alignment requirements.

Definition at line 78 of file osapi-os-net.h.

11.14.2 Field Documentation

11.14.2.1 AlignPtr

```
void* OS_SockAddrData_t::AlignPtr
```

Ensures pointer alignment.

Definition at line 82 of file osapi-os-net.h.

11.14.2.2 AlignU32

```
uint32 OS_SockAddrData_t::AlignU32
```

Ensures uint32 alignment.

Definition at line 81 of file osapi-os-net.h.

11.14.2.3 Buffer

```
uint8 OS_SockAddrData_t::Buffer[OS_SOCKADDR_MAX_LEN]
```

Ensures length of at least OS_SOCKADDR_MAX_LEN.

Definition at line 80 of file osapi-os-net.h.

The documentation for this union was generated from the following file:

- osal/src/os/inc/osapi-os-net.h

11.15 OS_socket_prop_t Struct Reference

Encapsulates socket properties.

`#include <osapi-os-net.h>`

Data Fields

- char name [OS_MAX_API_NAME]

  Name of the socket.
- uint32 creator

  OSAL TaskID which opened the socket.

11.15.1 Detailed Description

Encapsulates socket properties.

This is for consistency with other OSAL resource types. Currently no extra properties are exposed here but this could change in a future revision of OSAL as needed.

Definition at line 105 of file osapi-os-net.h.

11.15.2 Field Documentation

11.15.2.1 creator

`uint32 OS_socket_prop_t::creator`

OSAL TaskID which opened the socket.

Definition at line 108 of file osapi-os-net.h.

11.15.2.2 name

`char OS_socket_prop_t::name[OS_MAX_API_NAME]`

Name of the socket.

Definition at line 107 of file osapi-os-net.h.

The documentation for this struct was generated from the following file:

- osal/src/os/inc/osapi-os-net.h

11.16 OS_static_symbol_record_t Struct Reference

Associates a single symbol name with a memory address.

`#include <osapi-os-loader.h>`

Data Fields

- const char * Name
- void(* Address)(void)
- const char * Module

11.16.1 Detailed Description

Associates a single symbol name with a memory address.

If the OS_STATIC_SYMBOL_TABLE feature is enabled, then an array of these structures should be provided by the application. When the application needs to find a symbol address, the static table will be checked in addition to (or instead of) the OS/library-provided lookup function.

This static symbol allows systems that do not implement dynamic module loading to maintain the same semantics as dynamically loaded modules.

Definition at line 67 of file osapi-os-loader.h.

11.16.2 Field Documentation

11.16.2.1 Address

```
void(* OS_static_symbol_record_t::Address) (void)
```

Definition at line 70 of file osapi-os-loader.h.

11.16.2.2 Module

```
const char* OS_static_symbol_record_t::Module
```

Definition at line 71 of file osapi-os-loader.h.

11.16.2.3 Name

```
const char* OS_static_symbol_record_t::Name
```

Definition at line 69 of file osapi-os-loader.h.

The documentation for this struct was generated from the following file:

- osal/src/os/inc/osapi-os-loader.h

11.17 OS_task_prop_t Struct Reference

OSAL task properties.

```
#include <osapi-os-core.h>
```

Data Fields

- char name [OS_MAX_API_NAME]
- uint32 creator
- uint32 stack_size
- uint32 priority
- uint32 OStask_id

11.17.1 Detailed Description

OSAL task properties.

Definition at line 70 of file osapi-os-core.h.

11.17.2 Field Documentation

11.17.2.1 creator

```
uint32 OS_task_prop_t::creator
```

Definition at line 73 of file osapi-os-core.h.

11.17.2.2 name

```
char OS_task_prop_t::name[OS_MAX_API_NAME]
```

Definition at line 72 of file osapi-os-core.h.

11.17.2.3 OStask_id

```
uint32 OS_task_prop_t::OStask_id
```

Definition at line 76 of file osapi-os-core.h.

11.17.2.4 priority

```
uint32 OS_task_prop_t::priority
```

Definition at line 75 of file osapi-os-core.h.

11.17.2.5 stack_size

```
uint32 OS_task_prop_t::stack_size
```

Definition at line 74 of file osapi-os-core.h.

The documentation for this struct was generated from the following file:

- osal/src/os/inc/osapi-os-core.h

11.18 OS_time_t Struct Reference

OSAL time.

```
#include <osapi-os-core.h>
```

Data Fields

- uint32 seconds
- uint32 microsecs

11.18.1 Detailed Description

OSAL time.

Definition at line 111 of file osapi-os-core.h.

11.18.2 Field Documentation

11.18.2.1 microsecs

```
uint32 OS_time_t::microsecs
```

Definition at line 114 of file osapi-os-core.h.

11.18.2.2 seconds

```
uint32 OS_time_t::seconds
```

Definition at line 113 of file osapi-os-core.h.

The documentation for this struct was generated from the following file:

- osal/src/os/inc/osapi-os-core.h

11.19 OS_timebase_prop_t Struct Reference

Time base properties.

```
#include <osapi-os-timer.h>
```

Data Fields

- char name [OS_MAX_API_NAME]
- uint32 creator
- uint32 nominal_interval_time
- uint32 freerun_time
- uint32 accuracy

11.19.1 Detailed Description

Time base properties.

Definition at line 40 of file osapi-os-timer.h.

11.19.2 Field Documentation

11.19.2.1 accuracy

```
uint32 OS_timebase_prop_t::accuracy
```

Definition at line 46 of file osapi-os-timer.h.

11.19.2.2 creator

```
uint32 OS_timebase_prop_t::creator
```

Definition at line 43 of file osapi-os-timer.h.

11.19.2.3 freerun_time

```
uint32 OS_timebase_prop_t::freerun_time
```

Definition at line 45 of file osapi-os-timer.h.

11.19.2.4 name

```
char OS_timebase_prop_t::name[OS_MAX_API_NAME]
```

Definition at line 42 of file osapi-os-timer.h.

11.19.2.5 nominal_interval_time

```
uint32 OS_timebase_prop_t::nominal_interval_time
```

Definition at line 44 of file osapi-os-timer.h.

The documentation for this struct was generated from the following file:

- osal/src/os/inc/osapi-os-timer.h

11.20 OS_timer_prop_t Struct Reference

Timer properties.

`#include <osapi-os-timer.h>`

Data Fields

- char name [OS_MAX_API_NAME]
- uint32 creator
- uint32 start_time
- uint32 interval_time
- uint32 accuracy

11.20.1 Detailed Description

Timer properties.

Definition at line 29 of file osapi-os-timer.h.

11.20.2 Field Documentation

11.20.2.1 accuracy

`uint32 OS_timer_prop_t::accuracy`

Definition at line 35 of file osapi-os-timer.h.

11.20.2.2 creator

`uint32 OS_timer_prop_t::creator`

Definition at line 32 of file osapi-os-timer.h.

11.20.2.3 interval_time

`uint32 OS_timer_prop_t::interval_time`

Definition at line 34 of file osapi-os-timer.h.

11.20.2.4 name

```
char OS_timer_prop_t::name[OS_MAX_API_NAME]
```

Definition at line 31 of file osapi-os-timer.h.

11.20.2.5 start_time

```
uint32 OS_timer_prop_t::start_time
```

Definition at line 33 of file osapi-os-timer.h.

The documentation for this struct was generated from the following file:

- osal/src/os/inc/osapi-os-timer.h

11.21 OS_VolumeInfo_t Struct Reference

Internal structure of the OS volume table for mounted file systems and path translation.

```
#include <osapi-os-filesys.h>
```

Data Fields

- char DeviceName [OS_FS_DEV_NAME_LEN]
- char PhysDevName [OS_FS_PHYS_NAME_LEN]
- uint32 VolumeType
- uint8 VolatileFlag
- uint8 FreeFlag
- uint8 IsMounted
- char VolumeName [OS_FS_VOL_NAME_LEN]
- char MountPoint [OS_MAX_PATH_LEN]
- uint32 BlockSize

11.21.1 Detailed Description

Internal structure of the OS volume table for mounted file systems and path translation.

Definition at line 112 of file osapi-os-filesys.h.

11.21.2 Field Documentation

11.21.2.1 BlockSize

```
uint32 OS_VolumeInfo_t::BlockSize
```

Definition at line 122 of file osapi-os-filesys.h.

11.21.2.2 DeviceName

```
char OS_VolumeInfo_t::DeviceName[OS_FS_DEV_NAME_LEN]
```

Definition at line 114 of file osapi-os-filesys.h.

11.21.2.3 FreeFlag

```
uint8 OS_VolumeInfo_t::FreeFlag
```

Definition at line 118 of file osapi-os-filesys.h.

11.21.2.4 IsMounted

```
uint8 OS_VolumeInfo_t::IsMounted
```

Definition at line 119 of file osapi-os-filesys.h.

11.21.2.5 MountPoint

```
char OS_VolumeInfo_t::MountPoint[OS_MAX_PATH_LEN]
```

Definition at line 121 of file osapi-os-filesys.h.

11.21.2.6 PhysDevName

```
char OS_VolumeInfo_t::PhysDevName[OS_FS_PHYS_NAME_LEN]
```

Definition at line 115 of file osapi-os-filesys.h.

11.21.2.7 VolatileFlag

```
uint8 OS_VolumeInfo_t::VolatileFlag
```

Definition at line 117 of file osapi-os-filesys.h.

11.21.2.8 VolumeName

```
char OS_VolumeInfo_t::VolumeName[OS_FS_VOL_NAME_LEN]
```

Definition at line 120 of file osapi-os-filesys.h.

11.21.2.9 VolumeType

```
uint32 OS_VolumeInfo_t::VolumeType
```

Definition at line 116 of file osapi-os-filesys.h.

The documentation for this struct was generated from the following file:

- osal/src/os/inc/osapi-os-filesys.h

12 File Documentation

12.1 cfe/docs/src/osal_fs.dox File Reference

12.2 cfe/docs/src/osal_timer.dox File Reference

12.3 cfe/docs/src/osalmain.dox File Reference

12.4 osal/src/os/inc/common_types.h File Reference

```
#include <stdint.h>
#include <stddef.h>
#include <stdbool.h>
```

Macros

- #define CompileTimeAssert(Condition, Message) typedef char Message[(Condition) ? 1 : -1]
- #define _EXTENSION_
- #define OS_PACK
- #define OS_ALIGN(n)
- #define OS_USED
- #define OS_PRINTF(n, m)
- #define TRUE true
- #define FALSE false
- #define NULL ((void *) 0)

Typedefs

- typedef int8_t int8
- typedef int16_t int16
- typedef int32_t int32
- typedef int64_t int64
- typedef uint8_t uint8
- typedef uint16_t uint16
- typedef uint32_t uint32
- typedef uint64_t uint64
- typedef intptr_t intptr
- typedef uintptr_t cpuaddr
- typedef size_t cpusize
- typedef ptrdiff_t cpudiff
- typedef bool osalbool
- typedef osalbool boolean

Functions

- CompileTimeAssert (sizeof(uint8)==1, TypeUint8WrongSize)
- CompileTimeAssert (sizeof(uint16)==2, TypeUint16WrongSize)
- CompileTimeAssert (sizeof(uint32)==4, TypeUint32WrongSize)
- CompileTimeAssert (sizeof(uint64)==8, TypeUint64WrongSize)
- CompileTimeAssert (sizeof(int8)==1, Typeint8WrongSize)
- CompileTimeAssert (sizeof(int16)==2, Typeint16WrongSize)
- CompileTimeAssert (sizeof(int32)==4, Typeint32WrongSize)
- CompileTimeAssert (sizeof(int64)==8, Typeint64WrongSize)
- CompileTimeAssert (sizeof(cpuaddr) >=sizeof(void *), TypePtrWrongSize)

12.4.1 Macro Definition Documentation

12.4.1.1 _EXTENSION_

```
#define _EXTENSION_
```

Definition at line 65 of file common_types.h.

12.4.1.2 CompileTimeAssert

```
#define CompileTimeAssert(
              Condition,
              Message ) typedef char Message[(Condition) ?  1 :  -1]
```

Definition at line 44 of file common_types.h.

12.4.1.3 FALSE

```
#define FALSE false
```

Deprecated Use false

Definition at line 127 of file common_types.h.

12.4.1.4 NULL

```
#define NULL ((void *) 0)
```

Definition at line 135 of file common_types.h.

12.4.1.5 OS_ALIGN

```
#define OS_ALIGN(
              n )
```

Definition at line 67 of file common_types.h.

12.4.1.6 OS_PACK

```
#define OS_PACK
```

Definition at line 66 of file common_types.h.

12.4.1.7 OS_PRINTF

```
#define OS_PRINTF(
              n,
              m )
```

Definition at line 69 of file common_types.h.

12.4.1.8 OS_USED

```
#define OS_USED
```

Definition at line 68 of file common_types.h.

12.4.1.9 TRUE

```
#define TRUE true
```

Deprecated Use true

Definition at line 123 of file common_types.h.

12.4.2 Typedef Documentation

12.4.2.1 boolean

```
typedef osalbool boolean
```

Deprecated Use bool

Definition at line 119 of file common_types.h.

12.4.2.2 cpuaddr

```
typedef uintptr_t cpuaddr
```

Definition at line 90 of file common_types.h.

12.4.2.3 cpudiff

```
typedef ptrdiff_t cpudiff
```

Definition at line 92 of file common_types.h.

12.4.2.4 cpusize

```
typedef size_t cpusize
```

Definition at line 91 of file common_types.h.

12.4.2.5 int16

```
typedef int16_t int16
```

Definition at line 82 of file common_types.h.

12.4.2.6 int32

```
typedef int32_t int32
```

Definition at line 83 of file common_types.h.

12.4.2.7 int64

```
typedef int64_t int64
```

Definition at line 84 of file common_types.h.

12.4.2.8 int8

```
typedef int8_t int8
```

Definition at line 81 of file common_types.h.

12.4.2.9 intptr

```
typedef intptr_t intptr
```

Definition at line 89 of file common_types.h.

12.4.2.10 osalbool

```
typedef bool osalbool
```

Deprecated Use bool

Definition at line 100 of file common_types.h.

12.4.2.11 uint16

```
typedef uint16_t uint16
```

Definition at line 86 of file common_types.h.

12.4.2.12 uint32

```
typedef uint32_t uint32
```

Definition at line 87 of file common_types.h.

12.4.2.13 uint64

```
typedef uint64_t uint64
```

Definition at line 88 of file common_types.h.

12.4.2.14 uint8

```
typedef uint8_t uint8
```

Definition at line 85 of file common_types.h.

12.4.3 Function Documentation

12.4.3.1 CompileTimeAssert() [1/9]

```
CompileTimeAssert (
             sizeof(uint8)  == 1,
             TypeUint8WrongSize  )
```

12.4.3.2 CompileTimeAssert() [2/9]

```
CompileTimeAssert (
             sizeof(uint16)  == 2,
             TypeUint16WrongSize  )
```

12.4.3.3 CompileTimeAssert() [3/9]

```
CompileTimeAssert (
             sizeof(uint32)  == 4,
             TypeUint32WrongSize  )
```

12.4.3.4 CompileTimeAssert() [4/9]

```
CompileTimeAssert (
             sizeof(uint64)  == 8,
             TypeUint64WrongSize  )
```

12.4.3.5 CompileTimeAssert() [5/9]

```
CompileTimeAssert (
             sizeof(int8)  == 1,
             Typeint8WrongSize  )
```

12.4.3.6 CompileTimeAssert() [6/9]

```
CompileTimeAssert (
             sizeof(int16)  == 2,
             Typeint16WrongSize  )
```

12.4.3.7 CompileTimeAssert() [7/9]

```
CompileTimeAssert (
            sizeof(int32)  = =4,
            Typeint32WrongSize  )
```

12.4.3.8 CompileTimeAssert() [8/9]

```
CompileTimeAssert (
            sizeof(int64)  = =8,
            Typeint64WrongSize  )
```

12.4.3.9 CompileTimeAssert() [9/9]

```
CompileTimeAssert (
            sizeof(cpuaddr) >=sizeof(void *) ,
            TypePtrWrongSize  )
```

12.5 osal/src/os/inc/osapi-os-core.h File Reference

```
#include <stdarg.h>
```

Data Structures

- struct OS_task_prop_t

  OSAL task properties.
- struct OS_queue_prop_t

  OSAL queue properties.
- struct OS_bin_sem_prop_t

  OSAL binary semaphore properties.
- struct OS_count_sem_prop_t

  OSAL counting semaphore properties.
- struct OS_mut_sem_prop_t

  OSAL mutexe properties.
- struct OS_time_t

  OSAL time.
- struct OS_heap_prop_t

  OSAL heap properties.
- struct OS_FdSet

  An abstract structure capable of holding several OSAL IDs.

Macros

- #define OS_OBJECT_INDEX_MASK 0xFFFF

  Object index mask.
- #define OS_OBJECT_TYPE_SHIFT 16

  Object type shift.
- #define OS_OBJECT_TYPE_UNDEFINED 0x00

  Object type undefined.
- #define OS_OBJECT_TYPE_OS_TASK 0x01

  Object task type.
- #define OS_OBJECT_TYPE_OS_QUEUE 0x02

  Object queue type.
- #define OS_OBJECT_TYPE_OS_COUNTSEM 0x03

  Object counting semaphore type.
- #define OS_OBJECT_TYPE_OS_BINSEM 0x04

  Object binary semaphore type.
- #define OS_OBJECT_TYPE_OS_MUTEX 0x05

  Object mutex type.
- #define OS_OBJECT_TYPE_OS_STREAM 0x06

  Object stream type.
- #define OS_OBJECT_TYPE_OS_DIR 0x07

  Object directory type.
- #define OS_OBJECT_TYPE_OS_TIMEBASE 0x08

  Object timebase type.
- #define OS_OBJECT_TYPE_OS_TIMECB 0x09

  Object timer callback type.
- #define OS_OBJECT_TYPE_OS_MODULE 0x0A

  Object module type.
- #define OS_OBJECT_TYPE_OS_FILESYS 0x0B

  Object file system type.
- #define OS_OBJECT_TYPE_OS_CONSOLE 0x0C

  Object console type.
- #define OS_OBJECT_TYPE_USER 0x10

  Object user type.
- #define OS_MAX_TASK_PRIORITY 255

  Upper limit for OSAL task priorities.
- #define OS_SEM_FULL 1

  Semaphore full state.
- #define OS_SEM_EMPTY 0

  Semaphore empty state.
- #define OS_FP_ENABLED 1

  Floating point enabled state for a task.
- #define OS_ERROR_NAME_LENGTH 35

  Error string name length.

Typedefs

- typedef char os_err_name_t[OS_ERROR_NAME_LENGTH]

  For the OS_GetErrorName() function, to ensure everyone is making an array of the same length.
- typedef void osal_task

  For task entry point.
- typedef void(∗ OS_ArgCallback_t) (uint32 object_id, void ∗arg)

  General purpose OSAL callback function.

Functions

- typedef osal_task ((∗osal_task_entry)(void))

  For task entry point.
- void OS_Application_Startup (void)

  Application startup.
- void OS_Application_Run (void)

  Application run.
- int32 OS_API_Init (void)

  Initialization of API.
- void OS_IdleLoop (void)

  Background thread implementation - waits forever for events to occur.
- void OS_DeleteAllObjects (void)

  delete all resources created in OSAL.
- void OS_ApplicationShutdown (uint8 flag)

  Initiate orderly shutdown.
- void OS_ApplicationExit (int32 Status)

  Exit/Abort the application.
- uint32 OS_IdentifyObject (uint32 object_id)

  Obtain the type of an object given an arbitrary object ID.
- int32 OS_ConvertToArrayIndex (uint32 object_id, uint32 ∗ArrayIndex)

  Converts an abstract ID into a number suitable for use as an array index.
- void OS_ForEachObject (uint32 creator_id, OS_ArgCallback_t callback_ptr, void ∗callback_arg)

  call the supplied callback function for all valid object IDs
- int32 OS_TaskCreate (uint32 ∗task_id, const char ∗task_name, osal_task_entry function_pointer, uint32 ∗stack↩ _pointer, uint32 stack_size, uint32 priority, uint32 flags)

  Creates a task and starts running it.
- int32 OS_TaskDelete (uint32 task_id)

  Deletes the specified Task.
- void OS_TaskExit (void)

  Exits the calling task.
- int32 OS_TaskInstallDeleteHandler (osal_task_entry function_pointer)

  Installs a handler for when the task is deleted.
- int32 OS_TaskDelay (uint32 millisecond)

  Delay a task for specified amount of milliseconds.
- int32 OS_TaskSetPriority (uint32 task_id, uint32 new_priority)

  Sets the given task to a new priority.
- int32 OS_TaskRegister (void)

Obsolete.

- uint32 OS_TaskGetId (void)

  Obtain the task id of the calling task.

- int32 OS_TaskGetIdByName (uint32 *task_id, const char *task_name)

  Find an existing task ID by name.

- int32 OS_TaskGetInfo (uint32 task_id, OS_task_prop_t *task_prop)

  Fill a property object buffer with details regarding the resource.

- int32 OS_QueueCreate (uint32 *queue_id, const char *queue_name, uint32 queue_depth, uint32 data_size, uint32 flags)

  Create a message queue.

- int32 OS_QueueDelete (uint32 queue_id)

  Deletes the specified message queue.

- int32 OS_QueueGet (uint32 queue_id, void *data, uint32 size, uint32 *size_copied, int32 timeout)

  Receive a message on a message queue.

- int32 OS_QueuePut (uint32 queue_id, const void *data, uint32 size, uint32 flags)

  Put a message on a message queue.

- int32 OS_QueueGetIdByName (uint32 *queue_id, const char *queue_name)

  Find an existing queue ID by name.

- int32 OS_QueueGetInfo (uint32 queue_id, OS_queue_prop_t *queue_prop)

  Fill a property object buffer with details regarding the resource.

- int32 OS_BinSemCreate (uint32 *sem_id, const char *sem_name, uint32 sem_initial_value, uint32 options)

  Creates a binary semaphore.

- int32 OS_BinSemFlush (uint32 sem_id)

  Unblock all tasks pending on the specified semaphore.

- int32 OS_BinSemGive (uint32 sem_id)

  Increment the semaphore value.

- int32 OS_BinSemTake (uint32 sem_id)

  Decrement the semaphore value.

- int32 OS_BinSemTimedWait (uint32 sem_id, uint32 msecs)

  Decrement the semaphore value with a timeout.

- int32 OS_BinSemDelete (uint32 sem_id)

  Deletes the specified Binary Semaphore.

- int32 OS_BinSemGetIdByName (uint32 *sem_id, const char *sem_name)

  Find an existing semaphore ID by name.

- int32 OS_BinSemGetInfo (uint32 sem_id, OS_bin_sem_prop_t *bin_prop)

  Fill a property object buffer with details regarding the resource.

- int32 OS_CountSemCreate (uint32 *sem_id, const char *sem_name, uint32 sem_initial_value, uint32 options)

  Creates a counting semaphore.

- int32 OS_CountSemGive (uint32 sem_id)

  Increment the semaphore value.

- int32 OS_CountSemTake (uint32 sem_id)

  Decrement the semaphore value.

- int32 OS_CountSemTimedWait (uint32 sem_id, uint32 msecs)

  Decrement the semaphore value with timeout.

- int32 OS_CountSemDelete (uint32 sem_id)

  Deletes the specified counting Semaphore.

- int32 OS_CountSemGetIdByName (uint32 *sem_id, const char *sem_name)

Find an existing semaphore ID by name.

- int32 OS_CountSemGetInfo (uint32 sem_id, OS_count_sem_prop_t *count_prop)

  Fill a property object buffer with details regarding the resource.

- int32 OS_MutSemCreate (uint32 *sem_id, const char *sem_name, uint32 options)

  Creates a mutex semaphore.

- int32 OS_MutSemGive (uint32 sem_id)

  Releases the mutex object referenced by sem_id.

- int32 OS_MutSemTake (uint32 sem_id)

  Acquire the mutex object referenced by sem_id.

- int32 OS_MutSemDelete (uint32 sem_id)

  Deletes the specified Mutex Semaphore.

- int32 OS_MutSemGetIdByName (uint32 *sem_id, const char *sem_name)

  Find an existing mutex ID by name.

- int32 OS_MutSemGetInfo (uint32 sem_id, OS_mut_sem_prop_t *mut_prop)

  Fill a property object buffer with details regarding the resource.

- int32 OS_Milli2Ticks (uint32 milli_seconds)

  Convert time units from milliseconds to system ticks.

- int32 OS_Tick2Micros (void)

  Get the system tick size, in microseconds.

- int32 OS_GetLocalTime (OS_time_t *time_struct)

  Get the local time.

- int32 OS_SetLocalTime (OS_time_t *time_struct)

  Set the local time.

- int32 OS_ExcAttachHandler (uint32 ExceptionNumber, void(*ExceptionHandler)(uint32, const void *, uint32), int32 parameter)

  placeholder; not currently implemented

- int32 OS_ExcEnable (int32 ExceptionNumber)

  placeholder; not currently implemented

- int32 OS_ExcDisable (int32 ExceptionNumber)

  placeholder; not currently implemented

- int32 OS_FPUExcAttachHandler (uint32 ExceptionNumber, osal_task_entry ExceptionHandler, int32 parameter)

  Set an FPU exception handler function.

- int32 OS_FPUExcEnable (int32 ExceptionNumber)

  Enable FPU exceptions.

- int32 OS_FPUExcDisable (int32 ExceptionNumber)

  Disable FPU exceptions.

- int32 OS_FPUExcSetMask (uint32 mask)

  Sets the FPU exception mask.

- int32 OS_FPUExcGetMask (uint32 *mask)

  Gets the FPU exception mask.

- int32 OS_IntAttachHandler (uint32 InterruptNumber, osal_task_entry InterruptHandler, int32 parameter)

  Associate an interrupt number to a specified handler routine.

- int32 OS_IntUnlock (int32 IntLevel)

  Enable interrupts.

- int32 OS_IntLock (void)

  Disable interrupts.

- int32 OS_IntEnable (int32 Level)

Enables interrupts through Level.

- int32 OS_IntDisable (int32 Level)

  Disable interrupts through Level.

- int32 OS_IntSetMask (uint32 mask)

  Set the CPU interrupt mask register.

- int32 OS_IntGetMask (uint32 *mask)

  Get the CPU interrupt mask register.

- int32 OS_IntAck (int32 InterruptNumber)

  Acknowledge the corresponding interrupt number.

- int32 OS_ShMemInit (void)

  placeholder; not currently implemented

- int32 OS_ShMemCreate (uint32 *Id, uint32 NBytes, const char *SegName)

  placeholder; not currently implemented

- int32 OS_ShMemSemTake (uint32 Id)

  placeholder; not currently implemented

- int32 OS_ShMemSemGive (uint32 Id)

  placeholder; not currently implemented

- int32 OS_ShMemAttach (cpuaddr *Address, uint32 Id)

  placeholder; not currently implemented

- int32 OS_ShMemGetIdByName (uint32 *ShMemId, const char *SegName)

  placeholder; not currently implemented

- int32 OS_HeapGetInfo (OS_heap_prop_t *heap_prop)

  Return current info on the heap.

- int32 OS_GetErrorName (int32 error_num, os_err_name_t *err_name)

  Convert an error number to a string.

- int32 OS_SelectMultiple (OS_FdSet *ReadSet, OS_FdSet *WriteSet, int32 msecs)

  Wait for events across multiple file handles.

- int32 OS_SelectSingle (uint32 objid, uint32 *StateFlags, int32 msecs)

  Wait for events on a single file handle.

- int32 OS_SelectFdZero (OS_FdSet *Set)

  Clear a FdSet structure.

- int32 OS_SelectFdAdd (OS_FdSet *Set, uint32 objid)

  Add an ID to an FdSet structure.

- int32 OS_SelectFdClear (OS_FdSet *Set, uint32 objid)

  Clear an ID from an FdSet structure.

- bool OS_SelectFdIsSet (OS_FdSet *Set, uint32 objid)

  Check if an FdSet structure contains a given ID.

- void OS_printf (const char *string,...) OS_PRINTF(1

  Abstraction for the system printf() call.

- void void OS_printf_disable (void)

  This function disables the output from OS_printf.

- void OS_printf_enable (void)

  This function enables the output from OS_printf.

12.5.1 Macro Definition Documentation

12.5.1.1 OS_ERROR_NAME_LENGTH

```
#define OS_ERROR_NAME_LENGTH 35
```

Error string name length.

The sizes of strings in OSAL functions are built with this limit in mind. Always check the uses of os_err_name_t when changing this value.

Definition at line 65 of file osapi-os-core.h.

12.5.1.2 OS_FP_ENABLED

```
#define OS_FP_ENABLED 1
```

Floating point enabled state for a task.

Definition at line 58 of file osapi-os-core.h.

12.5.1.3 OS_MAX_TASK_PRIORITY

```
#define OS_MAX_TASK_PRIORITY 255
```

Upper limit for OSAL task priorities.

Definition at line 48 of file osapi-os-core.h.

12.5.1.4 OS_OBJECT_INDEX_MASK

```
#define OS_OBJECT_INDEX_MASK 0xFFFF
```

Object index mask.

Definition at line 25 of file osapi-os-core.h.

12.5.1.5 OS_OBJECT_TYPE_SHIFT

```
#define OS_OBJECT_TYPE_SHIFT 16
```

Object type shift.

Definition at line 26 of file osapi-os-core.h.

12.5.2 Typedef Documentation

12.5.2.1 OS_ArgCallback_t

```
typedef void(* OS_ArgCallback_t) (uint32 object_id, void *arg)
```

General purpose OSAL callback function.

This may be used by multiple APIS

Definition at line 164 of file osapi-os-core.h.

12.5.2.2 os_err_name_t

```
typedef char os_err_name_t[OS_ERROR_NAME_LENGTH]
```

For the OS_GetErrorName() function, to ensure everyone is making an array of the same length.

Implementation note for developers:

The sizes of strings in OSAL functions are built with this OS_ERROR_NAME_LENGTH limit in mind. Always check the uses of os_err_name_t when changing this value.

Definition at line 151 of file osapi-os-core.h.

12.5.2.3 osal_task

```
typedef void osal_task
```

For task entry point.

Definition at line 156 of file osapi-os-core.h.

12.5.3 Function Documentation

12.5.3.1 osal_task()

```
typedef osal_task (
            (*)(void) osal_task_entry )
```

For task entry point.

12.6 osal/src/os/inc/osapi-os-filesys.h File Reference

Data Structures

- struct OS_VolumeInfo_t

  Internal structure of the OS volume table for mounted file systems and path translation.
- struct os_fsinfo_t

  OSAL file system info.
- struct OS_file_prop_t

  OSAL file properties.
- struct os_fstat_t

  File system status.
- struct os_dirent_t

  Directory entry.

Macros

- #define OS_READ_ONLY 0
- #define OS_WRITE_ONLY 1
- #define OS_READ_WRITE 2
- #define OS_SEEK_SET 0
- #define OS_SEEK_CUR 1
- #define OS_SEEK_END 2
- #define OS_CHK_ONLY 0
- #define OS_REPAIR 1
- #define FS_BASED 0
- #define RAM_DISK 1
- #define EEPROM_DISK 2
- #define ATA_DISK 3
- #define NUM_TABLE_ENTRIES 14

  Number of entries in the internal volume table.
- #define OS_FS_DEV_NAME_LEN 32
- #define OS_FS_PHYS_NAME_LEN 64
- #define OS_FS_VOL_NAME_LEN 32
- #define OS_FS_ERR_PATH_TOO_LONG (-103)

  FS path too long.
- #define OS_FS_ERR_NAME_TOO_LONG (-104)

  FS name too long.
- #define OS_FS_ERR_DRIVE_NOT_CREATED (-106)

  FS drive not created.
- #define OS_FS_ERR_DEVICE_NOT_FREE (-107)

  FS device not free.
- #define OS_FS_ERR_PATH_INVALID (-108)

  FS path invalid.
- #define OS_FS_SUCCESS OS_SUCCESS

  Successful execution.
- #define OS_FS_ERROR OS_ERROR

  Failed execution.

- #define OS_FS_ERR_INVALID_POINTER OS_INVALID_POINTER

  Invalid pointer.
- #define OS_FS_ERR_NO_FREE_FDS OS_ERR_NO_FREE_IDS

  No free IDs.
- #define OS_FS_ERR_INVALID_FD OS_ERR_INVALID_ID

  Invalid ID.
- #define OS_FS_UNIMPLEMENTED OS_ERR_NOT_IMPLEMENTED

  Not implemented.
- #define OS_FILESTAT_MODE(x) ((x).FileModeBits)

  Access file stat mode bits.
- #define OS_FILESTAT_ISDIR(x) ((x).FileModeBits & OS_FILESTAT_MODE_DIR)

  File stat is directory logical.
- #define OS_FILESTAT_EXEC(x) ((x).FileModeBits & OS_FILESTAT_MODE_EXEC)

  File stat is executable logical.
- #define OS_FILESTAT_WRITE(x) ((x).FileModeBits & OS_FILESTAT_MODE_WRITE)

  File stat is write enabled logical.
- #define OS_FILESTAT_READ(x) ((x).FileModeBits & OS_FILESTAT_MODE_READ)

  File stat is read enabled logical.
- #define OS_FILESTAT_SIZE(x) ((x).FileSize)

  Access file stat size field.
- #define OS_FILESTAT_TIME(x) ((x).FileTime)

  Access file stat time field.
- #define OS_DIRENTRY_NAME(x) ((x).FileName)

  Access filename part of the dirent structure.

Typedefs

- typedef os_err_name_t os_fs_err_name_t
- typedef void * os_dirp_t
- typedef int32 os_fshealth_t
- typedef OS_file_prop_t OS_FDTableEntry

Enumerations

- enum { OS_FILESTAT_MODE_EXEC = 0x00001, OS_FILESTAT_MODE_WRITE = 0x00002, OS_FILESTAT↩
  _MODE_READ = 0x00004, OS_FILESTAT_MODE_DIR = 0x10000 }

  File stat mode bits.

Functions

- int32 OS_creat (const char *path, int32 access)

  Creates a file specified by path.
- int32 OS_open (const char *path, int32 access, uint32 mode)

  Opens a file.
- int32 OS_close (uint32 filedes)

  Closes an open file handle.
- int32 OS_read (uint32 filedes, void *buffer, uint32 nbytes)

  Read from a file handle.
- int32 OS_write (uint32 filedes, const void *buffer, uint32 nbytes)

  Write to a file handle.
- int32 OS_TimedRead (uint32 filedes, void *buffer, uint32 nbytes, int32 timeout)

  File/Stream input read with a timeout.
- int32 OS_TimedWrite (uint32 filedes, const void *buffer, uint32 nbytes, int32 timeout)

  File/Stream output write with a timeout.
- int32 OS_chmod (const char *path, uint32 access)

  Changes the permissions of a file.
- int32 OS_stat (const char *path, os_fstat_t *filestats)

  Obtain information about a file or directory.
- int32 OS_lseek (uint32 filedes, int32 offset, uint32 whence)

  Seeks to the specified position of an open file.
- int32 OS_remove (const char *path)

  Removes a file from the file system.
- int32 OS_rename (const char *old_filename, const char *new_filename)

  Renames a file.
- int32 OS_cp (const char *src, const char *dest)

  Copies a single file from src to dest.
- int32 OS_mv (const char *src, const char *dest)

  Move a single file from src to dest.
- int32 OS_FDGetInfo (uint32 filedes, OS_file_prop_t *fd_prop)

  Obtain information about an open file.
- int32 OS_FileOpenCheck (const char *Filename)

  Checks to see if a file is open.
- int32 OS_CloseAllFiles (void)

  Close all open files.
- int32 OS_CloseFileByName (const char *Filename)

  Close a file by filename.
- os_dirp_t OS_opendir (const char *path)

  Opens a directory for searching.
- int32 OS_closedir (os_dirp_t directory)
- void OS_rewinddir (os_dirp_t directory)
- os_dirent_t * OS_readdir (os_dirp_t directory)
- int32 OS_DirectoryOpen (uint32 *dir_id, const char *path)

  Opens a directory.
- int32 OS_DirectoryClose (uint32 dir_id)

  Closes an open directory.

- int32 OS_DirectoryRewind (uint32 dir_id)

  Rewinds an open directory.
- int32 OS_DirectoryRead (uint32 dir_id, os_dirent_t *dirent)

  Reads the next name in the directory.
- int32 OS_mkdir (const char *path, uint32 access)

  Makes a new directory.
- int32 OS_rmdir (const char *path)

  Removes a directory from the file system.
- int32 OS_FileSysAddFixedMap (uint32 *filesys_id, const char *phys_path, const char *virt_path)

  Create a fixed mapping between an existing directory and a virtual OSAL mount point.
- int32 OS_mkfs (char *address, const char *devname, const char *volname, uint32 blocksize, uint32 numblocks)

  Makes a file system on the target.
- int32 OS_mount (const char *devname, const char *mountpoint)

  Mounts a file system.
- int32 OS_initfs (char *address, const char *devname, const char *volname, uint32 blocksize, uint32 numblocks)

  Initializes an existing file system.
- int32 OS_rmfs (const char *devname)

  Removes a file system.
- int32 OS_unmount (const char *mountpoint)

  Unmounts a mounted file system.
- int32 OS_fsBlocksFree (const char *name)

  Obtain number of blocks free.
- int32 OS_fsBytesFree (const char *name, uint64 *bytes_free)

  Obtains the number of free bytes in a volume.
- int32 OS_chkfs (const char *name, bool repair)

  Checks the health of a file system and repairs it if necessary.
- int32 OS_FS_GetPhysDriveName (char *PhysDriveName, const char *MountPoint)

  Obtains the physical drive name associated with a mount point.
- int32 OS_TranslatePath (const char *VirtualPath, char *LocalPath)

  Translates a OSAL Virtual file system path to a host Local path.
- int32 OS_GetFsInfo (os_fsinfo_t *filesys_info)

  Returns information about the file system.
- int32 OS_ShellOutputToFile (const char *Cmd, uint32 filedes)

  Executes the command and sends output to a file.

12.6.1 Macro Definition Documentation

12.6.1.1 NUM_TABLE_ENTRIES

```
#define NUM_TABLE_ENTRIES 14
```

Number of entries in the internal volume table.

Definition at line 53 of file osapi-os-filesys.h.

12.6.1.2 OS_CHK_ONLY

```
#define OS_CHK_ONLY 0
```

Unused, API takes bool

Definition at line 38 of file osapi-os-filesys.h.

12.6.1.3 OS_DIRENTRY_NAME

```
#define OS_DIRENTRY_NAME(
              x ) ((x).FileName)
```

Access filename part of the dirent structure.

Definition at line 205 of file osapi-os-filesys.h.

12.6.1.4 OS_FILESTAT_EXEC

```
#define OS_FILESTAT_EXEC(
              x ) ((x).FileModeBits & OS_FILESTAT_MODE_EXEC)
```

File stat is executable logical.

Definition at line 179 of file osapi-os-filesys.h.

12.6.1.5 OS_FILESTAT_ISDIR

```
#define OS_FILESTAT_ISDIR(
              x ) ((x).FileModeBits & OS_FILESTAT_MODE_DIR)
```

File stat is directory logical.

Definition at line 177 of file osapi-os-filesys.h.

12.6.1.6 OS_FILESTAT_MODE

```
#define OS_FILESTAT_MODE(
              x ) ((x).FileModeBits)
```

Access file stat mode bits.

Definition at line 175 of file osapi-os-filesys.h.

12.6.1.7 OS_FILESTAT_READ

```
#define OS_FILESTAT_READ(
              x ) ((x).FileModeBits & OS_FILESTAT_MODE_READ)
```

File stat is read enabled logical.

Definition at line 183 of file osapi-os-filesys.h.

12.6.1.8 OS_FILESTAT_SIZE

```
#define OS_FILESTAT_SIZE(
              x ) ((x).FileSize)
```

Access file stat size field.

Definition at line 185 of file osapi-os-filesys.h.

12.6.1.9 OS_FILESTAT_TIME

```
#define OS_FILESTAT_TIME(
              x ) ((x).FileTime)
```

Access file stat time field.

Definition at line 187 of file osapi-os-filesys.h.

12.6.1.10 OS_FILESTAT_WRITE

```
#define OS_FILESTAT_WRITE(
              x ) ((x).FileModeBits & OS_FILESTAT_MODE_WRITE)
```

File stat is write enabled logical.

Definition at line 181 of file osapi-os-filesys.h.

12.6.1.11 OS_FS_DEV_NAME_LEN

```
#define OS_FS_DEV_NAME_LEN 32
```

Device name length

Definition at line 58 of file osapi-os-filesys.h.

12.6.1.12 OS_FS_PHYS_NAME_LEN

```
#define OS_FS_PHYS_NAME_LEN 64
```

Physical drive name length

Definition at line 59 of file osapi-os-filesys.h.

12.6.1.13 OS_FS_VOL_NAME_LEN

```
#define OS_FS_VOL_NAME_LEN 32
```

Volume name length

Definition at line 60 of file osapi-os-filesys.h.

12.6.1.14 OS_REPAIR

```
#define OS_REPAIR 1
```

Unused, API takes bool

Definition at line 39 of file osapi-os-filesys.h.

12.6.2 Typedef Documentation

12.6.2.1 os_dirp_t

```
typedef void* os_dirp_t
```

Deprecated

Definition at line 201 of file osapi-os-filesys.h.

12.6.2.2 OS_FDTableEntry

```
typedef OS_file_prop_t OS_FDTableEntry
```

Deprecated Use OS_file_prop_t

Definition at line 212 of file osapi-os-filesys.h.

12.6.2.3 os_fs_err_name_t

```
typedef os_err_name_t os_fs_err_name_t
```

Definition at line 105 of file osapi-os-filesys.h.

12.6.2.4 os_fshealth_t

```
typedef int32 os_fshealth_t
```

Deprecated type no longer used

Definition at line 211 of file osapi-os-filesys.h.

12.6.3 Enumeration Type Documentation

12.6.3.1 anonymous enum

```
anonymous enum
```

File stat mode bits.

We must also define replacements for the stat structure's mode bits. This is currently just a small subset since the OSAL just presents a very simplified view of the filesystem to the upper layers. And since not all OS'es are POSIX, the more POSIX-specific bits are not relevant anyway.

Enumerator

OS_FILESTAT_MODE_EXEC	
OS_FILESTAT_MODE_WRITE	
OS_FILESTAT_MODE_READ	
OS_FILESTAT_MODE_DIR	

Definition at line 166 of file osapi-os-filesys.h.

12.7 osal/src/os/inc/osapi-os-loader.h File Reference

Data Structures

- struct OS_module_address_t

OSAL module address properties.

- struct OS_module_prop_t

  OSAL module properties.

- struct OS_static_symbol_record_t

  Associates a single symbol name with a memory address.

Typedefs

- typedef OS_module_prop_t OS_module_record_t

Functions

- int32 OS_SymbolLookup (cpuaddr *symbol_address, const char *symbol_name)

  Find the Address of a Symbol.

- int32 OS_SymbolTableDump (const char *filename, uint32 size_limit)

  Dumps the system symbol table to a file.

- int32 OS_ModuleLoad (uint32 *module_id, const char *module_name, const char *filename)

  Loads an object file.

- int32 OS_ModuleUnload (uint32 module_id)

  Unloads the module file.

- int32 OS_ModuleInfo (uint32 module_id, OS_module_prop_t *module_info)

  Obtain information about a module.

12.7.1 Typedef Documentation

12.7.1.1 OS_module_record_t

```
typedef OS_module_prop_t OS_module_record_t
```

Deprecated Use OS_module_prop_t

Definition at line 86 of file osapi-os-loader.h.

12.8 osal/src/os/inc/osapi-os-net.h File Reference

```
#include <osconfig.h>
```

Data Structures

- union OS_SockAddrData_t

  Storage buffer for generic network address.
- struct OS_SockAddr_t

  Encapsulates a generic network address.
- struct OS_socket_prop_t

  Encapsulates socket properties.

Macros

- #define OS_SOCKADDR_MAX_LEN 28

Enumerations

- enum OS_SocketDomain_t { OS_SocketDomain_INVALID, OS_SocketDomain_INET, OS_SocketDomain_IN↩ ET6, OS_SocketDomain_MAX }

  Socket domain.
- enum OS_SocketType_t { OS_SocketType_INVALID, OS_SocketType_DATAGRAM, OS_SocketType_STREAM, OS_SocketType_MAX }

  Socket type.

Functions

- int32 OS_SocketAddrInit (OS_SockAddr_t ∗Addr, OS_SocketDomain_t Domain)

  Initialize a socket address structure to hold an address of the given family.
- int32 OS_SocketAddrToString (char ∗buffer, uint32 buflen, const OS_SockAddr_t ∗Addr)

  Get a string representation of a network host address.
- int32 OS_SocketAddrFromString (OS_SockAddr_t ∗Addr, const char ∗string)

  Set a network host address from a string representation.
- int32 OS_SocketAddrGetPort (uint16 ∗PortNum, const OS_SockAddr_t ∗Addr)

  Get the port number of a network address.
- int32 OS_SocketAddrSetPort (OS_SockAddr_t ∗Addr, uint16 PortNum)

  Set the port number of a network address.
- int32 OS_SocketOpen (uint32 ∗sock_id, OS_SocketDomain_t Domain, OS_SocketType_t Type)

  Opens a socket.
- int32 OS_SocketBind (uint32 sock_id, const OS_SockAddr_t ∗Addr)

  Binds a socket to a given local address.
- int32 OS_SocketConnect (uint32 sock_id, const OS_SockAddr_t ∗Addr, int32 timeout)

  Connects a socket to a given remote address.
- int32 OS_SocketAccept (uint32 sock_id, uint32 ∗connsock_id, OS_SockAddr_t ∗Addr, int32 timeout)

  Waits for and accept the next incoming connection on the given socket.
- int32 OS_SocketRecvFrom (uint32 sock_id, void ∗buffer, uint32 buflen, OS_SockAddr_t ∗RemoteAddr, int32 timeout)

  Reads data from a message-oriented (datagram) socket.
- int32 OS_SocketSendTo (uint32 sock_id, const void ∗buffer, uint32 buflen, const OS_SockAddr_t ∗RemoteAddr)

Sends data to a message-oriented (datagram) socket.

- int32 OS_SocketGetIdByName (uint32 *sock_id, const char *sock_name)

  Gets an OSAL ID from a given name.

- int32 OS_SocketGetInfo (uint32 sock_id, OS_socket_prop_t *sock_prop)

  Gets information about an OSAL Socket ID.

- int32 OS_NetworkGetID (void)

  Gets the network ID of the local machine.

- int32 OS_NetworkGetHostName (char *host_name, uint32 name_len)

  Gets the local machine network host name.

12.8.1 Macro Definition Documentation

12.8.1.1 OS_SOCKADDR_MAX_LEN

```
#define OS_SOCKADDR_MAX_LEN 28
```

Definition at line 37 of file osapi-os-net.h.

12.8.2 Enumeration Type Documentation

12.8.2.1 OS_SocketDomain_t

```
enum OS_SocketDomain_t
```

Socket domain.

Enumerator

OS_SocketDomain_INVALID	Invalid.
OS_SocketDomain_INET	IPv4 address family, most commonly used)
OS_SocketDomain_INET6	IPv6 address family, depends on OS/network stack support.
OS_SocketDomain_MAX	Maximum.

Definition at line 53 of file osapi-os-net.h.

12.8.2.2 OS_SocketType_t

```
enum OS_SocketType_t
```

Socket type.

Enumerator

OS_SocketType_INVALID	Invalid.
OS_SocketType_DATAGRAM	A connectionless, message-oriented socket.
OS_SocketType_STREAM	A stream-oriented socket with the concept of a connection.
OS_SocketType_MAX	Maximum.

Definition at line 62 of file osapi-os-net.h.

12.9 osal/src/os/inc/osapi-os-timer.h File Reference

Data Structures

- struct OS_timer_prop_t
  Timer properties.
- struct OS_timebase_prop_t
  Time base properties.

Typedefs

- typedef void(∗ OS_TimerCallback_t) (uint32 timer_id)
  Timer callback.
- typedef uint32(∗ OS_TimerSync_t) (uint32 timer_id)
  Timer sync.

Functions

- int32 OS_TimeBaseCreate (uint32 ∗timebase_id, const char ∗timebase_name, OS_TimerSync_t external_sync)
  Create an abstract Time Base resource.
- int32 OS_TimeBaseSet (uint32 timebase_id, uint32 start_time, uint32 interval_time)
  Sets the tick period for simulated time base objects.
- int32 OS_TimeBaseDelete (uint32 timebase_id)
  Deletes a time base object.
- int32 OS_TimeBaseGetIdByName (uint32 ∗timebase_id, const char ∗timebase_name)
  Find the ID of an existing time base resource.
- int32 OS_TimeBaseGetInfo (uint32 timebase_id, OS_timebase_prop_t ∗timebase_prop)
  Obtain information about a timebase resource.
- int32 OS_TimeBaseGetFreeRun (uint32 timebase_id, uint32 ∗freerun_val)
  Read the value of the timebase free run counter.
- int32 OS_TimerCreate (uint32 ∗timer_id, const char ∗timer_name, uint32 ∗clock_accuracy, OS_TimerCallback↩ _t callback_ptr)
  Create a timer object.
- int32 OS_TimerAdd (uint32 ∗timer_id, const char ∗timer_name, uint32 timebase_id, OS_ArgCallback_↩ t callback_ptr, void ∗callback_arg)

Add a timer object based on an existing TimeBase resource.

- int32 OS_TimerSet (uint32 timer_id, uint32 start_time, uint32 interval_time)

  Configures a periodic or one shot timer.
- int32 OS_TimerDelete (uint32 timer_id)

  Deletes a timer resource.
- int32 OS_TimerGetIdByName (uint32 *timer_id, const char *timer_name)

  Locate an existing timer resource by name.
- int32 OS_TimerGetInfo (uint32 timer_id, OS_timer_prop_t *timer_prop)

  Gets information about an existing timer.

12.9.1 Typedef Documentation

12.9.1.1 OS_TimerCallback_t

```
typedef void(* OS_TimerCallback_t) (uint32 timer_id)
```

Timer callback.

Definition at line 25 of file osapi-os-timer.h.

12.9.1.2 OS_TimerSync_t

```
typedef uint32(* OS_TimerSync_t) (uint32 timer_id)
```

Timer sync.

Definition at line 26 of file osapi-os-timer.h.

12.10 osal/src/os/inc/osapi-version.h File Reference

Macros

- #define OS_MAJOR_VERSION 5

  Major version number.
- #define OS_MINOR_VERSION 0

  Minor version number.
- #define OS_REVISION 8

  Revision number.
- #define OS_MISSION_REV 0

  Mission revision.
- #define OSAL_API_VERSION ((OS_MAJOR_VERSION * 10000) + (OS_MINOR_VERSION * 100) + OS_RE↩ VISION)

12.10.1 Macro Definition Documentation

12.10.1.1 OS_MAJOR_VERSION

```
#define OS_MAJOR_VERSION 5
```

Major version number.

Definition at line 21 of file osapi-version.h.

12.10.1.2 OS_MINOR_VERSION

```
#define OS_MINOR_VERSION 0
```

Minor version number.

Definition at line 22 of file osapi-version.h.

12.10.1.3 OS_MISSION_REV

```
#define OS_MISSION_REV 0
```

Mission revision.

Definition at line 24 of file osapi-version.h.

12.10.1.4 OS_REVISION

```
#define OS_REVISION 8
```

Revision number.

Definition at line 23 of file osapi-version.h.

12.10.1.5 OSAL_API_VERSION

```
#define OSAL_API_VERSION ((OS_MAJOR_VERSION * 10000) + (OS_MINOR_VERSION * 100) + OS_REVISION)
```

Combine the revision components into a single value that application code can check against e.g. "#if OSAL_API_V↩ ERSION >= 40100" would check if some feature added in OSAL 4.1 is present.

Definition at line 30 of file osapi-version.h.

12.11 osal/src/os/inc/osapi.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "common_types.h"
#include "osapi-version.h"
#include "osconfig.h"
#include "osapi-os-core.h"
#include "osapi-os-filesys.h"
#include "osapi-os-net.h"
#include "osapi-os-loader.h"
#include "osapi-os-timer.h"
```

Macros

- #define OS_SUCCESS (0)

  Successful execution.
- #define OS_ERROR (-1)

  Failed execution.
- #define OS_INVALID_POINTER (-2)

  Invalid pointer.
- #define OS_ERROR_ADDRESS_MISALIGNED (-3)

  Address misalignment.
- #define OS_ERROR_TIMEOUT (-4)

  Error timeout.
- #define OS_INVALID_INT_NUM (-5)

  Invalid Interrupt number.
- #define OS_SEM_FAILURE (-6)

  Semaphore failure.
- #define OS_SEM_TIMEOUT (-7)

  Semaphore timeout.
- #define OS_QUEUE_EMPTY (-8)

  Queue empty.
- #define OS_QUEUE_FULL (-9)

  Queue full.
- #define OS_QUEUE_TIMEOUT (-10)

  Queue timeout.
- #define OS_QUEUE_INVALID_SIZE (-11)

  Queue invalid size.
- #define OS_QUEUE_ID_ERROR (-12)

  Queue ID error.
- #define OS_ERR_NAME_TOO_LONG (-13)

  Name too long.
- #define OS_ERR_NO_FREE_IDS (-14)

  No free IDs.
- #define OS_ERR_NAME_TAKEN (-15)

  Name taken.

- #define OS_ERR_INVALID_ID (-16)
  Invalid ID.
- #define OS_ERR_NAME_NOT_FOUND (-17)
  Name not found.
- #define OS_ERR_SEM_NOT_FULL (-18)
  Semaphore not full.
- #define OS_ERR_INVALID_PRIORITY (-19)
  Invalid priority.
- #define OS_INVALID_SEM_VALUE (-20)
  Invalid semaphore value.
- #define OS_ERR_FILE (-27)
  File error.
- #define OS_ERR_NOT_IMPLEMENTED (-28)
  Not implemented.
- #define OS_TIMER_ERR_INVALID_ARGS (-29)
  Timer invalid arguments.
- #define OS_TIMER_ERR_TIMER_ID (-30)
  Timer ID error.
- #define OS_TIMER_ERR_UNAVAILABLE (-31)
  Timer unavailable.
- #define OS_TIMER_ERR_INTERNAL (-32)
  Timer internal error.
- #define OS_ERR_OBJECT_IN_USE (-33)
  Object in use.
- #define OS_ERR_BAD_ADDRESS (-34)
  Bad address.
- #define OS_ERR_INCORRECT_OBJ_STATE (-35)
  Incorrect object state.
- #define OS_ERR_INCORRECT_OBJ_TYPE (-36)
  Incorrect object type.
- #define OS_ERR_STREAM_DISCONNECTED (-37)
  Stream disconnected.
- #define OS_PEND (-1)
- #define OS_CHECK (0)

12.11.1 Macro Definition Documentation

12.11.1.1 OS_CHECK

```
#define OS_CHECK (0)
```

Definition at line 86 of file osapi.h.

12.11.1.2 OS_PEND

```
#define OS_PEND (-1)
```

Definition at line 85 of file osapi.h.

Index