OSAL User's Guide

Generated by Doxygen 1.9.1

1 Osal API Documentation

- General Information and Concepts
  - OSAL Introduction
- Core
  - OSAL Return Code Defines
  - OSAL Object Type Defines
  - APIs
    - OSAL Core Operation APIs
    - OSAL Object ID Utility APIs
    - OSAL Task APIs

1.1 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.

1.2 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_↩ BASED 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.

1.3 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 the possible file system calls are not implemented. "Special" files requiring OS specific control/operations are by nature not portable. Abstraction in this case 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.

1.4 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.

2 Module Index

2.1 Modules

Here is a list of all modules:

3 Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

5 Module Documentation

5.1 OSAL Semaphore State Defines

Macros

- #define OS_SEM_FULL 1

  Semaphore full state.
- #define OS_SEM_EMPTY 0

  Semaphore empty state.

5.1.1 Detailed Description

5.1.2 Macro Definition Documentation

5.1.2.1 OS_SEM_EMPTY `#define OS_SEM_EMPTY 0`

Semaphore empty state.

Definition at line 35 of file osapi-binsem.h.

5.1.2.2 OS_SEM_FULL `#define OS_SEM_FULL 1`

Semaphore full state.

Definition at line 34 of file osapi-binsem.h.

5.2 OSAL Binary Semaphore APIs

Functions

- int32 OS_BinSemCreate (osal_id_t *sem_id, const char *sem_name, uint32 sem_initial_value, uint32 options)
  Creates a binary semaphore.
- int32 OS_BinSemFlush (osal_id_t sem_id)
  Unblock all tasks pending on the specified semaphore.
- int32 OS_BinSemGive (osal_id_t sem_id)
  Increment the semaphore value.
- int32 OS_BinSemTake (osal_id_t sem_id)
  Decrement the semaphore value.
- int32 OS_BinSemTimedWait (osal_id_t sem_id, uint32 msecs)
  Decrement the semaphore value with a timeout.
- int32 OS_BinSemDelete (osal_id_t sem_id)
  Deletes the specified Binary Semaphore.
- int32 OS_BinSemGetIdByName (osal_id_t *sem_id, const char *sem_name)
  Find an existing semaphore ID by name.
- int32 OS_BinSemGetInfo (osal_id_t sem_id, OS_bin_sem_prop_t *bin_prop)
  Fill a property object buffer with details regarding the resource.

5.2.1 Detailed Description

5.2.2 Function Documentation

5.2.2.1 OS_BinSemCreate()

```
int32 OS_BinSemCreate (
        osal_id_t * 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 non-zero ID of the newly-created resource (must not be null)
in	*sem_name*	the name of the new resource to create (must not be null)
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	name length including null terminator greater than OS_MAX_API_NAME
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 (return value only verified in coverage test)

5.2.2.2 OS_BinSemDelete()

```
int32 OS_BinSemDelete (
        osal_id_t 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	if an unspecified failure occurs (return value only verified in coverage test)

5.2.2.3 OS_BinSemFlush() `int32 OS_BinSemFlush ( osal_id_t 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 (return value only verified in coverage test)

5.2.2.4 OS_BinSemGetIdByName() `int32 OS_BinSemGetIdByName ( osal_id_t * 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 (must not be null)

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	name length including null terminator greater than OS_MAX_API_NAME
OS_ERR_NAME_NOT_FOUND	if the name was not found in the table

5.2.2.5 OS_BinSemGetInfo()

```
int32 OS_BinSemGetInfo (
        osal_id_t 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 (must not be null)

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
OS_ERR_NOT_IMPLEMENTED	Not implemented.

5.2.2.6 OS_BinSemGive() `int32 OS_BinSemGive ( osal_id_t 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_ERR_INVALID_ID	if the id passed in is not a binary semaphore
OS_SEM_FAILURE	if an unspecified failure occurs (return value only verified in coverage test)

5.2.2.7 OS_BinSemTake() `int32 OS_BinSemTake ( osal_id_t 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.

Return values

OS_ERR_INVALID_ID	the Id passed in is not a valid binary semaphore
OS_SEM_FAILURE	if an unspecified failure occurs (return value only verified in coverage test)

5.2.2.8 OS_BinSemTimedWait()

```
int32 OS_BinSemTimedWait (
    osal_id_t 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_ERR_INVALID_ID	if the ID passed in is not a valid semaphore ID
OS_SEM_FAILURE	if an unspecified failure occurs (return value only verified in coverage test)

5.3 OSAL BSP low level access APIs

Functions

- void OS_BSP_SetResourceTypeConfig (uint32 ResourceType, uint32 ConfigOptionValue)
- uint32 OS_BSP_GetResourceTypeConfig (uint32 ResourceType)
- uint32 OS_BSP_GetArgC (void)
- char ∗const ∗ OS_BSP_GetArgV (void)
- void OS_BSP_SetExitCode (int32 code)

5.3.1 Detailed Description

These are for OSAL internal BSP information access to pass any BSP-specific boot/command line/startup arguments through to the application, and return a status code back to the OS after exit.

Not intended for user application use

5.3.2 Function Documentation

5.3.2.1 OS_BSP_GetArgC()

```
uint32 OS_BSP_GetArgC (
    void  )
```

5.3.2.2 OS_BSP_GetArgV()

```
char* const* OS_BSP_GetArgV (
    void  )
```

5.3.2.3 OS_BSP_GetResourceTypeConfig()

```
uint32 OS_BSP_GetResourceTypeConfig (
    uint32 ResourceType )
```

5.3.2.4 OS_BSP_SetExitCode()

```
void OS_BSP_SetExitCode (
    int32 code )
```

5.3.2.5 OS_BSP_SetResourceTypeConfig()

```
void OS_BSP_SetResourceTypeConfig (
    uint32 ResourceType,
    uint32 ConfigOptionValue )
```

5.4 OSAL Real Time Clock APIs

Functions

- int32 OS_GetLocalTime (OS_time_t *time_struct)

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

  Set the local time.
- OS_time_t OS_TimeFromRelativeMilliseconds (int32 relative_msec)

  Gets an absolute time value relative to the current time.
- int32 OS_TimeToRelativeMilliseconds (OS_time_t time)

  Gets a relative time value from an absolute time.
- static int64 OS_TimeGetTotalSeconds (OS_time_t tm)

  Get interval from an OS_time_t object normalized to whole number of seconds.
- static OS_time_t OS_TimeFromTotalSeconds (int64 tm)

  Get an OS_time_t interval object from an integer number of seconds.
- static int64 OS_TimeGetTotalMilliseconds (OS_time_t tm)

  Get interval from an OS_time_t object normalized to millisecond units.
- static OS_time_t OS_TimeFromTotalMilliseconds (int64 tm)

  Get an OS_time_t interval object from a integer number of milliseconds.
- static int64 OS_TimeGetTotalMicroseconds (OS_time_t tm)

  Get interval from an OS_time_t object normalized to microsecond units.
- static OS_time_t OS_TimeFromTotalMicroseconds (int64 tm)

  Get an OS_time_t interval object from a integer number of microseconds.
- static int64 OS_TimeGetTotalNanoseconds (OS_time_t tm)

  Get interval from an OS_time_t object normalized to nanosecond units.
- static OS_time_t OS_TimeFromTotalNanoseconds (int64 tm)

  Get an OS_time_t interval object from a integer number of nanoseconds.
- static int64 OS_TimeGetFractionalPart (OS_time_t tm)

  Get subseconds portion (fractional part only) from an OS_time_t object.
- static uint32 OS_TimeGetSubsecondsPart (OS_time_t tm)

  Get 32-bit normalized subseconds (fractional part only) from an OS_time_t object.
- static uint32 OS_TimeGetMillisecondsPart (OS_time_t tm)

  Get milliseconds portion (fractional part only) from an OS_time_t object.
- static uint32 OS_TimeGetMicrosecondsPart (OS_time_t tm)

  Get microseconds portion (fractional part only) from an OS_time_t object.
- static uint32 OS_TimeGetNanosecondsPart (OS_time_t tm)

  Get nanoseconds portion (fractional part only) from an OS_time_t object.
- static OS_time_t OS_TimeAssembleFromNanoseconds (int64 seconds, uint32 nanoseconds)

  Assemble/Convert a number of seconds + nanoseconds into an OS_time_t interval.
- static OS_time_t OS_TimeAssembleFromMicroseconds (int64 seconds, uint32 microseconds)

  Assemble/Convert a number of seconds + microseconds into an OS_time_t interval.
- static OS_time_t OS_TimeAssembleFromMilliseconds (int64 seconds, uint32 milliseconds)

  Assemble/Convert a number of seconds + milliseconds into an OS_time_t interval.
- static OS_time_t OS_TimeAssembleFromSubseconds (int64 seconds, uint32 subseconds)

  Assemble/Convert a number of seconds + subseconds into an OS_time_t interval.
- static OS_time_t OS_TimeAdd (OS_time_t time1, OS_time_t time2)

Computes the sum of two time intervals.

- static OS_time_t OS_TimeSubtract (OS_time_t time1, OS_time_t time2)

  Computes the difference between two time intervals.
- static bool OS_TimeEqual (OS_time_t time1, OS_time_t time2)

  Checks if two time values are equal.
- static int8_t OS_TimeGetSign (OS_time_t time)

  Checks the sign of the time value.
- static int8_t OS_TimeCompare (OS_time_t time1, OS_time_t time2)

  Compares two time values.

5.4.1 Detailed Description

5.4.2 Function Documentation

5.4.2.1 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 (must not be null)

Returns

Get local time status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if time_struct is null

5.4.2.2 OS_SetLocalTime() `int32 OS_SetLocalTime ( const 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 (must not be null)

Returns

Set local time status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if time_struct is null

5.4.2.3 OS_TimeAdd()

```
static OS_time_t OS_TimeAdd (
        OS_time_t time1,
        OS_time_t time2 )  [inline], [static]
```

Computes the sum of two time intervals.

Parameters

in	*time1*	The first interval
in	*time2*	The second interval

Returns

The sum of the two intervals (time1 + time2)

Definition at line 530 of file osapi-clock.h.

References OS_time_t::ticks.

5.4.2.4 OS_TimeAssembleFromMicroseconds()

```
static OS_time_t OS_TimeAssembleFromMicroseconds (
        int64 seconds,
        uint32 microseconds )  [inline], [static]
```

Assemble/Convert a number of seconds + microseconds into an OS_time_t interval.

This creates an OS_time_t value using a whole number of seconds and a fractional part in units of microseconds. This is the inverse of OS_TimeGetTotalSeconds() and OS_TimeGetMicrosecondsPart(), and should recreate the original OS_time_t value from these separate values (aside from any potential conversion losses due to limited resolution of the data types/units).

See also

OS_TimeGetTotalSeconds(), OS_TimeGetMicrosecondsPart()

Parameters

in	*seconds*	Whole number of seconds
in	*microseconds*	Number of microseconds (fractional part only)

Returns

The input arguments represented as an OS_time_t interval

Definition at line 465 of file osapi-clock.h.

References OS_TIME_TICKS_PER_SECOND, OS_TIME_TICKS_PER_USEC, and OS_time_t::ticks.

5.4.2.5 OS_TimeAssembleFromMilliseconds()

```
static OS_time_t OS_TimeAssembleFromMilliseconds (
        int64 seconds,
        uint32 milliseconds )  [inline], [static]
```

Assemble/Convert a number of seconds + milliseconds into an OS_time_t interval.

This creates an OS_time_t value using a whole number of seconds and a fractional part in units of milliseconds. This is the inverse of OS_TimeGetTotalSeconds() and OS_TimeGetMillisecondsPart(), and should recreate the original OS_time_t value from these separate values (aside from any potential conversion losses due to limited resolution of the data types/units).

See also

OS_TimeGetTotalSeconds(), OS_TimeGetMillisecondsPart()

Parameters

in	*seconds*	Whole number of seconds
in	*milliseconds*	Number of milliseconds (fractional part only)

Returns

The input arguments represented as an OS_time_t interval

Definition at line 489 of file osapi-clock.h.

References OS_TIME_TICKS_PER_MSEC, OS_TIME_TICKS_PER_SECOND, and OS_time_t::ticks.

5.4.2.6 OS_TimeAssembleFromNanoseconds()

```
static OS_time_t OS_TimeAssembleFromNanoseconds (
        int64 seconds,
        uint32 nanoseconds )  [inline], [static]
```

Assemble/Convert a number of seconds + nanoseconds into an OS_time_t interval.

This creates an OS_time_t value using a whole number of seconds and a fractional part in units of nanoseconds. This is the inverse of OS_TimeGetTotalSeconds() and OS_TimeGetNanosecondsPart(), and should recreate the original OS_time_t value from these separate values (aside from any potential conversion losses due to limited resolution of the data types/units).

See also

OS_TimeGetTotalSeconds(), OS_TimeGetNanosecondsPart()

Parameters

in	*seconds*	Whole number of seconds
in	*nanoseconds*	Number of nanoseconds (fractional part only)

Returns

The input arguments represented as an OS_time_t interval

Definition at line 441 of file osapi-clock.h.

References OS_TIME_TICK_RESOLUTION_NS, OS_TIME_TICKS_PER_SECOND, and OS_time_t::ticks.

5.4.2.7 OS_TimeAssembleFromSubseconds()

```
static OS_time_t OS_TimeAssembleFromSubseconds (
        int64 seconds,
        uint32 subseconds )  [inline], [static]
```

Assemble/Convert a number of seconds + subseconds into an OS_time_t interval.

This creates an OS_time_t value using a whole number of seconds and a fractional part in units of sub-seconds (1/2^32). This is the inverse of OS_TimeGetTotalSeconds() and OS_TimeGetSubsecondsPart(), and should recreate the original OS_time_t value from these separate values (aside from any potential conversion losses due to limited resolution of the data types/units).

See also

OS_TimeGetTotalSeconds(), OS_TimeGetNanosecondsPart()

Parameters

in	*seconds*	Whole number of seconds
in	*subseconds*	Number of subseconds (32 bit fixed point fractional part)

Returns

The input arguments represented as an OS_time_t interval

Definition at line 512 of file osapi-clock.h.

References OS_TIME_TICKS_PER_SECOND, and OS_time_t::ticks.

5.4.2.8 OS_TimeCompare()

```
static int8_t OS_TimeCompare (
        OS_time_t time1,
        OS_time_t time2 )  [inline], [static]
```

Compares two time values.

Parameters

in	*time1*	The first time
in	*time2*	The second time

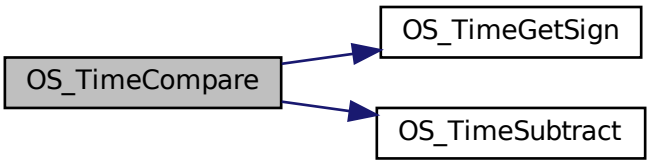
Return values

-1	if the time1 $<$ time2
0	if the times are equal
1	if the time1 $>$ time2

Definition at line 592 of file osapi-clock.h.

References OS_TimeGetSign(), and OS_TimeSubtract().

Here is the call graph for this function:



5.4.2.9 OS_TimeEqual() `static bool OS_TimeEqual (`

`OS_time_t time1,`
`OS_time_t time2 ) [inline], [static]`

Checks if two time values are equal.

Parameters

in	*time1*	The first time value
in	*time2*	The second time value

Return values

true	if the two values are equal
false	if the two values are not equal

Definition at line 561 of file osapi-clock.h.

References OS_time_t::ticks.

5.4.2.10 OS_TimeFromRelativeMilliseconds() `OS_time_t OS_TimeFromRelativeMilliseconds (`

`int32 relative_msec )`

Gets an absolute time value relative to the current time.

This function adds the given interval, expressed in milliseconds, to the current clock and returns the result.

Note

This is intended to ease transitioning from a relative timeout value to and absolute timeout value. The result can be passed to any function that accepts an absolute timeout, to mimic the behavior of a relative timeout.

Parameters

in	*relative_msec*	A relative time interval, in milliseconds

Returns

Absolute time value after adding interval

5.4.2.11 OS_TimeFromTotalMicroseconds() `static OS_time_t OS_TimeFromTotalMicroseconds ( int64 tm ) [inline], [static]`

Get an OS_time_t interval object from a integer number of microseconds.

This is the inverse operation of OS_TimeGetTotalMicroseconds(), converting the total number of microseconds into an OS_time_t value.

See also

OS_TimeGetTotalMicroseconds()

Parameters

in	*tm*	Time interval value, in microseconds

Returns

OS_time_t value representing the interval

Definition at line 278 of file osapi-clock.h.

References OS_TIME_TICKS_PER_USEC.

5.4.2.12 OS_TimeFromTotalMilliseconds() `static OS_time_t OS_TimeFromTotalMilliseconds ( int64 tm ) [inline], [static]`

Get an OS_time_t interval object from a integer number of milliseconds.

This is the inverse operation of OS_TimeGetTotalMilliseconds(), converting the total number of milliseconds into an OS_time_t value.

See also

OS_TimeGetTotalMilliseconds()

Parameters

in	*tm*	Time interval value, in milliseconds

Returns

OS_time_t value representing the interval

Definition at line 244 of file osapi-clock.h.

References OS_TIME_TICKS_PER_MSEC.

5.4.2.13 OS_TimeFromTotalNanoseconds() `static OS_time_t OS_TimeFromTotalNanoseconds ( int64 tm ) [inline], [static]`

Get an OS_time_t interval object from a integer number of nanoseconds.

This is the inverse operation of OS_TimeGetTotalNanoseconds(), converting the total number of nanoseconds into an OS_time_t value.

See also

OS_TimeGetTotalNanoseconds()

Parameters

in	*tm*	Time interval value, in nanoseconds

Returns

OS_time_t value representing the interval

Definition at line 317 of file osapi-clock.h.

References OS_TIME_TICK_RESOLUTION_NS.

5.4.2.14 OS_TimeFromTotalSeconds() `static OS_time_t OS_TimeFromTotalSeconds ( int64 tm ) [inline], [static]`

Get an OS_time_t interval object from an integer number of seconds.

This is the inverse operation of OS_TimeGetTotalSeconds(), converting the total number of seconds into an OS_time_t value.

See also

OS_TimeGetTotalSeconds()

Parameters

in	*tm*	Time interval value, in seconds

Returns

OS_time_t value representing the interval

Definition at line 210 of file osapi-clock.h.

References OS_TIME_TICKS_PER_SECOND.

5.4.2.15 OS_TimeGetFractionalPart()

```
static int64 OS_TimeGetFractionalPart (
            OS_time_t tm ) [inline], [static]
```

Get subseconds portion (fractional part only) from an OS_time_t object.

Extracts the fractional part from a given OS_time_t object. Units returned are in ticks, not normalized to any standard time unit.

Parameters

in	*tm*	Time interval value

Returns

Fractional/subsecond portion of time interval in ticks

Definition at line 333 of file osapi-clock.h.

References OS_TIME_TICKS_PER_SECOND, and OS_time_t::ticks.

Referenced by OS_TimeGetMicrosecondsPart(), OS_TimeGetMillisecondsPart(), OS_TimeGetNanosecondsPart(), and OS_TimeGetSubsecondsPart().

5.4.2.16 OS_TimeGetMicrosecondsPart()

```
static uint32 OS_TimeGetMicrosecondsPart (
            OS_time_t tm ) [inline], [static]
```

Get microseconds portion (fractional part only) from an OS_time_t object.

Extracts the fractional part from a given OS_time_t object normalized to units of microseconds.

This function may be used to adapt applications initially implemented using an older OSAL version where OS_time_t was a structure containing a "seconds" and "microsecs" field.

This function will obtain a value that is compatible with the "microsecs" field of OS_time_t as it was defined in previous versions of OSAL, as well as the "tv_usec" field of POSIX-style "struct timeval" values.

See also

OS_TimeGetTotalSeconds()

Parameters

in	*tm*	Time interval value

Returns

Number of microseconds in time interval

Definition at line 401 of file osapi-clock.h.

References OS_TIME_TICKS_PER_USEC, and OS_TimeGetFractionalPart().

Here is the call graph for this function:

5.4.2.17 OS_TimeGetMillisecondsPart()

```
static uint32 OS_TimeGetMillisecondsPart (
        OS_time_t tm ) [inline], [static]
```

Get milliseconds portion (fractional part only) from an OS_time_t object.

Extracts the fractional part from a given OS_time_t object normalized to units of milliseconds.

See also

OS_TimeGetTotalSeconds()

Parameters

in	*tm*	Time interval value

Returns

Number of milliseconds in time interval

Definition at line 376 of file osapi-clock.h.

References OS_TIME_TICKS_PER_MSEC, and OS_TimeGetFractionalPart().

Here is the call graph for this function:

5.4.2.18 OS_TimeGetNanosecondsPart() `static uint32 OS_TimeGetNanosecondsPart ( OS_time_t tm ) [inline], [static]`

Get nanoseconds portion (fractional part only) from an OS_time_t object.

Extracts the only number of nanoseconds from a given OS_time_t object.

This function will obtain a value that is compatible with the "tv_nsec" field of POSIX-style "struct timespec" values.

See also

OS_TimeGetTotalSeconds()

Parameters

in	*tm*	Time interval value

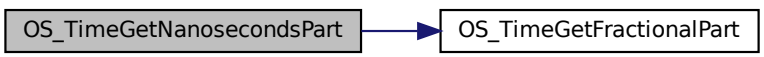
Returns

Number of nanoseconds in time interval

Definition at line 420 of file osapi-clock.h.

References OS_TIME_TICK_RESOLUTION_NS, and OS_TimeGetFractionalPart().

Here is the call graph for this function:



5.4.2.19 OS_TimeGetSign() `static int8_t OS_TimeGetSign ( OS_time_t time ) [inline], [static]`

Checks the sign of the time value.

Parameters

in	*time*	The time to check

Return values

-1	if the time value is negative / below 0
0	if the time value is 0
1	if the time value is positive / above 0

Definition at line 576 of file osapi-clock.h.

References OS_time_t::ticks.

Referenced by OS_TimeCompare().

5.4.2.20 OS_TimeGetSubsecondsPart() `static uint32 OS_TimeGetSubsecondsPart ( OS_time_t tm ) [inline], [static]`

Get 32-bit normalized subseconds (fractional part only) from an OS_time_t object.

Extracts the fractional part from a given OS_time_t object in maximum precision, with units of 2^(-32) sec. This is a base-2 fixed-point fractional value with the point left-justified in the 32-bit value (i.e. left of MSB).

This is (mostly) compatible with the CFE "subseconds" value, where 0x80000000 represents exactly one half second, and 0 represents a full second.

Parameters

in	*tm*	Time interval value

Returns

Fractional/subsecond portion of time interval as 32-bit fixed point value

Definition at line 352 of file osapi-clock.h.

References OS_TIME_TICKS_PER_SECOND, and OS_TimeGetFractionalPart().

Here is the call graph for this function:

5.4.2.21 OS_TimeGetTotalMicroseconds() `static int64 OS_TimeGetTotalMicroseconds ( OS_time_t tm ) [inline], [static]`

Get interval from an OS_time_t object normalized to microsecond units.

Note this refers to the complete interval, not just the fractional part.

See also

OS_TimeFromTotalMicroseconds()

Parameters

in	*tm*	Time interval value

Returns

Whole number of microseconds in time interval

Definition at line 261 of file osapi-clock.h.

References OS_TIME_TICKS_PER_USEC, and OS_time_t::ticks.

5.4.2.22 OS_TimeGetTotalMilliseconds() `static int64 OS_TimeGetTotalMilliseconds ( OS_time_t tm ) [inline], [static]`

Get interval from an OS_time_t object normalized to millisecond units.

Note this refers to the complete interval, not just the fractional part.

See also

OS_TimeFromTotalMilliseconds()

Parameters

in	*tm*	Time interval value

Returns

Whole number of milliseconds in time interval

Definition at line 227 of file osapi-clock.h.

References OS_TIME_TICKS_PER_MSEC, and OS_time_t::ticks.

5.4.2.23 OS_TimeGetTotalNanoseconds()

```
static int64 OS_TimeGetTotalNanoseconds (
        OS_time_t tm )  [inline], [static]
```

Get interval from an OS_time_t object normalized to nanosecond units.

Note this refers to the complete interval, not just the fractional part.

Note

There is no protection against overflow of the 64-bit return value. Applications must use caution to ensure that the interval does not exceed the representable range of a signed 64 bit integer - approximately 140 years.

See also

OS_TimeFromTotalNanoseconds

Parameters

in	*tm*	Time interval value

Returns

Whole number of microseconds in time interval

Definition at line 300 of file osapi-clock.h.

References OS_TIME_TICK_RESOLUTION_NS, and OS_time_t::ticks.

5.4.2.24 OS_TimeGetTotalSeconds() `static int64 OS_TimeGetTotalSeconds ( OS_time_t tm ) [inline], [static]`

Get interval from an OS_time_t object normalized to whole number of seconds.

Extracts the number of whole seconds from a given OS_time_t object, discarding any fractional component.

This may also replace a direct read of the "seconds" field from the OS_time_t object from previous versions of OSAL, where the structure was defined with separate seconds/microseconds fields.

See also

OS_TimeGetMicrosecondsPart()

OS_TimeFromTotalSeconds()

Parameters

in	*tm*	Time interval value

Returns

Whole number of seconds in time interval

Definition at line 193 of file osapi-clock.h.

References OS_TIME_TICKS_PER_SECOND, and OS_time_t::ticks.

5.4.2.25 OS_TimeSubtract() `static OS_time_t OS_TimeSubtract ( OS_time_t time1, OS_time_t time2 ) [inline], [static]`

Computes the difference between two time intervals.

Parameters

in	*time1*	The first interval
in	*time2*	The second interval

Returns

The difference of the two intervals (time1 - time2)

Definition at line 545 of file osapi-clock.h.

References OS_time_t::ticks.

Referenced by OS_TimeCompare().

5.4.2.26 OS_TimeToRelativeMilliseconds() int32 OS_TimeToRelativeMilliseconds (OS_time_t *time*)

Gets a relative time value from an absolute time.

This function computes the number of milliseconds until the given absolute time value is reached in the system clock.

Note

This is intended to ease transitioning from a relative timeout value to and absolute timeout value. The result can be passed to any function that accepts a relative timeout, to mimic the behavior of an absolute timeout.

The return value of this function is intended to be compatible with the relative timeout parameter of various OSAL APIs e.g. OS_TimedRead() / OS_TimedWrite()

Parameters

in	*time*	An absolute time value

Returns

Milliseconds until time value will be reached

Return values

OS_CHECK	(0) if time is the current time or is in the past
OS_PEND	(-1) if time is far in the future (not expressable as an int32)

5.5 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_API_Teardown (void)

  Teardown/de-initialization of OSAL 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.

- int32 OS_RegisterEventHandler (OS_EventHandler_t handler)

  Callback routine registration.

- size_t OS_strnlen (const char *s, size_t maxlen)

  get string length

5.5.1 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

5.5.2 Function Documentation

5.5.2.1 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. (return value only verified in coverage test)

5.5.2.2 OS_API_Teardown() `void OS_API_Teardown ( void )`

Teardown/de-initialization of OSAL API.

This is the inverse of OS_API_Init(). It will release all OS resources and return the system to a state similar to what it was prior to invoking OS_API_Init() initially.

Normally for embedded applications, the OSAL is initialized after boot and will remain initialized in memory until the processor is rebooted. However for testing and development purposes, it is potentially useful to reset back to initial conditions.

For testing purposes, this API is designed/intended to be compatible with the UtTest_AddTeardown() routine provided by the UT-Assert subsystem.

Note

This is a "best-effort" routine and it may not always be possible/guaranteed to recover all resources, particularly in the case of off-nominal conditions, or if a resource is used outside of OSAL.

For example, while this will attempt to unload all dynamically-loaded modules, doing so may not be possible and/or may induce undefined behavior if resources are in use by tasks/functions outside of OSAL.

5.5.2.3 OS_Application_Run() `void OS_Application_Run ( void )`

Application run.

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

5.5.2.4 OS_Application_Startup() `void OS_Application_Startup ( void )`

Application startup.

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

5.5.2.5 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.

5.5.2.6 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

5.5.2.7 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.

5.5.2.8 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.

5.5.2.9 OS_RegisterEventHandler() `int32 OS_RegisterEventHandler ( OS_EventHandler_t handler )`

Callback routine registration.

This hook enables the application code to perform extra platform-specific operations on various system events such as resource creation/deletion.

Note

Some events are invoked while the resource is "locked" and therefore application-defined handlers for these events should not block or attempt to access other OSAL resources.

Parameters

in	*handler*	The application-provided event handler (must not be null)

Returns

Execution status, see OSAL Return Code Defines.

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if handler is NULL

5.5.2.10 OS_strnlen()

```
size_t OS_strnlen (
    const char * s,
    size_t maxlen )
```

get string length

Provides an OSAL routine to get the functionality of the (non-C99) "strnlen()" function, via the C89/C99 standard "memchr()" function instead.

Parameters

in	*s*	The input string
in	*maxlen*	Maximum length to check

Return values

Length	of the string or maxlen, whichever is smaller.

5.6 OSAL Condition Variable APIs

Functions

- int32 OS_CondVarCreate (osal_id_t *var_id, const char *var_name, uint32 options)

  Creates a condition variable resource.
- int32 OS_CondVarLock (osal_id_t var_id)

  Locks/Acquires the underlying mutex associated with a condition variable.
- int32 OS_CondVarUnlock (osal_id_t var_id)

  Unlocks/Releases the underlying mutex associated with a condition variable.
- int32 OS_CondVarSignal (osal_id_t var_id)

  Signals the condition variable resource referenced by var_id.
- int32 OS_CondVarBroadcast (osal_id_t var_id)

  Broadcasts the condition variable resource referenced by var_id.
- int32 OS_CondVarWait (osal_id_t var_id)

  Waits on the condition variable object referenced by var_id.
- int32 OS_CondVarTimedWait (osal_id_t var_id, const OS_time_t *abs_wakeup_time)

  Time-limited wait on the condition variable object referenced by var_id.
- int32 OS_CondVarDelete (osal_id_t var_id)

Deletes the specified condition variable.

- int32 OS_CondVarGetIdByName (osal_id_t *var_id, const char *var_name)

  Find an existing condition variable ID by name.

- int32 OS_CondVarGetInfo (osal_id_t var_id, OS_condvar_prop_t *condvar_prop)

  Fill a property object buffer with details regarding the resource.

5.6.1 Detailed Description

5.6.2 Function Documentation

5.6.2.1 OS_CondVarBroadcast() int32 OS_CondVarBroadcast (osal_id_t *var_id*)

Broadcasts the condition variable resource referenced by var_id.

This function may be used to indicate when the state of a data object has been changed.

If there are threads blocked on the condition variable object referenced by var_id when this function is called, all threads will be unblocked.

Note that although all threads are unblocked, because the mutex is re-acquired before the wait function returns, only a single task will be testing the condition at a given time. The order with which each blocked task runs is determined by the scheduling policy.

Parameters

in	*var_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 condition variable

5.6.2.2 OS_CondVarCreate() int32 OS_CondVarCreate (osal_id_t * *var_id*,

```
const char * var_name,
uint32 options )
```

Creates a condition variable resource.

A condition variable adds a more sophisticated synchronization option for mutexes, such that it can operate on arbitrary user-defined conditions rather than simply a counter or boolean (as in the case of simple semaphores).

Creating a condition variable resource in OSAL will in turn create both a basic mutex as well as a synchronization overlay. The underlying mutex is similar to the mutex functionality provided by the OSAL mutex subsystem, and can be locked and unlocked normally.

This mutex is intended to protect access to any arbitrary user-defined data object that serves as the condition being tested.

A task that needs a particular state of the object should follow this general flow:

- Lock the underlying mutex
- Test for the condition being waited for (a user-defined check on user-defined data)
- If condition IS NOT met, then call OS_CondVarWait() to wait, then repeat test
- If condition IS met, then unlock the underlying mutex and continue

A task that changes the state of the object should follow this general flow:

- Lock the underlying mutex
- Change the state as necessary
- Call either OS_CondVarSignal() or OS_CondVarBroadcast()
- Unlock the underlying mutex

Parameters

out	*var_id*	will be set to the non-zero ID of the newly-created resource (must not be null)
in	*var_name*	the name of the new resource to create (must not be null)
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 var_id or var_name are NULL
OS_ERR_NAME_TOO_LONG	name length including null terminator greater than OS_MAX_API_NAME
OS_ERR_NO_FREE_IDS	if there are no more free condition variable Ids
OS_ERR_NAME_TAKEN	if there is already a condition variable with the same name

5.6.2.3 OS_CondVarDelete() int32 OS_CondVarDelete (osal_id_t var_id)

Deletes the specified condition variable.

Delete the condition variable and releases any related system resources.

Parameters

in	*var_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 condvar

5.6.2.4 OS_CondVarGetIdByName() int32 OS_CondVarGetIdByName (osal_id_t * var_id, const char * var_name)

Find an existing condition variable ID by name.

This function tries to find an existing condition variable ID given the name. The id is returned through var_id.

Parameters

out	*var_id*	will be set to the ID of the existing resource
in	*var_name*	the name of the existing resource to find (must not be null)

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	is var_id or var_name are NULL pointers

Return values

OS_ERR_NAME_TOO_LONG	name length including null terminator greater than OS_MAX_API_NAME
OS_ERR_NAME_NOT_FOUND	if the name was not found in the table

5.6.2.5 OS_CondVarGetInfo()

```
int32 OS_CondVarGetInfo (
        osal_id_t var_id,
        OS_condvar_prop_t * condvar_prop )
```

Fill a property object buffer with details regarding the resource.

This function will fill a structure to contain the information (name and creator) about the specified condition variable.

Parameters

in	*var_id*	The object ID to operate on
out	*condvar_prop*	The property object buffer to fill (must not be null)

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

5.6.2.6 OS_CondVarLock()

```
int32 OS_CondVarLock (
        osal_id_t var_id )
```

Locks/Acquires the underlying mutex associated with a condition variable.

The mutex should always be locked by a task before reading or modifying the data object associated with a condition variable.

Note

This lock must be acquired by a task before invoking OS_CondVarWait() or OS_CondVarTimedWait() on the same condition variable.

Parameters

in	*var_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 condition variable

5.6.2.7 OS_CondVarSignal()

```
int32 OS_CondVarSignal (
        osal_id_t var_id )
```

Signals the condition variable resource referenced by var_id.

This function may be used to indicate when the state of a data object has been changed.

If there are threads blocked on the condition variable object referenced by var_id when this function is called, one of those threads will be unblocked, as determined by the scheduling policy.

Parameters

in	*var_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 condition variable

5.6.2.8 OS_CondVarTimedWait()

```
int32 OS_CondVarTimedWait (
        osal_id_t var_id,
        const OS_time_t * abs_wakeup_time )
```

Time-limited wait on the condition variable object referenced by var_id.

Identical in operation to OS_CondVarWait(), except that the maximum amount of time that the task will be blocked is limited.

The abs_wakeup_time refers to the absolute time of the system clock at which the task should be unblocked to run, regardless of the state of the condition variable. This refers to the same system clock that is the subject of the OS_GetLocalTime() API.

Parameters

in	*var_id*	The object ID to operate on
in	*abs_wakeup_time*	The system time at which the task should be unblocked (must not be null)

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 condvar

5.6.2.9 OS_CondVarUnlock() `int32 OS_CondVarUnlock ( osal_id_t var_id )`

Unlocks/Releases the underlying mutex associated with a condition variable.

The mutex should be unlocked by a task once reading or modifying the data object associated with a condition variable is complete.

Parameters

in	*var_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 condition variable

5.6.2.10 OS_CondVarWait() int32 OS_CondVarWait (osal_id_t *var_id*)

Waits on the condition variable object referenced by var_id.

The calling task will be blocked until another task calls the function OS_CondVarSignal() or OS_CondVarBroadcast() on the same condition variable.

The underlying mutex associated with the condition variable must be locked and owned by the calling task at the time this function is invoked. As part of this call, the mutex will be unlocked as the task blocks. This is done in such a way that there is no possibility that another task could aquire the mutex before the calling task has actually blocked.

This atomicity with respect to blocking the task and unlocking the mutex is a critical difference between condition variables and other synchronization primitives. It avoids a window of opportunity where inherent in the simpler synchronization resource types where the state of the data could change between the time that the calling task tested the state and the time that the task actually blocks on the sync resource.

Parameters

in	*var_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 condvar

5.7 OSAL Counting Semaphore APIs

Functions

- int32 OS_CountSemCreate (osal_id_t *sem_id, const char *sem_name, uint32 sem_initial_value, uint32 options)
  Creates a counting semaphore.
- int32 OS_CountSemGive (osal_id_t sem_id)
  Increment the semaphore value.
- int32 OS_CountSemTake (osal_id_t sem_id)
  Decrement the semaphore value.
- int32 OS_CountSemTimedWait (osal_id_t sem_id, uint32 msecs)
  Decrement the semaphore value with timeout.
- int32 OS_CountSemDelete (osal_id_t sem_id)

Deletes the specified counting Semaphore.

- int32 OS_CountSemGetIdByName (osal_id_t *sem_id, const char *sem_name)

  Find an existing semaphore ID by name.

- int32 OS_CountSemGetInfo (osal_id_t sem_id, OS_count_sem_prop_t *count_prop)

  Fill a property object buffer with details regarding the resource.

5.7.1 Detailed Description

5.7.2 Function Documentation

5.7.2.1 OS_CountSemCreate()

```
int32 OS_CountSemCreate (
        osal_id_t * 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.

Note

Underlying RTOS implementations may or may not impose a specific upper limit to the value of a counting semaphore. If the OS has a specific limit and the sem_initial_value exceeds this limit, then OS_INVALID_SEM_VALUE is returned. On other implementations, any 32-bit integer value may be acceptable. For maximum portability, it is recommended to keep counting semaphore values within the range of a "short int" (i.e. between 0 and 32767). Many platforms do accept larger values, but may not be guaranteed.

Parameters

out	*sem_id*	will be set to the non-zero ID of the newly-created resource (must not be null)
in	*sem_name*	the name of the new resource to create (must not be null)
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

Return values

OS_ERR_NAME_TOO_LONG	name length including null terminator greater than OS_MAX_API_NAME
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_INVALID_SEM_VALUE	if the semaphore value is too high (return value only verified in coverage test)
OS_SEM_FAILURE	if an unspecified implementation error occurs (return value only verified in coverage test)

5.7.2.2 OS_CountSemDelete()

```
int32 OS_CountSemDelete (
        osal_id_t 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	if an unspecified implementation error occurs (return value only verified in coverage test)

5.7.2.3 OS_CountSemGetIdByName()

```
int32 OS_CountSemGetIdByName (
        osal_id_t * 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 (must not be null)

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	name length including null terminator greater than OS_MAX_API_NAME
OS_ERR_NAME_NOT_FOUND	if the name was not found in the table

5.7.2.4 OS_CountSemGetInfo()

```
int32 OS_CountSemGetInfo (
        osal_id_t 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 (must not be null)

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
OS_ERR_NOT_IMPLEMENTED	Not implemented.

5.7.2.5 OS_CountSemGive()

```
int32 OS_CountSemGive (
        osal_id_t 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_ERR_INVALID_ID	if the id passed in is not a counting semaphore
OS_SEM_FAILURE	if an unspecified implementation error occurs (return value only verified in coverage test)

5.7.2.6 OS_CountSemTake() `int32 OS_CountSemTake ( osal_id_t 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 an unspecified implementation error occurs (return value only verified in coverage test)

5.7.2.7 OS_CountSemTimedWait() `int32 OS_CountSemTimedWait (`

```
osal_id_t 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_ERR_INVALID_ID	if the ID passed in is not a valid semaphore ID
OS_SEM_FAILURE	if an unspecified implementation error occurs (return value only verified in coverage test)

5.8 OSAL Directory APIs

Functions

- int32 OS_DirectoryOpen (osal_id_t *dir_id, const char *path)
  Opens a directory.
- int32 OS_DirectoryClose (osal_id_t dir_id)
  Closes an open directory.
- int32 OS_DirectoryRewind (osal_id_t dir_id)
  Rewinds an open directory.
- int32 OS_DirectoryRead (osal_id_t 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.

5.8.1 Detailed Description

5.8.2 Function Documentation

5.8.2.1 OS_DirectoryClose() `int32 OS_DirectoryClose ( osal_id_t 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

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the directory handle is invalid

5.8.2.2 OS_DirectoryOpen() `int32 OS_DirectoryOpen ( osal_id_t * dir_id, const char * path )`

Opens a directory.

Prepares for reading the files within a directory

Parameters

out	*dir_id*	Location to store handle ID of the directory (must not be null)
in	*path*	The directory to open (must not be null)

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if dir_id or path is NULL
OS_FS_ERR_PATH_TOO_LONG	if the path argument exceeds the maximum length
OS_FS_ERR_PATH_INVALID	if the path argument is not valid
OS_ERROR	if the directory could not be opened

5.8.2.3 OS_DirectoryRead()

```
int32 OS_DirectoryRead (
        osal_id_t 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 (must not be null)

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if dirent argument is NULL
OS_ERR_INVALID_ID	if the directory handle is invalid
OS_ERROR	at the end of the directory or if the OS call otherwise fails

5.8.2.4 OS_DirectoryRewind()

```
int32 OS_DirectoryRewind (
        osal_id_t 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

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the directory handle is invalid

5.8.2.5 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 (must not be null)
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 (return value only verified in coverage test)

5.8.2.6 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 (return value only verified in coverage test)

5.9 OSAL Return Code Defines

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 length including null terminator greater than OS_MAX_API_NAME
- #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_ERR_OPERATION_NOT_SUPPORTED (-38)

  Requested operation not support on supplied object(s)
- #define OS_ERR_INVALID_SIZE (-40)

  Invalid Size.
- #define OS_ERR_OUTPUT_TOO_LARGE (-41)

  Size of output exceeds limit

- #define OS_ERR_INVALID_ARGUMENT (-42)

  Invalid argument value (other than ID or size)
- #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.

5.9.1 Detailed Description

The specific status/return code definitions listed in this section may be extended or refined in future versions of OSAL.

Note

Application developers should assume that any OSAL API may return any status value listed here. While the documentation of each OSAL API function indicates the return/status values that function may directly generate, functions may also pass through other status codes from related functions, so that list should not be considered absolute/exhaustive.

The `int32` data type should be used to store an OSAL status code. Negative values will always represent errors, while non-negative values indicate success. Most APIs specifically return OS_SUCCESS (0) upon successful execution, but some return a nonzero value, such as data size.

Ideally, in order to more easily adapt to future OSAL versions and status code extensions/refinements, applications should typically check for errors as follows:

```
int32 status;
status = OS_TaskCreate(...);  (or any other API)
if (status < OS_SUCCESS)
{
    handle or report error....
    may also check for specific codes here.
}
else
{
    handle normal/successful status...
}
```

5.9.2 Macro Definition Documentation

5.9.2.1 OS_ERR_BAD_ADDRESS `#define OS_ERR_BAD_ADDRESS (-34)`

Bad address.

Definition at line 124 of file osapi-error.h.

5.9.2.2 OS_ERR_FILE `#define OS_ERR_FILE (-27)`

File error.

Definition at line 117 of file osapi-error.h.

5.9.2.3 OS_ERR_INCORRECT_OBJ_STATE `#define OS_ERR_INCORRECT_OBJ_STATE (-35)`

Incorrect object state.

Definition at line 125 of file osapi-error.h.

5.9.2.4 OS_ERR_INCORRECT_OBJ_TYPE `#define OS_ERR_INCORRECT_OBJ_TYPE (-36)`

Incorrect object type.

Definition at line 126 of file osapi-error.h.

5.9.2.5 OS_ERR_INVALID_ARGUMENT `#define OS_ERR_INVALID_ARGUMENT (-42)`

Invalid argument value (other than ID or size)

Definition at line 131 of file osapi-error.h.

5.9.2.6 OS_ERR_INVALID_ID `#define OS_ERR_INVALID_ID (-16)`

Invalid ID.

Definition at line 112 of file osapi-error.h.

5.9.2.7 OS_ERR_INVALID_PRIORITY `#define OS_ERR_INVALID_PRIORITY (-19)`

Invalid priority.

Definition at line 115 of file osapi-error.h.

5.9.2.8 OS_ERR_INVALID_SIZE `#define OS_ERR_INVALID_SIZE (-40)`

Invalid Size.

Definition at line 129 of file osapi-error.h.

5.9.2.9 OS_ERR_NAME_NOT_FOUND `#define OS_ERR_NAME_NOT_FOUND (-17)`

Name not found.

Definition at line 113 of file osapi-error.h.

5.9.2.10 OS_ERR_NAME_TAKEN `#define OS_ERR_NAME_TAKEN (-15)`

Name taken.

Definition at line 111 of file osapi-error.h.

5.9.2.11 OS_ERR_NAME_TOO_LONG `#define OS_ERR_NAME_TOO_LONG (-13)`

name length including null terminator greater than OS_MAX_API_NAME

Definition at line 109 of file osapi-error.h.

5.9.2.12 OS_ERR_NO_FREE_IDS `#define OS_ERR_NO_FREE_IDS (-14)`

No free IDs.

Definition at line 110 of file osapi-error.h.

5.9.2.13 OS_ERR_NOT_IMPLEMENTED `#define OS_ERR_NOT_IMPLEMENTED (-28)`

Not implemented.

Definition at line 118 of file osapi-error.h.

5.9.2.14 OS_ERR_OBJECT_IN_USE `#define OS_ERR_OBJECT_IN_USE (-33)`

Object in use.

Definition at line 123 of file osapi-error.h.

5.9.2.15 OS_ERR_OPERATION_NOT_SUPPORTED `#define OS_ERR_OPERATION_NOT_SUPPORTED (-38)`

Requested operation not support on supplied object(s)

Definition at line 128 of file osapi-error.h.

5.9.2.16 OS_ERR_OUTPUT_TOO_LARGE `#define OS_ERR_OUTPUT_TOO_LARGE (-41)`

Size of output exceeds limit

Definition at line 130 of file osapi-error.h.

5.9.2.17 OS_ERR_SEM_NOT_FULL `#define OS_ERR_SEM_NOT_FULL (-18)`

Semaphore not full.

Definition at line 114 of file osapi-error.h.

5.9.2.18 OS_ERR_STREAM_DISCONNECTED `#define OS_ERR_STREAM_DISCONNECTED (-37)`

Stream disconnected.

Definition at line 127 of file osapi-error.h.

5.9.2.19 OS_ERROR `#define OS_ERROR (-1)`

Failed execution.

Definition at line 97 of file osapi-error.h.

5.9.2.20 OS_ERROR_ADDRESS_MISALIGNED `#define OS_ERROR_ADDRESS_MISALIGNED (-3)`

Address misalignment.

Definition at line 99 of file osapi-error.h.

5.9.2.21 OS_ERROR_TIMEOUT `#define OS_ERROR_TIMEOUT (-4)`

Error timeout.

Definition at line 100 of file osapi-error.h.

5.9.2.22 OS_FS_ERR_DEVICE_NOT_FREE `#define OS_FS_ERR_DEVICE_NOT_FREE (-107)`

FS device not free.

Definition at line 144 of file osapi-error.h.

5.9.2.23 OS_FS_ERR_DRIVE_NOT_CREATED `#define OS_FS_ERR_DRIVE_NOT_CREATED (-106)`

FS drive not created.

Definition at line 143 of file osapi-error.h.

5.9.2.24 OS_FS_ERR_NAME_TOO_LONG `#define OS_FS_ERR_NAME_TOO_LONG (-104)`

FS name too long.

Definition at line 142 of file osapi-error.h.

5.9.2.25 OS_FS_ERR_PATH_INVALID `#define OS_FS_ERR_PATH_INVALID (-108)`

FS path invalid.

Definition at line 145 of file osapi-error.h.

5.9.2.26 OS_FS_ERR_PATH_TOO_LONG `#define OS_FS_ERR_PATH_TOO_LONG (-103)`

FS path too long.

Definition at line 141 of file osapi-error.h.

5.9.2.27 OS_INVALID_INT_NUM `#define OS_INVALID_INT_NUM (-5)`

Invalid Interrupt number.

Definition at line 101 of file osapi-error.h.

5.9.2.28 OS_INVALID_POINTER `#define OS_INVALID_POINTER (-2)`

Invalid pointer.

Definition at line 98 of file osapi-error.h.

5.9.2.29 OS_INVALID_SEM_VALUE `#define OS_INVALID_SEM_VALUE (-20)`

Invalid semaphore value.

Definition at line 116 of file osapi-error.h.

5.9.2.30 OS_QUEUE_EMPTY `#define OS_QUEUE_EMPTY (-8)`

Queue empty.

Definition at line 104 of file osapi-error.h.

5.9.2.31 OS_QUEUE_FULL `#define OS_QUEUE_FULL (-9)`

Queue full.

Definition at line 105 of file osapi-error.h.

5.9.2.32 OS_QUEUE_ID_ERROR `#define OS_QUEUE_ID_ERROR (-12)`

Queue ID error.

Definition at line 108 of file osapi-error.h.

5.9.2.33 OS_QUEUE_INVALID_SIZE `#define OS_QUEUE_INVALID_SIZE (-11)`

Queue invalid size.

Definition at line 107 of file osapi-error.h.

5.9.2.34 OS_QUEUE_TIMEOUT `#define OS_QUEUE_TIMEOUT (-10)`

Queue timeout.

Definition at line 106 of file osapi-error.h.

5.9.2.35 OS_SEM_FAILURE `#define OS_SEM_FAILURE (-6)`

Semaphore failure.

Definition at line 102 of file osapi-error.h.

5.9.2.36 OS_SEM_TIMEOUT `#define OS_SEM_TIMEOUT (-7)`

Semaphore timeout.

Definition at line 103 of file osapi-error.h.

5.9.2.37 OS_SUCCESS `#define OS_SUCCESS (0)`

Successful execution.

Definition at line 96 of file osapi-error.h.

5.9.2.38 OS_TIMER_ERR_INTERNAL `#define OS_TIMER_ERR_INTERNAL (-32)`

Timer internal error.

Definition at line 122 of file osapi-error.h.

5.9.2.39 OS_TIMER_ERR_INVALID_ARGS `#define OS_TIMER_ERR_INVALID_ARGS (-29)`

Timer invalid arguments.

Definition at line 119 of file osapi-error.h.

5.9.2.40 OS_TIMER_ERR_TIMER_ID `#define OS_TIMER_ERR_TIMER_ID (-30)`

Timer ID error.

Definition at line 120 of file osapi-error.h.

5.9.2.41 OS_TIMER_ERR_UNAVAILABLE `#define OS_TIMER_ERR_UNAVAILABLE (-31)`

Timer unavailable.

Definition at line 121 of file osapi-error.h.

5.10 OSAL Error Info APIs

Functions

- static long OS_StatusToInteger (osal_status_t Status)
  Convert a status code to a native "long" type.
- int32 OS_GetErrorName (int32 error_num, os_err_name_t *err_name)
  Convert an error number to a string.
- char * OS_StatusToString (osal_status_t status, os_status_string_t *status_string)
  Convert status to a string.

5.10.1 Detailed Description

5.10.2 Function Documentation

5.10.2.1 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

Return values

OS_SUCCESS	if successfully converted to a string
OS_INVALID_POINTER	if err_name is NULL
OS_ERROR	if error could not be converted

5.10.2.2 OS_StatusToInteger()

```
static long OS_StatusToInteger (
        osal_status_t Status ) [inline], [static]
```

Convert a status code to a native "long" type.

For printing or logging purposes, this converts the given status code to a "long" (signed integer) value. It should be used in conjunction with the "%ld" conversion specifier in printf-style statements.

Parameters

in	*Status*	Execution status, see OSAL Return Code Defines

Returns

Same status value converted to the "long" data type

Definition at line 164 of file osapi-error.h.

5.10.2.3 OS_StatusToString()

```
char* OS_StatusToString (
        osal_status_t status,
        os_status_string_t * status_string )
```

Convert status to a string.

Parameters

in	*status*	Status value to convert
out	*status_string*	Buffer to store status converted to string

Returns

Passed in string pointer

5.11 OSAL File Access Option Defines

Macros

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

5.11.1 Detailed Description

5.11.2 Macro Definition Documentation

5.11.2.1 OS_READ_ONLY `#define OS_READ_ONLY 0`

Read only file access

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

5.11.2.2 OS_READ_WRITE `#define OS_READ_WRITE 2`

Read write file access

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

5.11.2.3 OS_WRITE_ONLY `#define OS_WRITE_ONLY 1`

Write only file access

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

5.12 OSAL Reference Point For Seek Offset Defines

Macros

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

5.12.1 Detailed Description

5.12.2 Macro Definition Documentation

5.12.2.1 OS_SEEK_CUR `#define OS_SEEK_CUR 1`

Seek offset current

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

5.12.2.2 OS_SEEK_END `#define OS_SEEK_END 2`

Seek offset end

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

5.12.2.3 OS_SEEK_SET `#define OS_SEEK_SET 0`

Seek offset set

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

5.13 OSAL Standard File APIs

Functions

- int32 OS_OpenCreate (osal_id_t *filedes, const char *path, int32 flags, int32 access_mode)
  Open or create a file.
- int32 OS_close (osal_id_t filedes)
  Closes an open file handle.
- int32 OS_read (osal_id_t filedes, void *buffer, size_t nbytes)
  Read from a file handle.
- int32 OS_write (osal_id_t filedes, const void *buffer, size_t nbytes)
  Write to a file handle.
- int32 OS_TimedReadAbs (osal_id_t filedes, void *buffer, size_t nbytes, OS_time_t abstime)
  File/Stream input read with a timeout.
- int32 OS_TimedRead (osal_id_t filedes, void *buffer, size_t nbytes, int32 timeout)
  File/Stream input read with a timeout.
- int32 OS_TimedWriteAbs (osal_id_t filedes, const void *buffer, size_t nbytes, OS_time_t abstime)
  File/Stream output write with a timeout.

- int32 OS_TimedWrite (osal_id_t filedes, const void *buffer, size_t nbytes, int32 timeout)

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

  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 (osal_id_t 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 (osal_id_t 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.

5.13.1 Detailed Description

5.13.2 Function Documentation

5.13.2.1 OS_chmod()

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

Changes the permissions of a file.

Parameters

in	*path*	File to change (must not be null)
in	*access_mode*	Desired access mode - see OSAL File Access Option Defines

Note

Some file systems do not implement permissions. If the underlying OS does not support this operation, then OS_ERR_NOT_IMPLEMENTED is returned.

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution. (return value only verified in coverage test)
OS_ERR_NOT_IMPLEMENTED	if the filesystem does not support this call
OS_INVALID_POINTER	if the path argument is NULL

5.13.2.2 OS_close() `int32 OS_close ( osal_id_t 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_ERR_INVALID_ID	if the file descriptor passed in is invalid
OS_ERROR	if an unexpected/unhandled error occurs (return value only verified in coverage test)

5.13.2.3 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 (return value only verified in coverage test)

5.13.2.4 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 (must not be null)

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 (return value only verified in coverage test)
OS_INVALID_POINTER	if the filename argument is NULL

5.13.2.5 OS_cp()

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

Copies a single file from src to dest.

Note

The behavior of this API on an open file is not defined at the OSAL level due to dependencies on the underlying OS which may or may not allow the related operation based on a variety of potential configurations. For portability, it is recommended that applications ensure the file is closed prior to removal.

Parameters

in	*src*	The source file to operate on (must not be null)
in	*dest*	The destination file (must not be null)

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

5.13.2.6 OS_FDGetInfo()

```
int32 OS_FDGetInfo (
        osal_id_t 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 (must not be null)

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
OS_INVALID_POINTER	if the fd_prop argument is NULL

5.13.2.7 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 (must not be null)

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	if the file is open
OS_ERROR	if the file is not open
OS_INVALID_POINTER	if the filename argument is NULL

5.13.2.8 OS_lseek()

```
int32 OS_lseek (
        osal_id_t 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 Reference 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 (return value only verified in coverage test)

5.13.2.9 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.

Note

The behavior of this API on an open file is not defined at the OSAL level due to dependencies on the underlying OS which may or may not allow the related operation based on a variety of potential configurations. For portability, it is recommended that applications ensure the file is closed prior to removal.

Parameters

in	*src*	The source file to operate on (must not be null)
in	*dest*	The destination file (must not be null)

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

5.13.2.10 OS_OpenCreate()

```
int32 OS_OpenCreate (
    osal_id_t * filedes,
    const char * path,
    int32 flags,
    int32 access_mode )
```

Open or create a file.

Implements the same as OS_open/OS_creat but follows the OSAL paradigm of outputting the ID/descriptor separately from the return value, rather than relying on the user to convert it back.

Parameters

out	*filedes*	The handle ID (OS_OBJECT_ID_UNDEFINED on failure) (must not be null)
in	*path*	File name to create or open (must not be null)
in	*flags*	The file permissions - see OS_file_flag_t
in	*access_mode*	Intended access mode - see OSAL File Access Option Defines

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERROR	if the command was not executed properly
OS_INVALID_POINTER	if pointer argument was NULL
OS_ERR_NO_FREE_IDS	if all available file handles are in use
OS_FS_ERR_NAME_TOO_LONG	if the filename portion of the path exceeds OS_MAX_FILE_NAME
OS_FS_ERR_PATH_INVALID	if the path argument is not valid
OS_FS_ERR_PATH_TOO_LONG	if the path argument exceeds OS_MAX_PATH_LEN

5.13.2.11 OS_read()

```
int32 OS_read (
    osal_id_t filedes,
    void * buffer,
    size_t nbytes )
```

Read from a file handle.

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

If the file position is at the end of file (or beyond, if the OS allows) then this function will return 0.

Parameters

in	*filedes*	The handle ID to operate on
out	*buffer*	Storage location for file data (must not be null)
in	*nbytes*	Maximum number of bytes to read (must not be zero)

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_ERR_INVALID_SIZE	if the passed-in size is not valid
OS_ERROR	if OS call failed (return value only verified in coverage test)
OS_ERR_INVALID_ID	if the file descriptor passed in is invalid
0	if at end of file/stream data

5.13.2.12 OS_remove()

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

Removes a file from the file system.

Removes a given filename from the drive

Note

The behavior of this API on an open file is not defined at the OSAL level due to dependencies on the underlying OS which may or may not allow the related operation based on a variety of potential configurations. For portability, it is recommended that applications ensure the file is closed prior to removal.

Parameters

in	*path*	The file to operate on (must not be null)

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

5.13.2.13 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

The behavior of this API on an open file is not defined at the OSAL level due to dependencies on the underlying OS which may or may not allow the related operation based on a variety of potential configurations. For portability, it is recommended that applications ensure the file is closed prior to removal.

Parameters

in	*old_filename*	The original filename (must not be null)
in	*new_filename*	The desired filename (must not be null)

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_filename or new_filename 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

5.13.2.14 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 an os_fstat_t structure

Parameters

in	*path*	The file to operate on (must not be null)
out	*filestats*	Buffer to store file information (must not be null)

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

5.13.2.15 OS_TimedRead()

```
int32 OS_TimedRead (
    osal_id_t filedes,
    void * buffer,
    size_t 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, such as pipes or special devices.

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 the file position is at the end of file or end of stream data (e.g. if the remote end has closed the connection), then this function will immediately return 0 without blocking for the timeout period.

If no data is immediately available, but the underlying resource/stream is still connected to a peer, this will wait up to the given timeout for additional data to appear. If no data appears within the timeout period, then this returns the OS_ERROR_TIMEOUT status code. This allows the caller to differentiate an open (but idle) socket connection from a connection which has been closed by the remote peer.

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
out	*buffer*	Storage location for file data (must not be null)
in	*nbytes*	Maximum number of bytes to read (must not be zero)
in	*timeout*	Maximum time to wait, in milliseconds, relative to current time (OS_PEND = forever)

Returns

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

Return values

OS_ERROR_TIMEOUT	if no data became available during timeout period
OS_ERR_INVALID_ID	if the file descriptor passed in is invalid
OS_ERR_INVALID_SIZE	if the passed-in size is not valid
OS_INVALID_POINTER	if the passed-in buffer is not valid
0	if at end of file/stream data

5.13.2.16 OS_TimedReadAbs()

```
int32 OS_TimedReadAbs (
    osal_id_t filedes,
    void * buffer,
    size_t nbytes,
    OS_time_t abstime )
```

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, such as pipes or special devices.

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 the file position is at the end of file or end of stream data (e.g. if the remote end has closed the connection), then this function will immediately return 0 without blocking for the timeout period.

If no data is immediately available, but the underlying resource/stream is still connected to a peer, this will wait up to the given timeout for additional data to appear. If no data appears within the timeout period, then this returns the OS_ERROR_TIMEOUT status code. This allows the caller to differentiate an open (but idle) socket connection from a connection which has been closed by the remote peer.

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
out	*buffer*	Storage location for file data (must not be null)
in	*nbytes*	Maximum number of bytes to read (must not be zero)
in	*abstime*	Absolute time at which this function should return, if no data is readable

Returns

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

Return values

OS_ERROR_TIMEOUT	if no data became available during timeout period
OS_ERR_INVALID_ID	if the file descriptor passed in is invalid
OS_ERR_INVALID_SIZE	if the passed-in size is not valid
OS_INVALID_POINTER	if the passed-in buffer is not valid
0	if at end of file/stream data

5.13.2.17 OS_TimedWrite()

```
int32 OS_TimedWrite (
    osal_id_t filedes,
    const void * buffer,
    size_t 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 (must not be null)
in	*nbytes*	Maximum number of bytes to read (must not be zero)
in	*timeout*	Maximum time to wait, in milliseconds, relative to current time (OS_PEND = forever)

Returns

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

Return values

OS_ERROR_TIMEOUT	if no data became available during timeout period
OS_ERR_INVALID_ID	if the file descriptor passed in is invalid
OS_ERR_INVALID_SIZE	if the passed-in size is not valid
OS_INVALID_POINTER	if the passed-in buffer is not valid
0	if file/stream cannot accept any more data

5.13.2.18 OS_TimedWriteAbs()

```
int32 OS_TimedWriteAbs (
    osal_id_t filedes,
    const void * buffer,
    size_t nbytes,
    OS_time_t abstime )
```

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 (must not be null)
in	*nbytes*	Maximum number of bytes to read (must not be zero)
in	*abstime*	Absolute time at which this function should return, if no data is readable

Returns

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

Return values

OS_ERROR_TIMEOUT	if no data became available during timeout period
OS_ERR_INVALID_ID	if the file descriptor passed in is invalid

Return values

OS_ERR_INVALID_SIZE	if the passed-in size is not valid
OS_INVALID_POINTER	if the passed-in buffer is not valid
0	if file/stream cannot accept any more data

5.13.2.19 OS_write()

```
int32 OS_write (
    osal_id_t filedes,
    const void * buffer,
    size_t 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 (must not be null)
in	*nbytes*	Maximum number of bytes to read (must not be zero)

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_ERR_INVALID_SIZE	if the passed-in size is not valid
OS_ERROR	if OS call failed (return value only verified in coverage test)
OS_ERR_INVALID_ID	if the file descriptor passed in is invalid
0	if file/stream cannot accept any more data

5.14 OSAL File System Level APIs

Functions

- int32 OS_FileSysAddFixedMap (osal_id_t *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, size_t blocksize, osal_blockcount_t 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, size_t blocksize, osal_blockcount_t 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_FileSysStatVolume (const char *name, OS_statvfs_t *statbuf)

  Obtains information about size and free space 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 an OSAL Virtual file system path to a host Local path.
- int32 OS_GetFsInfo (os_fsinfo_t *filesys_info)

  Returns information about the file system.

5.14.1 Detailed Description

5.14.2 Function Documentation

5.14.2.1 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. If the underlying OS does not provide a facility to check the volume, then OS_ERR_NOT_IMPLEMENTED will be returned.

Parameters

in	*name*	The device/path to operate on (must not be null)
in	*repair*	Whether to also repair inconsistencies

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution. (return value only verified in coverage test)
OS_INVALID_POINTER	Name is NULL
OS_ERR_NOT_IMPLEMENTED	Not implemented.
OS_FS_ERR_PATH_TOO_LONG	if the name is too long
OS_ERROR	Failed execution. (return value only verified in coverage test)

5.14.2.2 OS_FileSysAddFixedMap()

```
int32 OS_FileSysAddFixedMap (
    osal_id_t * 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 application.

Note

OSAL virtual mount points are required to be a single, non-empty top-level directory name. Virtual path names always follow the form /<virt_mount_point>/<relative_path>/<file>. Only the relative path may be omitted/empty (i.e. /<virt_mount_point>/<file>) but the virtual mount point must be present and not an empty string. In particular this means it is not possible to directly refer to files in the "root" of the native file system from OSAL. However it is possible to create a virtual map to the root, such as by calling:

```
OS_FileSysAddFixedMap(&fs_id, "/", "/root");
```

Parameters

out	*filesys_id*	A buffer to store the ID of the file system mapping (must not be null)
in	*phys_path*	The native system directory (an existing mount point) (must not be null)
in	*virt_path*	The virtual mount point of this filesystem (must not be null)

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_FS_ERR_PATH_TOO_LONG	if the overall phys_path is too long

Return values

OS_ERR_NAME_TOO_LONG	if the phys_path basename (filesystem name) is too long
OS_INVALID_POINTER	if any argument is NULL

5.14.2.3 OS_FileSysStatVolume()

```
int32 OS_FileSysStatVolume (
    const char * name,
    OS_statvfs_t * statbuf )
```

Obtains information about size and free space in a volume.

Populates the supplied OS_statvfs_t structure, which includes the block size and total/free blocks in a file system volume.

This replaces two older OSAL calls:

OS_fsBlocksFree() is determined by reading the blocks_free output struct member OS_fsBytesFree() is determined by multiplying blocks_free by the block_size member

Parameters

in	*name*	The device/path to operate on (must not be null)
out	*statbuf*	Output structure to populate (must not be null)

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if name or statbuf is NULL
OS_FS_ERR_PATH_TOO_LONG	if the name is too long
OS_ERROR	if an unexpected/unhandled OS error occurs (return value only verified in coverage test)

5.14.2.4 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 (must not be null)
in	*MountPoint*	OSAL mount point (must not be null)

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if either parameter is NULL
OS_ERR_NAME_NOT_FOUND	if the MountPoint is not mounted in OSAL
OS_FS_ERR_PATH_TOO_LONG	if the MountPoint is too long

5.14.2.5 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 (must not be null)

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if filesys_info is NULL

5.14.2.6 OS_initfs()

```
int32 OS_initfs (
        char * address,
        const char * devname,
        const char * volname,
```

```
        size_t blocksize,
        osal_blockcount_t numblocks )
```

Initializes an existing file system.

Initializes a file system on the target.

Note

The "volname" parameter of RAM disks should always begin with the string "RAM", e.g. "RAMDISK" or "RAM0","↩ RAM1", etc if multiple devices are created. The underlying implementation uses this to select the correct filesystem type/format, and this may also be used to differentiate between RAM disks and real physical disks.

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 underlying kernel device to use, if applicable. (must not be null)
in	*volname*	The name of the volume (see note) (must not be null)
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	if an unexpected/unhandled OS error occurs (return value only verified in coverage test)

5.14.2.7 OS_mkfs()

```
int32 OS_mkfs (
        char * address,
        const char * devname,
        const char * volname,
        size_t blocksize,
        osal_blockcount_t 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.

Note

The "volname" parameter of RAM disks should always begin with the string "RAM", e.g. "RAMDISK" or "RAM0","↩ RAM1", etc if multiple devices are created. The underlying implementation uses this to select the correct filesystem type/format, and this may also be used to differentiate between RAM disks and real physical disks.

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 underlying kernel device to use, if applicable. (must not be null)
in	*volname*	The name of the volume (see note) (must not be null)
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 is NULL
OS_FS_ERR_PATH_TOO_LONG	if the overall devname or volname is too long
OS_FS_ERR_DEVICE_NOT_FREE	if the volume table is full
OS_FS_ERR_DRIVE_NOT_CREATED	if an unexpected/unhandled OS error occurs (return value only verified in coverage test)

5.14.2.8 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 (must not be null)
in	*mountpoint*	The name to call this disk from now on (must not be null)

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_NAME_NOT_FOUND	if the device name does not exist in OSAL
OS_FS_ERR_PATH_TOO_LONG	if the mount point string is too long
OS_INVALID_POINTER	if any argument is NULL
OS_ERROR	if an unexpected/unhandled OS error occurs (return value only verified in coverage test)

5.14.2.9 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 (must not be null)

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if devname is NULL
OS_FS_ERR_PATH_TOO_LONG	if the devname is too long
OS_ERR_NAME_NOT_FOUND	if the devname does not exist in OSAL
OS_ERROR	if an unexpected/unhandled OS error occurs (return value only verified in coverage test)

5.14.2.10 OS_TranslatePath()

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

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

Translates a virtual path to an actual system path name

Note

The buffer provided in the LocalPath argument is required to be at least OS_MAX_PATH_LEN characters in length.

Parameters

in	*VirtualPath*	OSAL virtual path name (must not be null)
out	*LocalPath*	Buffer to store native/translated path name (must not be null)

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if either parameter is NULL
OS_FS_ERR_NAME_TOO_LONG	if the filename component is too long
OS_FS_ERR_PATH_INVALID	if either parameter cannot be interpreted as a path
OS_FS_ERR_PATH_TOO_LONG	if either input or output pathnames are too long

5.14.2.11 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 (must not be null)

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_ERR_NAME_NOT_FOUND	if the mountpoint is not mounted in OSAL
OS_ERROR	if an unexpected/unhandled OS error occurs (return value only verified in coverage test)

5.15 OSAL Heap APIs

Functions

- int32 OS_HeapGetInfo (OS_heap_prop_t *heap_prop)

  Return current info on the heap.

5.15.1 Detailed Description

5.15.2 Function Documentation

5.15.2.1 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

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if the heap_prop argument is NULL

5.16 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_OS_CONDVAR 0x0D

  Object condition variable type.
- #define OS_OBJECT_TYPE_USER 0x10

  Object user type.

5.16.1 Detailed Description

5.16.2 Macro Definition Documentation

5.16.2.1 OS_OBJECT_TYPE_OS_BINSEM `#define OS_OBJECT_TYPE_OS_BINSEM 0x04`

Object binary semaphore type.

Definition at line 42 of file osapi-idmap.h.

5.16.2.2 OS_OBJECT_TYPE_OS_CONDVAR `#define OS_OBJECT_TYPE_OS_CONDVAR 0x0D`

Object condition variable type.

Definition at line 51 of file osapi-idmap.h.

5.16.2.3 OS_OBJECT_TYPE_OS_CONSOLE `#define OS_OBJECT_TYPE_OS_CONSOLE 0x0C`

Object console type.

Definition at line 50 of file osapi-idmap.h.

5.16.2.4 OS_OBJECT_TYPE_OS_COUNTSEM `#define OS_OBJECT_TYPE_OS_COUNTSEM 0x03`

Object counting semaphore type.

Definition at line 41 of file osapi-idmap.h.

5.16.2.5 OS_OBJECT_TYPE_OS_DIR `#define OS_OBJECT_TYPE_OS_DIR 0x07`

Object directory type.

Definition at line 45 of file osapi-idmap.h.

5.16.2.6 OS_OBJECT_TYPE_OS_FILESYS `#define OS_OBJECT_TYPE_OS_FILESYS 0x0B`

Object file system type.

Definition at line 49 of file osapi-idmap.h.

5.16.2.7 OS_OBJECT_TYPE_OS_MODULE `#define OS_OBJECT_TYPE_OS_MODULE 0x0A`

Object module type.

Definition at line 48 of file osapi-idmap.h.

5.16.2.8 OS_OBJECT_TYPE_OS_MUTEX `#define OS_OBJECT_TYPE_OS_MUTEX 0x05`

Object mutex type.

Definition at line 43 of file osapi-idmap.h.

5.16.2.9 OS_OBJECT_TYPE_OS_QUEUE `#define OS_OBJECT_TYPE_OS_QUEUE 0x02`

Object queue type.

Definition at line 40 of file osapi-idmap.h.

5.16.2.10 OS_OBJECT_TYPE_OS_STREAM `#define OS_OBJECT_TYPE_OS_STREAM 0x06`

Object stream type.

Definition at line 44 of file osapi-idmap.h.

5.16.2.11 OS_OBJECT_TYPE_OS_TASK `#define OS_OBJECT_TYPE_OS_TASK 0x01`

Object task type.

Definition at line 39 of file osapi-idmap.h.

5.16.2.12 OS_OBJECT_TYPE_OS_TIMEBASE `#define OS_OBJECT_TYPE_OS_TIMEBASE 0x08`

Object timebase type.

Definition at line 46 of file osapi-idmap.h.

5.16.2.13 OS_OBJECT_TYPE_OS_TIMECB `#define OS_OBJECT_TYPE_OS_TIMECB 0x09`

Object timer callback type.

Definition at line 47 of file osapi-idmap.h.

5.16.2.14 OS_OBJECT_TYPE_UNDEFINED `#define OS_OBJECT_TYPE_UNDEFINED 0x00`

Object type undefined.

Definition at line 38 of file osapi-idmap.h.

5.16.2.15 OS_OBJECT_TYPE_USER `#define OS_OBJECT_TYPE_USER 0x10`

Object user type.

Definition at line 52 of file osapi-idmap.h.

5.17 OSAL Object ID Utility APIs

Functions

- static unsigned long OS_ObjectIdToInteger (osal_id_t object_id)

  Obtain an integer value corresponding to an object ID.
- static osal_id_t OS_ObjectIdFromInteger (unsigned long value)

  Obtain an osal ID corresponding to an integer value.
- static bool OS_ObjectIdEqual (osal_id_t object_id1, osal_id_t object_id2)

  Check two OSAL object ID values for equality.
- static bool OS_ObjectIdDefined (osal_id_t object_id)

  Check if an object ID is defined.
- int32 OS_GetResourceName (osal_id_t object_id, char *buffer, size_t buffer_size)

  Obtain the name of an object given an arbitrary object ID.
- osal_objtype_t OS_IdentifyObject (osal_id_t object_id)

  Obtain the type of an object given an arbitrary object ID.
- int32 OS_ConvertToArrayIndex (osal_id_t object_id, osal_index_t *ArrayIndex)

  Converts an abstract ID into a number suitable for use as an array index.
- int32 OS_ObjectIdToArrayIndex (osal_objtype_t idtype, osal_id_t object_id, osal_index_t *ArrayIndex)

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

  call the supplied callback function for all valid object IDs
- void OS_ForEachObjectOfType (osal_objtype_t objtype, osal_id_t creator_id, OS_ArgCallback_t callback_ptr, void *callback_arg)

  call the supplied callback function for valid object IDs of a specific type

5.17.1 Detailed Description

5.17.2 Function Documentation

5.17.2.1 OS_ConvertToArrayIndex()

```
int32 OS_ConvertToArrayIndex (
        osal_id_t object_id,
        osal_index_t * 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.

This routine accepts any object type, and returns a value based on the maximum number of objects for that type. This is equivalent to invoking OS_ObjectIdToArrayIndex() with the idtype set to OS_OBJECT_TYPE_UNDEFINED.

See also

OS_ObjectIdToArrayIndex

Parameters

in	*object_id*	The object ID to operate on
out	**ArrayIndex*	The Index to return (must not be null)

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the object_id argument is not valid
OS_INVALID_POINTER	if the ArrayIndex is NULL

5.17.2.2 OS_ForEachObject()

```
void OS_ForEachObject (
        osal_id_t 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 of all types and calls callback_ptr on each one If creator_id is nonzero then only objects with matching creator id are processed.

Parameters

in	*creator_id*	Filter objects to those created by a specific task This may be passed as OS_OBJECT_CREATOR_ANY to return all objects
in	*callback_ptr*	Function to invoke for each matching object ID
in	*callback_arg*	Opaque Argument to pass to callback function (may be NULL)

5.17.2.3 OS_ForEachObjectOfType()

```
void OS_ForEachObjectOfType (
    osal_objtype_t objtype,
    osal_id_t creator_id,
    OS_ArgCallback_t callback_ptr,
    void * callback_arg )
```

call the supplied callback function for valid object IDs of a specific type

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

Parameters

in	*objtype*	The type of objects to iterate
in	*creator_id*	Filter objects to those created by a specific task This may be passed as OS_OBJECT_CREATOR_ANY to return all objects
in	*callback_ptr*	Function to invoke for each matching object ID
in	*callback_arg*	Opaque Argument to pass to callback function (may be NULL)

5.17.2.4 OS_GetResourceName()

```
int32 OS_GetResourceName (
    osal_id_t object_id,
    char * buffer,
    size_t buffer_size )
```

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

All OSAL resources generally have a name associated with them. This allows application code to retrieve the name of any valid OSAL object ID.

Parameters

in	*object_id*	The object ID to operate on
out	*buffer*	Buffer in which to store the name (must not be null)
in	*buffer_size*	Size of the output storage buffer (must not be zero)

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the passed-in ID is not a valid OSAL ID
OS_INVALID_POINTER	if the passed-in buffer is invalid
OS_ERR_NAME_TOO_LONG	if the name will not fit in the buffer provided

5.17.2.5 OS_IdentifyObject() `osal_objtype_t OS_IdentifyObject ( osal_id_t 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

5.17.2.6 OS_ObjectIdDefined() `static bool OS_ObjectIdDefined ( osal_id_t object_id ) [inline], [static]`

Check if an object ID is defined.

The OSAL ID values should be treated as abstract values by applications, and not directly manipulated using standard C operators.

This returns false if the ID is NOT a defined resource (i.e. free/empty/invalid).

Note

OS_ObjectIdDefined(OS_OBJECT_ID_UNDEFINED) is always guaranteed to be false.

Parameters

in	*object_id*	The first object ID

Definition at line 150 of file osapi-idmap.h.

References OS_ObjectIdToInteger().

5.17.2.7 OS_ObjectIdEqual()

```
static bool OS_ObjectIdEqual (
        osal_id_t object_id1,
        osal_id_t object_id2 )  [inline], [static]
```

Check two OSAL object ID values for equality.

The OSAL ID values should be treated as abstract values by applications, and not directly manipulated using standard C operators.

This checks two values for equality, replacing the "==" operator.

Parameters

in	*object_id1*	The first object ID
in	*object_id2*	The second object ID

Returns

true if the object IDs are equal

Definition at line 129 of file osapi-idmap.h.

References OS_ObjectIdToInteger().

5.17.2.8 OS_ObjectIdFromInteger()

```
static osal_id_t OS_ObjectIdFromInteger (
        unsigned long value )  [inline], [static]
```

Obtain an osal ID corresponding to an integer value.

Provides the inverse of OS_ObjectIdToInteger(). Reconstitutes the original osal_id_t type from an integer representation.

Parameters

in	*value*	The integer representation of an OSAL ID

Returns

The ID value converted to an osal_id_t

Definition at line 102 of file osapi-idmap.h.

5.17.2.9 OS_ObjectIdToArrayIndex()

```
int32 OS_ObjectIdToArrayIndex (
        osal_objtype_t idtype,
        osal_id_t object_id,
        osal_index_t * 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.

This routine operates on a specific object type, and returns a value based on the maximum number of objects for that type.

If the idtype is passed as OS_OBJECT_TYPE_UNDEFINED, then object type verification is skipped and any object ID will be accepted and converted to an index. In this mode, the range of the output depends on the actual passed-in object type.

If the idtype is passed as any other value, the passed-in ID value is first confirmed to be the correct type. This check will guarantee that the output is within an expected range; for instance, if the type is passed as OS_OBJECT_TYPE_OS_TASK, then the output index is guaranteed to be between 0 and OS_MAX_TASKS-1 after successful conversion.

Parameters

in	*idtype*	The object type to convert
in	*object_id*	The object ID to operate on
out	**ArrayIndex*	The Index to return (must not be null)

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the object_id argument is not valid
OS_INVALID_POINTER	if the ArrayIndex is NULL

5.17.2.10 OS_ObjectIdToInteger() `static unsigned long OS_ObjectIdToInteger ( osal_id_t object_id ) [inline], [static]`

Obtain an integer value corresponding to an object ID.

Obtains an integer representation of an object id, generally for the purpose of printing to the console or system logs.

The returned value is of the type "unsigned long" for direct use with printf-style functions. It is recommended to use the "%lx" conversion specifier as the hexadecimal encoding clearly delineates the internal fields.

Note

This provides the raw integer value and is *not* suitable for use as an array index, as the result is not zero-based. See the OS_ConvertToArrayIndex() to obtain a zero-based index value.

Parameters

in	*object_id*	The object ID

Returns

integer value representation of object ID

Definition at line 80 of file osapi-idmap.h.

Referenced by OS_ObjectIdDefined(), and OS_ObjectIdEqual().

5.18 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_ModuleSymbolLookup (osal_id_t module_id, cpuaddr *symbol_address, const char *symbol_name)
  Find the Address of a Symbol within a module.
- int32 OS_SymbolTableDump (const char *filename, size_t size_limit)
  Dumps the system symbol table to a file.
- int32 OS_ModuleLoad (osal_id_t *module_id, const char *module_name, const char *filename, uint32 flags)
  Loads an object file.
- int32 OS_ModuleUnload (osal_id_t module_id)
  Unloads the module file.
- int32 OS_ModuleInfo (osal_id_t module_id, OS_module_prop_t *module_info)
  Obtain information about a module.

5.18.1 Detailed Description

5.18.2 Function Documentation

5.18.2.1 OS_ModuleInfo()

```
int32 OS_ModuleInfo (
        osal_id_t 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 (must not be null)

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
OS_ERROR	if an other/unspecified error occurs (return value only verified in coverage test)

5.18.2.2 OS_ModuleLoad()

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

Loads an object file.

Loads an object file into the running operating system

The "flags" parameter may influence how the loaded module symbols are made available for use in the application. See OS_MODULE_FLAG_LOCAL_SYMBOLS and OS_MODULE_FLAG_GLOBAL_SYMBOLS for descriptions.

Parameters

out	*module_id*	Non-zero OSAL ID corresponding to the loaded module
in	*module_name*	Name of module (must not be null)
in	*filename*	File containing the object code to load (must not be null)
in	*flags*	Options for the loaded module

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
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
OS_ERR_NAME_TOO_LONG	if the module_name is too long
OS_FS_ERR_PATH_INVALID	if the filename argument is not valid
OS_ERROR	if an other/unspecified error occurs (return value only verified in coverage test)

5.18.2.3 OS_ModuleSymbolLookup()

```
int32 OS_ModuleSymbolLookup (
    osal_id_t module_id,
    cpuaddr * symbol_address,
    const char * symbol_name )
```

Find the Address of a Symbol within a module.

This is similar to OS_SymbolLookup() but for a specific module ID. This should be used to look up a symbol in a module that has been loaded with the OS_MODULE_FLAG_LOCAL_SYMBOLS flag.

Parameters

in	*module_id*	Module ID that should contain the symbol
out	*symbol_address*	Set to the address of the symbol (must not be null)
in	*symbol_name*	Name of the symbol to look up (must not be null)

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

5.18.2.4 OS_ModuleUnload() `int32 OS_ModuleUnload ( osal_id_t 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_ERR_INVALID_ID	if the module id invalid
OS_ERROR	if an other/unspecified error occurs (return value only verified in coverage test)

5.18.2.5 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 (must not be null)
in	*symbol_name*	Name of the symbol to look up (must not be null)

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

5.18.2.6 OS_SymbolTableDump()

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

Dumps the system symbol table to a file.

Dumps the system symbol table to the specified filename

Note

Not all RTOS implementations support this API. If the underlying module subsystem does not provide a facility to iterate through the symbol table, then the OS_ERR_NOT_IMPLEMENTED status code is returned.

Parameters

in	*filename*	File to write to (must not be null)
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_INVALID_POINTER	if the filename argument is NULL
OS_FS_ERR_PATH_INVALID	if the filename argument is not valid
OS_ERR_NAME_TOO_LONG	if any of the symbol names are too long (return value only verified in coverage test)
OS_ERR_OUTPUT_TOO_LARGE	if the size_limit was reached before completing all symbols (return value only verified in coverage test)
OS_ERROR	if an other/unspecified error occurs (return value only verified in coverage test)

5.19 OSAL Mutex APIs

Functions

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

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

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

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

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

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

  Fill a property object buffer with details regarding the resource.

5.19.1 Detailed Description

5.19.2 Function Documentation

5.19.2.1 OS_MutSemCreate()

```
int32 OS_MutSemCreate (
    osal_id_t * 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 non-zero ID of the newly-created resource (must not be null)
in	*sem_name*	the name of the new resource to create (must not be null)
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.

Return values

OS_INVALID_POINTER	if sem_id or sem_name are NULL
OS_ERR_NAME_TOO_LONG	name length including null terminator greater than OS_MAX_API_NAME
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 (return value only verified in coverage test)

5.19.2.2 OS_MutSemDelete()

```
int32 OS_MutSemDelete (
    osal_id_t 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 an unspecified error occurs (return value only verified in coverage test)

5.19.2.3 OS_MutSemGetIdByName()

```
int32 OS_MutSemGetIdByName (
    osal_id_t * 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 (must not be null)

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	name length including null terminator greater than OS_MAX_API_NAME
OS_ERR_NAME_NOT_FOUND	if the name was not found in the table

5.19.2.4 OS_MutSemGetInfo()

```
int32 OS_MutSemGetInfo (
    osal_id_t 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 (must not be null)

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

5.19.2.5 OS_MutSemGive()

```
int32 OS_MutSemGive (
    osal_id_t 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 (return value only verified in coverage test)

5.19.2.6 OS_MutSemTake() `int32 OS_MutSemTake ( osal_id_t 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_ERR_INVALID_ID	the id passed in is not a valid mutex
OS_SEM_FAILURE	if an unspecified error occurs (return value only verified in coverage test)

5.20 OSAL Network ID APIs

Functions

- int32 OS_NetworkGetID (void)

Gets the network ID of the local machine.

- int32 OS_NetworkGetHostName (char *host_name, size_t name_len)

  Gets the local machine network host name.

5.20.1 Detailed Description

Provides some basic methods to query a network host name and ID

5.20.2 Function Documentation

5.20.2.1 OS_NetworkGetHostName() int32 OS_NetworkGetHostName (char * *host_name*, size_t *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 (must not be null)
in	*name_len*	Maximum length of host name buffer (must not be zero)

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_SIZE	if the name_len is zero
OS_INVALID_POINTER	if the host_name is NULL

5.20.2.2 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.

5.21 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.

5.21.1 Detailed Description

5.21.2 Function Documentation

5.21.2.1 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.

Strings (including terminator) longer than OS_BUFFER_SIZE will be truncated.

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

5.21.2.2 OS_printf_disable() `void void OS_printf_disable (`

```
void  )
```

This function disables the output from OS_printf.

5.21.2.3 OS_printf_enable() `void OS_printf_enable (`

```
void  )
```

This function enables the output from OS_printf.

5.22 OSAL Message Queue APIs

Functions

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

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

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

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

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

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

  Fill a property object buffer with details regarding the resource.

5.22.1 Detailed Description

5.22.2 Function Documentation

5.22.2.1 OS_QueueCreate() `int32 OS_QueueCreate (`

```
osal_id_t * queue_id,
const char * queue_name,
osal_blockcount_t queue_depth,
size_t 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 non-zero ID of the newly-created resource (must not be null)
in	*queue_name*	the name of the new resource to create (must not be null)
in	*queue_depth*	the maximum depth of the queue
in	*data_size*	the size of each entry in the queue (must not be zero)
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	name length including null terminator greater than OS_MAX_API_NAME
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_ERR_INVALID_SIZE	if data_size is 0
OS_QUEUE_INVALID_SIZE	if the queue depth exceeds the limit
OS_ERROR	if the OS create call fails

5.22.2.2 OS_QueueDelete() `int32 OS_QueueDelete ( osal_id_t 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 returns an unexpected error (return value only verified in coverage test)

5.22.2.3 OS_QueueGet()

```
int32 OS_QueueGet (
    osal_id_t queue_id,
    void * data,
    size_t size,
    size_t * 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 (must not be null)
in	*size*	The size of the data buffer (must not be zero)
out	*size_copied*	Set to the actual size of the message (must not be null)
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 received
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
OS_ERROR	if the OS call returns an unexpected error (return value only verified in coverage test)

5.22.2.4 OS_QueueGetIdByName()

```
int32 OS_QueueGetIdByName (
        osal_id_t * 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 (must not be null)

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	name length including null terminator greater than OS_MAX_API_NAME
OS_ERR_NAME_NOT_FOUND	the name was not found in the table

5.22.2.5 OS_QueueGetInfo()

```
int32 OS_QueueGetInfo (
        osal_id_t 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 (must not be null)

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

5.22.2.6 OS_QueuePut()

```
int32 OS_QueuePut (
    osal_id_t queue_id,
    const void * data,
    size_t 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 (must not be null)
in	*size*	The size of the data buffer (must not be zero)
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_INVALID_SIZE	if the data message is too large for the queue
OS_QUEUE_FULL	if the queue cannot accept another message
OS_ERROR	if the OS call returns an unexpected error (return value only verified in coverage test)

5.23 OSAL Select APIs

Functions

- int32 OS_SelectMultipleAbs (OS_FdSet *ReadSet, OS_FdSet *WriteSet, OS_time_t abs_timeout)

  Wait for events across multiple file handles.

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

  Wait for events across multiple file handles.
- int32 OS_SelectSingleAbs (osal_id_t objid, uint32 *StateFlags, OS_time_t abs_timeout)

  Wait for events on a single file handle.
- int32 OS_SelectSingle (osal_id_t 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, osal_id_t objid)

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

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

  Check if an FdSet structure contains a given ID.

5.23.1 Detailed Description

5.23.2 Function Documentation

5.23.2.1 OS_SelectFdAdd()

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

Add an ID to an FdSet structure.

After this call the set will contain the given OSAL ID

Parameters

in,out	*Set*	Pointer to OS_FdSet object to operate on (must not be null)
in	*objid*	The handle ID to add to the set

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if argument is NULL
OS_ERR_INVALID_ID	if the objid is not a valid handle

5.23.2.2 OS_SelectFdClear()

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

Clear an ID from an FdSet structure.

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

Parameters

in,out	*Set*	Pointer to OS_FdSet object to operate on (must not be null)
in	*objid*	The handle ID to remove from the set

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if argument is NULL
OS_ERR_INVALID_ID	if the objid is not a valid handle

5.23.2.3 OS_SelectFdIsSet()

```
bool OS_SelectFdIsSet (
        const OS_FdSet * Set,
        osal_id_t objid )
```

Check if an FdSet structure contains a given ID.

Parameters

in	*Set*	Pointer to OS_FdSet object to operate on (must not be null)
in	*objid*	The handle ID to check for in the set

Returns

Boolean set status

Return values

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

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

Clear a FdSet structure.

After this call the set will contain no OSAL IDs

Parameters

out	*Set*	Pointer to OS_FdSet object to clear (must not be null)

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if argument is NULL

5.23.2.5 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 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.

The timeout is expressed in milliseconds, relative to the time that the API was invoked. Use OS_SelectMultipleAbs() for higher timing precision.

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.

Parameters

in,out	*ReadSet*	Set of handles to check/wait to become readable
in,out	*WriteSet*	Set of handles to check/wait to become writable
in	*msecs*	Indicates the timeout. Positive values will wait up to that many milliseconds. Zero will not wait (poll). Negative values will wait forever (pend)

See also

OS_SelectMultipleAbs()

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	If any handle in the ReadSet or WriteSet is readable or writable, respectively
OS_ERROR_TIMEOUT	If no handles in the ReadSet or WriteSet became readable or writable within the timeout
OS_ERR_OPERATION_NOT_SUPPORTED	if a specified handle does not support select
OS_ERR_INVALID_ID	if no valid handles were contained in the ReadSet/WriteSet

5.23.2.6 OS_SelectMultipleAbs()

```
int32 OS_SelectMultipleAbs (
        OS_FdSet * ReadSet,
        OS_FdSet * WriteSet,
        OS_time_t abs_timeout )
```

Wait for events across multiple file handles.

Wait for any of the given sets of IDs to 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.

This API is identical to OS_SelectMultiple() except for the timeout parameter. In this call, timeout is expressed as an absolute value of the OS clock, in the same time domain as obtained via OS_GetLocalTime(). This allows for a more precise timeout than what is possible via the normal OS_SelectMultiple().

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.

Parameters

in,out	*ReadSet*	Set of handles to check/wait to become readable
in,out	*WriteSet*	Set of handles to check/wait to become writable
in	*abs_timeout*	The absolute time that the call may block until

See also

OS_SelectMultiple()

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	If any handle in the ReadSet or WriteSet is readable or writable, respectively
OS_ERROR_TIMEOUT	If no handles in the ReadSet or WriteSet became readable or writable within the timeout
OS_ERR_OPERATION_NOT_SUPPORTED	if a specified handle does not support select
OS_ERR_INVALID_ID	if no valid handles were contained in the ReadSet/WriteSet

5.23.2.7 OS_SelectSingle()

```
int32 OS_SelectSingle (
        osal_id_t 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 (OS_STREAM_STATE_READABLE and/or OS_STREAM_STATE↩ _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.

The timeout is expressed in milliseconds, relative to the time that the API was invoked. Use OS_SelectSingleAbs() for higher timing precision.

Parameters

in	*objid*	The handle ID to select on
in,out	*StateFlags*	State flag(s) (readable or writable) (must not be null)
in	*msecs*	Indicates the timeout. Positive values will wait up to that many milliseconds. Zero will not wait (poll). Negative values will wait forever (pend)

See also

OS_SelectSingleAbs()

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	If the handle is readable and/or writable, as requested
OS_ERROR_TIMEOUT	If the handle did not become readable or writable within the timeout
OS_INVALID_POINTER	if argument is NULL
OS_ERR_INVALID_ID	if the objid is not a valid handle

5.23.2.8 OS_SelectSingleAbs()

```
int32 OS_SelectSingleAbs (
    osal_id_t objid,
    uint32 * StateFlags,
    OS_time_t abs_timeout )
```

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 (OS_STREAM_STATE_READABLE and/or OS_STREAM_STATE↩ _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.

This API is identical to OS_SelectSingle() except for the timeout parameter. In this call, timeout is expressed as an absolute value of the OS clock, in the same time domain as obtained via OS_GetLocalTime(). This allows for a more precise timeout than what is possible via the normal OS_SelectSingle().

Parameters

in	*objid*	The handle ID to select on
in,out	*StateFlags*	State flag(s) (readable or writable) (must not be null)
in	*abs_timeout*	The absolute time that the call may block until

See also

OS_SelectSingle()

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	If the handle is readable and/or writable, as requested
OS_ERROR_TIMEOUT	If the handle did not become readable or writable within the timeout
OS_INVALID_POINTER	if argument is NULL
OS_ERR_INVALID_ID	if the objid is not a valid handle

5.24 OSAL Shell APIs

Functions

- int32 OS_ShellOutputToFile (const char ∗Cmd, osal_id_t filedes)

  Executes the command and sends output to a file.

5.24.1 Detailed Description

5.24.2 Function Documentation

5.24.2.1 OS_ShellOutputToFile()

```
int32 OS_ShellOutputToFile (
        const char * Cmd,
        osal_id_t 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 (must not be null)
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_INVALID_POINTER	if Cmd argument is NULL
OS_ERR_INVALID_ID	if the file descriptor passed in is invalid

5.25 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, size_t 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.

5.25.1 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 the "common denominator" to all address types.

5.25.2 Function Documentation

5.25.2.1 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 (must not be null)
in	*string*	The string to initialize the address from (must not be null)

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if argument is NULL
OS_ERROR	if the string cannot be converted to an address

5.25.2.2 OS_SocketAddrGetPort()

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

Get the port number of a network address.

For network protocols 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 (must not be null)
in	*Addr*	The network address buffer (must not be null)

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if argument is NULL
OS_ERR_BAD_ADDRESS	if the address domain is not compatible

5.25.2.3 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 (must not be null)
in	*Domain*	The address family

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if Addr argument is NULL
OS_ERR_NOT_IMPLEMENTED	if the system does not implement the requested domain

5.25.2.4 OS_SocketAddrSetPort()

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

Set the port number of a network address.

For network protocols 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

out	*Addr*	The network address buffer (must not be null)
in	*PortNum*	The port number to set

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if argument is NULL
OS_ERR_BAD_ADDRESS	if the address domain is not compatible

5.25.2.5 OS_SocketAddrToString()

```
int32 OS_SocketAddrToString (
        char * buffer,
        size_t 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 (must not be null)
in	*buflen*	Maximum length of the output string (must not be zero)
in	*Addr*	The network address buffer to convert (must not be null)

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if argument is NULL
OS_ERR_INVALID_SIZE	if passed-in buflen is not valid
OS_ERROR	if the address cannot be converted to string, or string buffer too small

5.26 OSAL Socket Management APIs

Functions

- int32 OS_SocketOpen (osal_id_t *sock_id, OS_SocketDomain_t Domain, OS_SocketType_t Type)

  Opens a socket.
- int32 OS_SocketBind (osal_id_t sock_id, const OS_SockAddr_t *Addr)

  Binds a socket to a given local address and enter listening (server) mode.
- int32 OS_SocketListen (osal_id_t sock_id)

  Places the specified socket into a listening state.
- int32 OS_SocketBindAddress (osal_id_t sock_id, const OS_SockAddr_t *Addr)

  Binds a socket to a given local address.
- int32 OS_SocketConnectAbs (osal_id_t sock_id, const OS_SockAddr_t *Addr, OS_time_t abs_timeout)

  Connects a socket to a given remote address.
- int32 OS_SocketConnect (osal_id_t sock_id, const OS_SockAddr_t *Addr, int32 timeout)

  Connects a socket to a given remote address.
- int32 OS_SocketShutdown (osal_id_t sock_id, OS_SocketShutdownMode_t Mode)

  Implement graceful shutdown of a stream socket.
- int32 OS_SocketAcceptAbs (osal_id_t sock_id, osal_id_t *connsock_id, OS_SockAddr_t *Addr, OS_time_t abs_timeout)

  Waits for and accept the next incoming connection on the given socket.
- int32 OS_SocketAccept (osal_id_t sock_id, osal_id_t *connsock_id, OS_SockAddr_t *Addr, int32 timeout)

  Waits for and accept the next incoming connection on the given socket.
- int32 OS_SocketRecvFromAbs (osal_id_t sock_id, void *buffer, size_t buflen, OS_SockAddr_t *RemoteAddr, OS_time_t abs_timeout)

  Reads data from a message-oriented (datagram) socket.
- int32 OS_SocketRecvFrom (osal_id_t sock_id, void *buffer, size_t buflen, OS_SockAddr_t *RemoteAddr, int32 timeout)

  Reads data from a message-oriented (datagram) socket.
- int32 OS_SocketSendTo (osal_id_t sock_id, const void *buffer, size_t buflen, const OS_SockAddr_t *RemoteAddr)

  Sends data to a message-oriented (datagram) socket.
- int32 OS_SocketGetIdByName (osal_id_t *sock_id, const char *sock_name)

  Gets an OSAL ID from a given name.
- int32 OS_SocketGetInfo (osal_id_t sock_id, OS_socket_prop_t *sock_prop)

  Gets information about an OSAL Socket ID.

5.26.1 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.

5.26.2 Function Documentation

5.26.2.1 OS_SocketAccept()

```
int32 OS_SocketAccept (
    osal_id_t sock_id,
    osal_id_t * 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.

The timeout is expressed in milliseconds, relative to the time that the API was invoked. Use OS_SocketAcceptAbs() for higher timing precision.

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 (must not be null)
in	*Addr*	The remote address of the incoming connection (must not be null)
in	*timeout*	The maximum amount of time to wait, or OS_PEND to wait forever

See also

OS_SocketAcceptAbs()

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if argument is NULL
OS_ERR_INVALID_ID	if the sock_id parameter is not valid
OS_ERR_INCORRECT_OBJ_TYPE	if the handle is not a socket
OS_ERR_INCORRECT_OBJ_STATE	if the socket is not bound or already connected

5.26.2.2 OS_SocketAcceptAbs()

```
int32 OS_SocketAcceptAbs (
    osal_id_t sock_id,
    osal_id_t * connsock_id,
    OS_SockAddr_t * Addr,
    OS_time_t abs_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.

This API is identical to OS_SocketAccept() except for the timeout parameter. In this call, timeout is expressed as an absolute value of the OS clock, in the same time domain as obtained via OS_GetLocalTime(). This allows for a more precise timeout than what is possible via the normal OS_SocketAccept().

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 (must not be null)
in	*Addr*	The remote address of the incoming connection (must not be null)
in	*abs_timeout*	The absolute time that the call may block until

See also

OS_SocketAccept()

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if argument is NULL
OS_ERR_INVALID_ID	if the sock_id parameter is not valid
OS_ERR_INCORRECT_OBJ_TYPE	if the handle is not a socket
OS_ERR_INCORRECT_OBJ_STATE	if the socket is not bound or already connected

5.26.2.3 OS_SocketBind()

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

Binds a socket to a given local address and enter listening (server) mode.

This is a convenience/compatibility routine to perform both OS_SocketBindAddress() and OS_SocketListen() operations in a single call, intended to simplify the setup for a server role.

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

Parameters

in	*sock_id*	The socket ID
in	*Addr*	The local address to bind to (must not be null)

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the sock_id parameter is not valid
OS_INVALID_POINTER	if argument is NULL
OS_ERR_INCORRECT_OBJ_STATE	if the socket is already bound
OS_ERR_INCORRECT_OBJ_TYPE	if the handle is not a socket

5.26.2.4 OS_SocketBindAddress()

```
int32 OS_SocketBindAddress (
    osal_id_t 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. This controls the source address reflected in network traffic transmitted via this socket.

After binding to the address, a stream socket may be followed by a call to either OS_SocketListen() for a server role or to OS_SocketConnect() for a client role.

Parameters

in	*sock↩ _id*	The socket ID
in	*Addr*	The local address to bind to (must not be null)

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the sock_id parameter is not valid
OS_INVALID_POINTER	if argument is NULL
OS_ERR_INCORRECT_OBJ_STATE	if the socket is already bound
OS_ERR_INCORRECT_OBJ_TYPE	if the handle is not a socket

5.26.2.5 OS_SocketConnect()

```
int32 OS_SocketConnect (
    osal_id_t 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).

The timeout is expressed in milliseconds, relative to the time that the API was invoked. Use OS_SocketConnectAbs() for higher timing precision.

Parameters

in	*sock↩ _id*	The socket ID
in	*Addr*	The remote address to connect to (must not be null)
in	*timeout*	The maximum amount of time to wait, or OS_PEND to wait forever

See also

OS_SocketConnectAbs()

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INCORRECT_OBJ_STATE	if the socket is already connected
OS_ERR_INVALID_ID	if the sock_id parameter is not valid
OS_ERR_INCORRECT_OBJ_TYPE	if the handle is not a socket
OS_INVALID_POINTER	if Addr argument is NULL

5.26.2.6 OS_SocketConnectAbs()

```
int32 OS_SocketConnectAbs (
    osal_id_t sock_id,
    const OS_SockAddr_t * Addr,
    OS_time_t abs_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).

This API is identical to OS_SocketConnect() except for the timeout parameter. In this call, timeout is expressed as an absolute value of the OS clock, in the same time domain as obtained via OS_GetLocalTime(). This allows for a more precise timeout than what is possible via the normal OS_SocketConnect().

Parameters

in	*sock_id*	The socket ID
in	*Addr*	The remote address to connect to (must not be null)
in	*abs_timeout*	The absolute time that the call may block until

See also

OS_SocketConnect()

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INCORRECT_OBJ_STATE	if the socket is already connected
OS_ERR_INVALID_ID	if the sock_id parameter is not valid
OS_ERR_INCORRECT_OBJ_TYPE	if the handle is not a socket
OS_INVALID_POINTER	if Addr argument is NULL

5.26.2.7 OS_SocketGetIdByName()

```
int32 OS_SocketGetIdByName (
    osal_id_t * 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 (must not be null)
in	*sock_name*	Name of socket to find (must not be null)

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	name length including null terminator greater than OS_MAX_API_NAME
OS_ERR_NAME_NOT_FOUND	if the name was not found in the table

5.26.2.8 OS_SocketGetInfo()

```
int32 OS_SocketGetInfo (
```

```
            osal_id_t 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 (must not be null)

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

5.26.2.9 OS_SocketListen() `int32 OS_SocketListen (`

```
            osal_id_t sock_id )
```

Places the specified socket into a listening state.

This function only applies to connection-oriented (stream) sockets that are intended to be used in a server-side role. This places the socket into a state where it can accept incoming connections from clients.

Parameters

in	*sock_id*	The socket ID

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the sock_id parameter is not valid
OS_ERR_INCORRECT_OBJ_STATE	if the socket is already listening
OS_ERR_INCORRECT_OBJ_TYPE	if the handle is not a stream socket

5.26.2.10 OS_SocketOpen()

```
int32 OS_SocketOpen (
    osal_id_t * 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 non-zero OSAL ID (must not be null)
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

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if argument is NULL
OS_ERR_NOT_IMPLEMENTED	if the system does not implement the requested socket/address domain

5.26.2.11 OS_SocketRecvFrom()

```
int32 OS_SocketRecvFrom (
    osal_id_t sock_id,
    void * buffer,
    size_t 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.

The timeout is expressed in milliseconds, relative to the time that the API was invoked. Use OS_SocketRecvFromAbs() for higher timing precision.

Parameters

in	*sock_id*	The socket ID, previously bound using OS_SocketBind()

Parameters

out	*buffer*	Pointer to message data receive buffer (must not be null)
in	*buflen*	The maximum length of the message data to receive (must not be zero)
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

See also

OS_SocketRecvFromAbs()

Returns

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

Return values

OS_INVALID_POINTER	if argument is NULL
OS_ERR_INVALID_SIZE	if passed-in buflen is not valid
OS_ERR_INVALID_ID	if the sock_id parameter is not valid
OS_ERR_INCORRECT_OBJ_TYPE	if the handle is not a socket

5.26.2.12 OS_SocketRecvFromAbs()

```
int32 OS_SocketRecvFromAbs (
    osal_id_t sock_id,
    void * buffer,
    size_t buflen,
    OS_SockAddr_t * RemoteAddr,
    OS_time_t abs_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.

This API is identical to OS_SocketRecvFrom() except for the timeout parameter. In this call, timeout is expressed as an absolute value of the OS clock, in the same time domain as obtained via OS_GetLocalTime(). This allows for a more precise timeout than what is possible via the normal OS_SocketRecvFrom().

Parameters

in	*sock_id*	The socket ID, previously bound using OS_SocketBind()
out	*buffer*	Pointer to message data receive buffer (must not be null)
in	*buflen*	The maximum length of the message data to receive (must not be zero)
out	*RemoteAddr*	Buffer to store the remote network address (may be NULL)
in	*abs_timeout*	The absolute time at which the call should return if nothing received

Returns

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

Return values

OS_INVALID_POINTER	if argument is NULL
OS_ERR_INVALID_SIZE	if passed-in buflen is not valid
OS_ERR_INVALID_ID	if the sock_id parameter is not valid
OS_ERR_INCORRECT_OBJ_TYPE	if the handle is not a socket

5.26.2.13 OS_SocketSendTo()

```
int32 OS_SocketSendTo (
    osal_id_t sock_id,
    const void * buffer,
    size_t 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 (must not be null)
in	*buflen*	The length of the message data to send (must not be zero)
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

Return values

OS_INVALID_POINTER	if argument is NULL
OS_ERR_INVALID_SIZE	if passed-in buflen is not valid
OS_ERR_INVALID_ID	if the sock_id parameter is not valid
OS_ERR_INCORRECT_OBJ_TYPE	if the handle is not a socket

5.26.2.14 OS_SocketShutdown()

```
int32 OS_SocketShutdown (
```

```
    osal_id_t sock_id,
    OS_SocketShutdownMode_t Mode )
```

Implement graceful shutdown of a stream socket.

This can be utilized to indicate the end of data stream without immediately closing the socket, giving the remote side an indication that the data transfer is complete.

Parameters

in	*sock↩ _id*	The socket ID
in	*Mode*	Whether to shutdown reading, writing, or both.

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the sock_id parameter is not valid
OS_ERR_INVALID_ARGUMENT	if the Mode argument is not one of the valid options
OS_ERR_INCORRECT_OBJ_TYPE	if the handle is not a socket
OS_ERR_INCORRECT_OBJ_STATE	if the socket is not connected

5.27 OSAL Task APIs

Functions

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

  Creates a task and starts running it.
- int32 OS_TaskDelete (osal_id_t 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 (osal_id_t task_id, osal_priority_t new_priority)

  Sets the given task to a new priority.
- osal_id_t OS_TaskGetId (void)

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

Find an existing task ID by name.

- int32 OS_TaskGetInfo (osal_id_t task_id, OS_task_prop_t *task_prop)

  Fill a property object buffer with details regarding the resource.

- int32 OS_TaskFindIdBySystemData (osal_id_t *task_id, const void *sysdata, size_t sysdata_size)

  Reverse-lookup the OSAL task ID from an operating system ID.

5.27.1 Detailed Description

5.27.2 Function Documentation

5.27.2.1 OS_TaskCreate()

```
int32 OS_TaskCreate (
    osal_id_t * task_id,
    const char * task_name,
    osal_task_entry function_pointer,
    osal_stackptr_t stack_pointer,
    size_t stack_size,
    osal_priority_t 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.

Portable applications should always specify the actual stack size in the stack_size parameter, not 0. This size value is not enforced/checked by OSAL, but is simply passed through to the RTOS for stack creation. Some RTOS implementations may assume 0 means a default stack size while others may actually create a task with no stack.

Unlike stack_size, the stack_pointer is optional and can be specified as NULL. In that case, a stack of the requested size will be dynamically allocated from the system heap.

Parameters

out	*task_id*	will be set to the non-zero ID of the newly-created resource (must not be null)
in	*task_name*	the name of the new resource to create (must not be null)
in	*function_pointer*	the entry point of the new task (must not be null)
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 (must not be zero)
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_INVALID_SIZE	if the stack_size argument is zero
OS_ERR_NAME_TOO_LONG	name length including null terminator greater than OS_MAX_API_NAME
OS_ERR_INVALID_PRIORITY	if the priority is bad (return value only verified in coverage test)
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 (return value only verified in coverage test)

5.27.2.2 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. This is a scheduled wait (clock_nanosleep/rtems_task_wake_after/taskDelay), not a "busy" wait.

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 an unspecified/other error occurs (return value only verified in coverage test)

5.27.2.3 OS_TaskDelete()

```
int32 OS_TaskDelete (
        osal_id_t 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 (return value only verified in coverage test)

5.27.2.4 OS_TaskExit()

```
void OS_TaskExit (
    void  )
```

Exits the calling task.

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

5.27.2.5 OS_TaskFindIdBySystemData()

```
int32 OS_TaskFindIdBySystemData (
    osal_id_t * task_id,
    const void * sysdata,
    size_t sysdata_size )
```

Reverse-lookup the OSAL task ID from an operating system ID.

This provides a method by which an external entity may find the OSAL task ID corresponding to a system-defined identifier (e.g. TASK_ID, pthread_t, rtems_id, etc).

Normally OSAL does not expose the underlying OS-specific values to the application, but in some circumstances, such as exception handling, the OS may provide this information directly to a BSP handler outside of the normal OSAL API.

Parameters

out	*task_id*	The buffer where the task id output is stored (must not be null)
in	*sysdata*	Pointer to the system-provided identification data
in	*sysdata_size*	Size of the system-provided identification data

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution. (return value only verified in coverage test)
OS_INVALID_POINTER	if a pointer argument is NULL

5.27.2.6 OS_TaskGetId()

```
osal_id_t 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)

5.27.2.7 OS_TaskGetIdByName()

```
int32 OS_TaskGetIdByName (
    osal_id_t * 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 (must not be null)

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	name length including null terminator greater than OS_MAX_API_NAME
OS_ERR_NAME_NOT_FOUND	if the name wasn't found in the table

5.27.2.8 OS_TaskGetInfo()

```
int32 OS_TaskGetInfo (
        osal_id_t 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 (must not be null)

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

5.27.2.9 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

Return values

OS_ERR_INVALID_ID	if the calling context is not an OSAL task

5.27.2.10 OS_TaskSetPriority()

```
int32 OS_TaskSetPriority (
    osal_id_t task_id,
    osal_priority_t 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 (return value only verified in coverage test)
OS_ERROR	if an unspecified/other error occurs (return value only verified in coverage test)

5.28 OSAL Time Base APIs

Functions

- int32 OS_TimeBaseCreate (osal_id_t *timebase_id, const char *timebase_name, OS_TimerSync_t external_↩ sync)

  Create an abstract Time Base resource.
- int32 OS_TimeBaseSet (osal_id_t timebase_id, uint32 start_time, uint32 interval_time)

  Sets the tick period for simulated time base objects.
- int32 OS_TimeBaseDelete (osal_id_t timebase_id)

  Deletes a time base object.
- int32 OS_TimeBaseGetIdByName (osal_id_t *timebase_id, const char *timebase_name)

  Find the ID of an existing time base resource.
- int32 OS_TimeBaseGetInfo (osal_id_t timebase_id, OS_timebase_prop_t *timebase_prop)

  Obtain information about a timebase resource.
- int32 OS_TimeBaseGetFreeRun (osal_id_t timebase_id, uint32 *freerun_val)

  Read the value of the timebase free run counter.

5.28.1 Detailed Description

5.28.2 Function Documentation

5.28.2.1 OS_TimeBaseCreate()

```
int32 OS_TimeBaseCreate (
    osal_id_t * 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.

This configuration API must not be used from the context of a timer callback. Timers should only be configured from the context of normal OSAL tasks.

Parameters

out	*timebase_id*	will be set to the non-zero ID of the newly-created resource (must not be null)
in	*timebase_name*	The name of the time base (must not be null)
in	*external_sync*	A synchronization function for BSP hardware-based timer ticks

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_ERR_NAME_TAKEN	if the name specified is already used
OS_ERR_NO_FREE_IDS	if there can be no more timebase resources created
OS_ERR_INCORRECT_OBJ_STATE	if called from timer/timebase context
OS_ERR_NAME_TOO_LONG	if the timebase_name is too long
OS_INVALID_POINTER	if a pointer argument is NULL

5.28.2.2 OS_TimeBaseDelete()

```
int32 OS_TimeBaseDelete (
        osal_id_t timebase_id )
```

Deletes a time base object.

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

Note

This configuration API must not be used from the context of a timer callback. Timers should only be configured from the context of normal OSAL tasks.

Parameters

in	*timebase_id*	The timebase resource 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 timebase
OS_ERR_INCORRECT_OBJ_STATE	if called from timer/timebase context

5.28.2.3 OS_TimeBaseGetFreeRun()

```
int32 OS_TimeBaseGetFreeRun (
        osal_id_t 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 (must not be null)

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 pointer argument is NULL

5.28.2.4 OS_TimeBaseGetIdByName()

```
int32 OS_TimeBaseGetIdByName (
    osal_id_t * 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.

Note

This configuration API must not be used from the context of a timer callback. Timers should only be configured from the context of normal OSAL tasks.

Parameters

out	*timebase_id*	will be set to the non-zero ID of the matching resource (must not be null)
in	*timebase_name*	The name of the timebase resource to find (must not be null)

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	name length including null terminator greater than OS_MAX_API_NAME
OS_ERR_NAME_NOT_FOUND	if the name was not found in the table
OS_ERR_INCORRECT_OBJ_STATE	if called from timer/timebase context

5.28.2.5 OS_TimeBaseGetInfo()

```
int32 OS_TimeBaseGetInfo (
    osal_id_t 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.

Note

This configuration API must not be used from the context of a timer callback. Timers should only be configured from the context of normal OSAL tasks.

Parameters

in	*timebase_id*	The timebase resource ID
out	*timebase_prop*	Buffer to store timebase properties (must not be null)

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
OS_ERR_INCORRECT_OBJ_STATE	if called from timer/timebase context

5.28.2.6 OS_TimeBaseSet()

```
int32 OS_TimeBaseSet (
        osal_id_t 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.

Note

This configuration API must not be used from the context of a timer callback. Timers should only be configured from the context of normal OSAL tasks.

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

Return values

OS_SUCCESS	Successful execution.
OS_ERR_INVALID_ID	if the id passed in is not a valid timebase
OS_ERR_INCORRECT_OBJ_STATE	if called from timer/timebase context
OS_TIMER_ERR_INVALID_ARGS	if start_time or interval_time are out of range

5.29 OSAL Timer APIs

Functions

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

  Create a timer object.
- int32 OS_TimerAdd (osal_id_t *timer_id, const char *timer_name, osal_id_t timebase_id, OS_ArgCallback_t callback_ptr, void *callback_arg)

  Add a timer object based on an existing TimeBase resource.
- int32 OS_TimerSet (osal_id_t timer_id, uint32 start_time, uint32 interval_time)

  Configures a periodic or one shot timer.
- int32 OS_TimerDelete (osal_id_t timer_id)

  Deletes a timer resource.
- int32 OS_TimerGetIdByName (osal_id_t *timer_id, const char *timer_name)

  Locate an existing timer resource by name.
- int32 OS_TimerGetInfo (osal_id_t timer_id, OS_timer_prop_t *timer_prop)

  Gets information about an existing timer.

5.29.1 Detailed Description

5.29.2 Function Documentation

5.29.2.1 OS_TimerAdd()

```
int32 OS_TimerAdd (
    osal_id_t * timer_id,
    const char * timer_name,
    osal_id_t 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.

The callback function for this method should be declared according to the OS_ArgCallback_t function pointer type. The timer_id is passed in to the function by the OSAL, and the arg parameter is passed through from the callback_arg argument on this call.

Note

This configuration API must not be used from the context of a timer callback. Timers should only be configured from the context of normal OSAL tasks.

See also

OS_ArgCallback_t

Parameters

out	*timer_id*	Will be set to the non-zero resource ID of the timer object (must not be null)
in	*timer_name*	Name of the timer object (must not be null)
in	*timebase_id*	The time base resource to use as a reference
in	*callback_ptr*	Application-provided function to invoke (must not be null)
in	*callback_arg*	Opaque argument to pass to callback function, may be NULL

Returns

Execution status, see OSAL Return Code Defines

Return values

OS_SUCCESS	Successful execution.
OS_INVALID_POINTER	if any parameters are NULL
OS_ERR_INVALID_ID	if the timebase_id parameter is not valid
OS_ERR_NAME_TOO_LONG	name length including null terminator greater than OS_MAX_API_NAME
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_ERR_INCORRECT_OBJ_STATE	if invoked from a timer context
OS_TIMER_ERR_INTERNAL	if there was an error programming the OS timer (return value only verified in coverage test)

5.29.2.2 OS_TimerCreate()

```
int32 OS_TimerCreate (
    osal_id_t * 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.

The callback function should be declared according to the OS_TimerCallback_t function pointer type. The timer_id value is passed to the callback function.

Note

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

This configuration API must not be used from the context of a timer callback. Timers should only be configured from the context of normal OSAL tasks.

See also

OS_TimerCallback_t

Parameters

out	*timer_id*	Will be set to the non-zero resource ID of the timer object (must not be null)
in	*timer_name*	Name of the timer object (must not be null)
out	*clock_accuracy*	Expected precision of the timer, in microseconds. This is the underlying tick value rounded to the nearest microsecond integer. (must not be null)
in	*callback_ptr*	The function pointer of the timer callback (must not be null).

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	name length including null terminator greater than OS_MAX_API_NAME
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_ERR_INCORRECT_OBJ_STATE	if invoked from a timer context
OS_TIMER_ERR_INTERNAL	if there was an error programming the OS timer (return value only verified in coverage test)

5.29.2.3 OS_TimerDelete() `int32 OS_TimerDelete ( osal_id_t timer_id )`

Deletes a timer resource.

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

Note

This configuration API must not be used from the context of a timer callback. Timers should only be configured from the context of normal OSAL tasks.

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 (return value only verified in coverage test)
OS_ERR_INCORRECT_OBJ_STATE	if called from timer/timebase context

5.29.2.4 OS_TimerGetIdByName()

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

Locate an existing timer resource by name.

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

Note

This configuration API must not be used from the context of a timer callback. Timers should only be configured from the context of normal OSAL tasks.

Parameters

out	*timer_id*	Will be set to the timer ID corresponding to the name (must not be null)
in	*timer_name*	The timer name to find (must not be null)

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	name length including null terminator greater than OS_MAX_API_NAME

Return values

OS_ERR_NAME_NOT_FOUND	if the name was not found in the table
OS_ERR_INCORRECT_OBJ_STATE	if called from timer/timebase context

5.29.2.5 OS_TimerGetInfo()

```
int32 OS_TimerGetInfo (
        osal_id_t 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.

Note

This configuration API must not be used from the context of a timer callback. Timers should only be configured from the context of normal OSAL tasks.

Parameters

in	*timer_id*	The timer ID to operate on
out	*timer_prop*	Buffer containing timer properties (must not be null) • 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
OS_ERR_INCORRECT_OBJ_STATE	if called from timer/timebase context

5.29.2.6 OS_TimerSet()

```
int32 OS_TimerSet (
        osal_id_t 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.

This configuration API must not be used from the context of a timer callback. Timers should only be configured from the context of normal OSAL tasks.

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 (return value only verified in coverage test)
OS_ERR_INCORRECT_OBJ_STATE	if called from timer/timebase context
OS_TIMER_ERR_INVALID_ARGS	if the start_time or interval_time is out of range, or both 0

6 Data Structure Documentation

6.1 OS_bin_sem_prop_t Struct Reference

OSAL binary semaphore properties.

`#include <osapi-binsem.h>`

Data Fields

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

6.1.1 Detailed Description

OSAL binary semaphore properties.

Definition at line 39 of file osapi-binsem.h.

6.1.2 Field Documentation

6.1.2.1 creator `osal_id_t OS_bin_sem_prop_t::creator`

Definition at line 42 of file osapi-binsem.h.

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

Definition at line 41 of file osapi-binsem.h.

6.1.2.3 value `int32 OS_bin_sem_prop_t::value`

Definition at line 43 of file osapi-binsem.h.

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

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

6.2 OS_condvar_prop_t Struct Reference

OSAL condition variable properties.

`#include <osapi-condvar.h>`

Data Fields

- char name [OS_MAX_API_NAME]
- osal_id_t creator

6.2.1 Detailed Description

OSAL condition variable properties.

Definition at line 34 of file osapi-condvar.h.

6.2.2 Field Documentation

6.2.2.1 creator `osal_id_t OS_condvar_prop_t::creator`

Definition at line 37 of file osapi-condvar.h.

6.2.2.2 name `char OS_condvar_prop_t::name[OS_MAX_API_NAME]`

Definition at line 36 of file osapi-condvar.h.

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

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

6.3 OS_count_sem_prop_t Struct Reference

OSAL counting semaphore properties.

`#include <osapi-countsem.h>`

Data Fields

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

6.3.1 Detailed Description

OSAL counting semaphore properties.

Definition at line 32 of file osapi-countsem.h.

6.3.2 Field Documentation

6.3.2.1 creator `osal_id_t OS_count_sem_prop_t::creator`

Definition at line 35 of file osapi-countsem.h.

6.3.2.2 name `char OS_count_sem_prop_t::name[OS_MAX_API_NAME]`

Definition at line 34 of file osapi-countsem.h.

6.3.2.3 value `int32 OS_count_sem_prop_t::value`

Definition at line 36 of file osapi-countsem.h.

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

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

6.4 os_dirent_t Struct Reference

Directory entry.

`#include <osapi-dir.h>`

Data Fields

- char FileName [OS_MAX_FILE_NAME]

6.4.1 Detailed Description

Directory entry.

Definition at line 32 of file osapi-dir.h.

6.4.2 Field Documentation

6.4.2.1 FileName `char os_dirent_t::FileName[OS_MAX_FILE_NAME]`

Definition at line 34 of file osapi-dir.h.

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

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

6.5 OS_FdSet Struct Reference

An abstract structure capable of holding several OSAL IDs.

`#include <osapi-select.h>`

Data Fields

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

6.5.1 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.

Note: Math is to determine uint8 array size needed to represent single bit OS_MAX_NUM_OPEN_FILES objects, + 7 rounds up and 8 is the size of uint8.

See also

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

Definition at line 44 of file osapi-select.h.

6.5.2 Field Documentation

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

Definition at line 46 of file osapi-select.h.

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

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

6.6 OS_file_prop_t Struct Reference

OSAL file properties.

`#include <osapi-file.h>`

Data Fields

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

6.6.1 Detailed Description

OSAL file properties.

Definition at line 49 of file osapi-file.h.

6.6.2 Field Documentation

6.6.2.1 IsValid `uint8 OS_file_prop_t::IsValid`

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

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

Definition at line 51 of file osapi-file.h.

6.6.2.3 User `osal_id_t OS_file_prop_t::User`

Definition at line 52 of file osapi-file.h.

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

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

6.7 os_fsinfo_t Struct Reference

OSAL file system info.

`#include <osapi-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.

6.7.1 Detailed Description

OSAL file system info.

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

6.7.2 Field Documentation

6.7.2.1 FreeFds `uint32 os_fsinfo_t::FreeFds`

Total number that are free.

Definition at line 38 of file osapi-filesys.h.

6.7.2.2 FreeVolumes `uint32 os_fsinfo_t::FreeVolumes`

Total number of volumes free.

Definition at line 40 of file osapi-filesys.h.

6.7.2.3 MaxFds `uint32 os_fsinfo_t::MaxFds`

Total number of file descriptors.

Definition at line 37 of file osapi-filesys.h.

6.7.2.4 MaxVolumes `uint32 os_fsinfo_t::MaxVolumes`

Maximum number of volumes.

Definition at line 39 of file osapi-filesys.h.

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

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

6.8 os_fstat_t Struct Reference

File system status.

`#include <osapi-file.h>`

Data Fields

- uint32 FileModeBits
- OS_time_t FileTime
- size_t FileSize

6.8.1 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 64 of file osapi-file.h.

6.8.2 Field Documentation

6.8.2.1 FileModeBits `uint32 os_fstat_t::FileModeBits`

Definition at line 66 of file osapi-file.h.

6.8.2.2 FileSize `size_t os_fstat_t::FileSize`

Definition at line 68 of file osapi-file.h.

6.8.2.3 FileTime `OS_time_t os_fstat_t::FileTime`

Definition at line 67 of file osapi-file.h.

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

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

6.9 OS_heap_prop_t Struct Reference

OSAL heap properties.

`#include <osapi-heap.h>`

Data Fields

- size_t free_bytes
- osal_blockcount_t free_blocks
- size_t largest_free_block

6.9.1 Detailed Description

OSAL heap properties.

See also

OS_HeapGetInfo()

Definition at line 36 of file osapi-heap.h.

6.9.2 Field Documentation

6.9.2.1 free_blocks `osal_blockcount_t OS_heap_prop_t::free_blocks`

Definition at line 39 of file osapi-heap.h.

6.9.2.2 free_bytes `size_t OS_heap_prop_t::free_bytes`

Definition at line 38 of file osapi-heap.h.

6.9.2.3 largest_free_block `size_t OS_heap_prop_t::largest_free_block`

Definition at line 40 of file osapi-heap.h.

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

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

6.10 OS_module_address_t Struct Reference

OSAL module address properties.

`#include <osapi-module.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

6.10.1 Detailed Description

OSAL module address properties.

Definition at line 78 of file osapi-module.h.

6.10.2 Field Documentation

6.10.2.1 bss_address `cpuaddr OS_module_address_t::bss_address`

Definition at line 86 of file osapi-module.h.

6.10.2.2 bss_size `cpuaddr OS_module_address_t::bss_size`

Definition at line 87 of file osapi-module.h.

6.10.2.3 code_address `cpuaddr OS_module_address_t::code_address`

Definition at line 82 of file osapi-module.h.

6.10.2.4 code_size `cpuaddr OS_module_address_t::code_size`

Definition at line 83 of file osapi-module.h.

6.10.2.5 data_address cpuaddr OS_module_address_t::data_address

Definition at line 84 of file osapi-module.h.

6.10.2.6 data_size cpuaddr OS_module_address_t::data_size

Definition at line 85 of file osapi-module.h.

6.10.2.7 flags uint32 OS_module_address_t::flags

Definition at line 81 of file osapi-module.h.

6.10.2.8 valid uint32 OS_module_address_t::valid

Definition at line 80 of file osapi-module.h.

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

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

6.11 OS_module_prop_t Struct Reference

OSAL module properties.

`#include <osapi-module.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

6.11.1 Detailed Description

OSAL module properties.

Definition at line 91 of file osapi-module.h.

6.11.2 Field Documentation

6.11.2.1 addr OS_module_address_t OS_module_prop_t::addr

Definition at line 97 of file osapi-module.h.

6.11.2.2 entry_point cpuaddr OS_module_prop_t::entry_point

Definition at line 93 of file osapi-module.h.

6.11.2.3 filename char OS_module_prop_t::filename[OS_MAX_PATH_LEN]

Definition at line 95 of file osapi-module.h.

6.11.2.4 host_module_id cpuaddr OS_module_prop_t::host_module_id

Definition at line 94 of file osapi-module.h.

6.11.2.5 name char OS_module_prop_t::name[OS_MAX_API_NAME]

Definition at line 96 of file osapi-module.h.

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

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

6.12 OS_mut_sem_prop_t Struct Reference

OSAL mutex properties.

`#include <osapi-mutex.h>`

Data Fields

- char name [OS_MAX_API_NAME]
- osal_id_t creator

6.12.1 Detailed Description

OSAL mutex properties.

Definition at line 32 of file osapi-mutex.h.

6.12.2 Field Documentation

6.12.2.1 creator `osal_id_t OS_mut_sem_prop_t::creator`

Definition at line 35 of file osapi-mutex.h.

6.12.2.2 name `char OS_mut_sem_prop_t::name[OS_MAX_API_NAME]`

Definition at line 34 of file osapi-mutex.h.

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

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

6.13 OS_queue_prop_t Struct Reference

OSAL queue properties.

```
#include <osapi-queue.h>
```

Data Fields

- char name [OS_MAX_API_NAME]
- osal_id_t creator

6.13.1 Detailed Description

OSAL queue properties.

Definition at line 32 of file osapi-queue.h.

6.13.2 Field Documentation

6.13.2.1 creator `osal_id_t OS_queue_prop_t::creator`

Definition at line 35 of file osapi-queue.h.

6.13.2.2 name `char OS_queue_prop_t::name[OS_MAX_API_NAME]`

Definition at line 34 of file osapi-queue.h.

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

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

6.14 OS_SockAddr_t Struct Reference

Encapsulates a generic network address.

`#include <osapi-sockets.h>`

Data Fields

- size_t ActualLength
  Length of the actual address data.
- OS_SockAddrData_t AddrData
  Abstract Address data.

6.14.1 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 110 of file osapi-sockets.h.

6.14.2 Field Documentation

6.14.2.1 ActualLength `size_t OS_SockAddr_t::ActualLength`

Length of the actual address data.

Definition at line 112 of file osapi-sockets.h.

6.14.2.2 AddrData `OS_SockAddrData_t OS_SockAddr_t::AddrData`

Abstract Address data.

Definition at line 113 of file osapi-sockets.h.

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

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

6.15 OS_SockAddrData_t Union Reference

Storage buffer for generic network address.

`#include <osapi-sockets.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.

6.15.1 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 96 of file osapi-sockets.h.

6.15.2 Field Documentation

6.15.2.1 AlignPtr `void* OS_SockAddrData_t::AlignPtr`

Ensures pointer alignment.

Definition at line 100 of file osapi-sockets.h.

6.15.2.2 AlignU32 `uint32 OS_SockAddrData_t::AlignU32`

Ensures uint32 alignment.

Definition at line 99 of file osapi-sockets.h.

6.15.2.3 Buffer `uint8 OS_SockAddrData_t::Buffer[OS_SOCKADDR_MAX_LEN]`

Ensures length of at least OS_SOCKADDR_MAX_LEN.

Definition at line 98 of file osapi-sockets.h.

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

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

6.16 OS_socket_prop_t Struct Reference

Encapsulates socket properties.

`#include <osapi-sockets.h>`

Data Fields

- char name [OS_MAX_API_NAME]
  Name of the socket.
- osal_id_t creator
  OSAL TaskID which opened the socket.

6.16.1 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 123 of file osapi-sockets.h.

6.16.2 Field Documentation

6.16.2.1 creator `osal_id_t OS_socket_prop_t::creator`

OSAL TaskID which opened the socket.

Definition at line 126 of file osapi-sockets.h.

6.16.2.2 name `char OS_socket_prop_t::name[OS_MAX_API_NAME]`

Name of the socket.

Definition at line 125 of file osapi-sockets.h.

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

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

6.17 OS_static_symbol_record_t Struct Reference

Associates a single symbol name with a memory address.

`#include <osapi-module.h>`

Data Fields

- const char * Name
- void(* Address)(void)
- const char * Module

6.17.1 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 113 of file osapi-module.h.

6.17.2 Field Documentation

6.17.2.1 Address `void(* OS_static_symbol_record_t::Address) (void)`

Definition at line 116 of file osapi-module.h.

6.17.2.2 Module `const char* OS_static_symbol_record_t::Module`

Definition at line 117 of file osapi-module.h.

6.17.2.3 Name `const char* OS_static_symbol_record_t::Name`

Definition at line 115 of file osapi-module.h.

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

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

6.18 OS_statvfs_t Struct Reference

`#include <osapi-filesys.h>`

Data Fields

- size_t block_size
- osal_blockcount_t total_blocks
- osal_blockcount_t blocks_free

6.18.1 Detailed Description

Definition at line 49 of file osapi-filesys.h.

6.18.2 Field Documentation

6.18.2.1 block_size `size_t OS_statvfs_t::block_size`

Block size of underlying FS

Definition at line 51 of file osapi-filesys.h.

6.18.2.2 blocks_free `osal_blockcount_t OS_statvfs_t::blocks_free`

Available blocks in underlying FS

Definition at line 53 of file osapi-filesys.h.

6.18.2.3 total_blocks `osal_blockcount_t OS_statvfs_t::total_blocks`

Total blocks in underlying FS

Definition at line 52 of file osapi-filesys.h.

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

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

6.19 OS_task_prop_t Struct Reference

OSAL task properties.

`#include <osapi-task.h>`

Data Fields

- char name [OS_MAX_API_NAME]
- osal_id_t creator
- size_t stack_size
- osal_priority_t priority

6.19.1 Detailed Description

OSAL task properties.

Definition at line 57 of file osapi-task.h.

6.19.2 Field Documentation

6.19.2.1 creator `osal_id_t OS_task_prop_t::creator`

Definition at line 60 of file osapi-task.h.

6.19.2.2 name `char OS_task_prop_t::name[OS_MAX_API_NAME]`

Definition at line 59 of file osapi-task.h.

6.19.2.3 priority `osal_priority_t OS_task_prop_t::priority`

Definition at line 62 of file osapi-task.h.

6.19.2.4 stack_size `size_t OS_task_prop_t::stack_size`

Definition at line 61 of file osapi-task.h.

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

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

6.20 OS_time_t Struct Reference

OSAL time interval structure.

`#include <osapi-clock.h>`

Data Fields

- int64 ticks

6.20.1 Detailed Description

OSAL time interval structure.

This is used to represent a basic time interval.

When used with OS_GetLocalTime/OS_SetLocalTime, this represents the interval from the OS's epoch point, typically 01 Jan 1970 00:00:00 UTC on systems that have a persistent real time clock (RTC), or the system boot time if there is no RTC available.

Applications should not directly access fields within this structure, as the definition may change in future versions of OSAL. Instead, applications should use the accessor/conversion methods defined below.

Definition at line 45 of file osapi-clock.h.

6.20.2 Field Documentation

6.20.2.1 ticks `int64 OS_time_t::ticks`

Ticks elapsed since reference point

Definition at line 47 of file osapi-clock.h.

Referenced by OS_TimeAdd(), OS_TimeAssembleFromMicroseconds(), OS_TimeAssembleFromMilliseconds(), OS_TimeAssembleFromNanoseconds(), OS_TimeAssembleFromSubseconds(), OS_TimeEqual(), OS_TimeGet↩FractionalPart(), OS_TimeGetSign(), OS_TimeGetTotalMicroseconds(), OS_TimeGetTotalMilliseconds(), OS_Time↩GetTotalNanoseconds(), OS_TimeGetTotalSeconds(), and OS_TimeSubtract().

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

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

6.21 OS_timebase_prop_t Struct Reference

Time base properties.

`#include <osapi-timebase.h>`

Data Fields

- char name [OS_MAX_API_NAME]
- osal_id_t creator
- uint32 nominal_interval_time
- uint32 freerun_time
- uint32 accuracy

6.21.1 Detailed Description

Time base properties.

Definition at line 37 of file osapi-timebase.h.

6.21.2 Field Documentation

6.21.2.1 accuracy `uint32 OS_timebase_prop_t::accuracy`

Definition at line 43 of file osapi-timebase.h.

6.21.2.2 creator `osal_id_t OS_timebase_prop_t::creator`

Definition at line 40 of file osapi-timebase.h.

6.21.2.3 freerun_time `uint32 OS_timebase_prop_t::freerun_time`

Definition at line 42 of file osapi-timebase.h.

6.21.2.4 name `char OS_timebase_prop_t::name[OS_MAX_API_NAME]`

Definition at line 39 of file osapi-timebase.h.

6.21.2.5 nominal_interval_time `uint32 OS_timebase_prop_t::nominal_interval_time`

Definition at line 41 of file osapi-timebase.h.

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

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

6.22 OS_timer_prop_t Struct Reference

Timer properties.

`#include <osapi-timer.h>`

Data Fields

- char name [OS_MAX_API_NAME]
- osal_id_t creator
- uint32 start_time
- uint32 interval_time
- uint32 accuracy

6.22.1 Detailed Description

Timer properties.

Definition at line 37 of file osapi-timer.h.

6.22.2 Field Documentation

6.22.2.1 accuracy `uint32 OS_timer_prop_t::accuracy`

Definition at line 43 of file osapi-timer.h.

6.22.2.2 creator `osal_id_t OS_timer_prop_t::creator`

Definition at line 40 of file osapi-timer.h.

6.22.2.3 interval_time `uint32 OS_timer_prop_t::interval_time`

Definition at line 42 of file osapi-timer.h.

6.22.2.4 name `char OS_timer_prop_t::name[OS_MAX_API_NAME]`

Definition at line 39 of file osapi-timer.h.

6.22.2.5 start_time `uint32 OS_timer_prop_t::start_time`

Definition at line 41 of file osapi-timer.h.

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

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

7 File Documentation

7.1 build/osal_public_api/inc/osconfig.h File Reference

Macros

- #define OSAL_CONFIG_INCLUDE_DYNAMIC_LOADER

  Configuration file Operating System Abstraction Layer.
- #define OSAL_CONFIG_INCLUDE_NETWORK
- #define OSAL_CONFIG_INCLUDE_STATIC_LOADER
- #define OSAL_CONFIG_CONSOLE_ASYNC
- #define OS_MAX_TASKS 64

  The maximum number of to support.
- #define OS_MAX_QUEUES 64

  The maximum number of queues to support.
- #define OS_MAX_COUNT_SEMAPHORES 20

  The maximum number of counting semaphores to support.
- #define OS_MAX_BIN_SEMAPHORES 20

  The maximum number of binary semaphores to support.
- #define OS_MAX_MUTEXES 20

  The maximum number of mutexes to support.
- #define OS_MAX_CONDVARS 4

  The maximum number of condition variables to support.
- #define OS_MAX_MODULES 20

  The maximum number of modules to support.

- #define OS_MAX_TIMEBASES 5

  The maximum number of timebases to support.
- #define OS_MAX_TIMERS 10

  The maximum number of timer callbacks to support.
- #define OS_MAX_NUM_OPEN_FILES 50

  The maximum number of concurrently open files to support.
- #define OS_MAX_NUM_OPEN_DIRS 4

  The maximum number of concurrently open directories to support.
- #define OS_MAX_FILE_SYSTEMS 14

  The maximum number of file systems to support.
- #define OS_MAX_SYM_LEN 64

  The maximum length of symbols.
- #define OS_MAX_FILE_NAME 20

  The maximum length of OSAL file names.
- #define OS_MAX_PATH_LEN 64

  The maximum length of OSAL path names.
- #define OS_MAX_API_NAME 20

  The maximum length of OSAL resource names.
- #define OS_SOCKADDR_MAX_LEN 28

  The maximum size of the socket address structure.
- #define OS_BUFFER_SIZE 172

  The maximum size of output produced by a single OS_printf()
- #define OS_BUFFER_MSG_DEPTH 100

  The maximum number of OS_printf() output strings to buffer.
- #define OS_UTILITYTASK_PRIORITY 245

  Priority level of the background utility task.
- #define OS_UTILITYTASK_STACK_SIZE 2048

  The stack size of the background utility task.
- #define OS_MAX_CMD_LEN 1000

  The maximum size of a shell command.
- #define OS_QUEUE_MAX_DEPTH 50

  The maximum depth of OSAL queues.
- #define OS_SHELL_CMD_INPUT_FILE_NAME ""

  The name of the temporary file used to store shell commands.
- #define OS_PRINTF_CONSOLE_NAME ""

  The name of the primary console device.
- #define OS_ADD_TASK_FLAGS 0

  Flags added to all tasks on creation.
- #define OS_MAX_CONSOLES 1

  The maximum number of console devices to support.
- #define OS_MODULE_FILE_EXTENSION ".so"

  The system-specific file extension used on loadable module files.
- #define OS_FS_DEV_NAME_LEN 32
- #define OS_FS_PHYS_NAME_LEN 64
- #define OS_FS_VOL_NAME_LEN 32

7.1.1 Macro Definition Documentation

7.1.1.1 OS_ADD_TASK_FLAGS `#define OS_ADD_TASK_FLAGS 0`

Flags added to all tasks on creation.

Added to the task flags on creation

Supports adding floating point support for all tasks when the OS requires it

Definition at line 254 of file osconfig.h.

7.1.1.2 OS_BUFFER_MSG_DEPTH `#define OS_BUFFER_MSG_DEPTH 100`

The maximum number of OS_printf() output strings to buffer.

Based on the OSAL_CONFIG_PRINTF_BUFFER_DEPTH configuration option

Definition at line 187 of file osconfig.h.

7.1.1.3 OS_BUFFER_SIZE `#define OS_BUFFER_SIZE 172`

The maximum size of output produced by a single OS_printf()

Based on the OSAL_CONFIG_PRINTF_BUFFER_SIZE configuration option

Definition at line 180 of file osconfig.h.

7.1.1.4 OS_FS_DEV_NAME_LEN `#define OS_FS_DEV_NAME_LEN 32`

Device name length

Definition at line 281 of file osconfig.h.

7.1.1.5 OS_FS_PHYS_NAME_LEN `#define OS_FS_PHYS_NAME_LEN 64`

Physical drive name length

Definition at line 282 of file osconfig.h.

7.1.1.6 OS_FS_VOL_NAME_LEN `#define OS_FS_VOL_NAME_LEN 32`

Volume name length

Definition at line 283 of file osconfig.h.

7.1.1.7 OS_MAX_API_NAME `#define OS_MAX_API_NAME 20`

The maximum length of OSAL resource names.

Based on the OSAL_CONFIG_MAX_API_NAME configuration option

Note

This value must include a terminating NUL character

Definition at line 163 of file osconfig.h.

7.1.1.8 OS_MAX_BIN_SEMAPHORES `#define OS_MAX_BIN_SEMAPHORES 20`

The maximum number of binary semaphores to support.

Based on the OSAL_CONFIG_MAX_BIN_SEMAPHORES configuration option

Definition at line 65 of file osconfig.h.

7.1.1.9 OS_MAX_CMD_LEN `#define OS_MAX_CMD_LEN 1000`

The maximum size of a shell command.

This limit is only applicable if shell support is enabled.

Based on the OSAL_CONFIG_MAX_CMD_LEN configuration option

Note

This value must include a terminating NUL character

Definition at line 218 of file osconfig.h.

7.1.1.10 OS_MAX_CONDVARS `#define OS_MAX_CONDVARS 4`

The maximum number of condition variables to support.

Based on the OSAL_CONFIG_MAX_CONDVARS configuration option

Definition at line 79 of file osconfig.h.

7.1.1.11 OS_MAX_CONSOLES `#define OS_MAX_CONSOLES 1`

The maximum number of console devices to support.

Fixed value based on current OSAL implementation, not user configurable.

Definition at line 269 of file osconfig.h.

7.1.1.12 OS_MAX_COUNT_SEMAPHORES `#define OS_MAX_COUNT_SEMAPHORES 20`

The maximum number of counting semaphores to support.

Based on the OSAL_CONFIG_MAX_COUNT_SEMAPHORES configuration option

Definition at line 58 of file osconfig.h.

7.1.1.13 OS_MAX_FILE_NAME `#define OS_MAX_FILE_NAME 20`

The maximum length of OSAL file names.

This limit applies specifically to the file name portion, not the directory portion, of a path name.

Based on the OSAL_CONFIG_MAX_FILE_NAME configuration option

Note

This value must include a terminating NUL character

Definition at line 142 of file osconfig.h.

7.1.1.14 OS_MAX_FILE_SYSTEMS `#define OS_MAX_FILE_SYSTEMS 14`

The maximum number of file systems to support.

Based on the OSAL_CONFIG_MAX_FILE_SYSTEMS configuration option

Definition at line 121 of file osconfig.h.

7.1.1.15 OS_MAX_MODULES `#define OS_MAX_MODULES 20`

The maximum number of modules to support.

Based on the OSAL_CONFIG_MAX_MODULES configuration option

Definition at line 86 of file osconfig.h.

7.1.1.16 OS_MAX_MUTEXES `#define OS_MAX_MUTEXES 20`

The maximum number of mutexes to support.

Based on the OSAL_CONFIG_MAX_MUTEXES configuration option

Definition at line 72 of file osconfig.h.

7.1.1.17 OS_MAX_NUM_OPEN_DIRS `#define OS_MAX_NUM_OPEN_DIRS 4`

The maximum number of concurrently open directories to support.

Based on the OSAL_CONFIG_MAX_NUM_OPEN_DIRS configuration option

Definition at line 114 of file osconfig.h.

7.1.1.18 OS_MAX_NUM_OPEN_FILES `#define OS_MAX_NUM_OPEN_FILES 50`

The maximum number of concurrently open files to support.

Based on the OSAL_CONFIG_MAX_NUM_OPEN_FILES configuration option

Definition at line 107 of file osconfig.h.

7.1.1.19 OS_MAX_PATH_LEN `#define OS_MAX_PATH_LEN 64`

The maximum length of OSAL path names.

This limit applies to the overall length of a path name, including the file name and directory portions.

Based on the OSAL_CONFIG_MAX_PATH_LEN configuration option

Note

This value must include a terminating NUL character

Definition at line 154 of file osconfig.h.

7.1.1.20 OS_MAX_QUEUES `#define OS_MAX_QUEUES 64`

The maximum number of queues to support.

Based on the OSAL_CONFIG_MAX_QUEUES configuration option

Definition at line 51 of file osconfig.h.

7.1.1.21 OS_MAX_SYM_LEN `#define OS_MAX_SYM_LEN 64`

The maximum length of symbols.

Based on the OSAL_CONFIG_MAX_SYM_LEN configuration option

Note

This value must include a terminating NUL character

Definition at line 130 of file osconfig.h.

7.1.1.22 OS_MAX_TASKS `#define OS_MAX_TASKS 64`

The maximum number of to support.

Based on the OSAL_CONFIG_MAX_TASKS configuration option

Definition at line 44 of file osconfig.h.

7.1.1.23 OS_MAX_TIMEBASES `#define OS_MAX_TIMEBASES 5`

The maximum number of timebases to support.

Based on the OSAL_CONFIG_MAX_TIMEBASES configuration option

Definition at line 93 of file osconfig.h.

7.1.1.24 OS_MAX_TIMERS `#define OS_MAX_TIMERS 10`

The maximum number of timer callbacks to support.

Based on the OSAL_CONFIG_MAX_TIMERS configuration option

Definition at line 100 of file osconfig.h.

7.1.1.25 OS_MODULE_FILE_EXTENSION `#define OS_MODULE_FILE_EXTENSION ".so"`

The system-specific file extension used on loadable module files.

Fixed value based on system selection, not user configurable.

Definition at line 276 of file osconfig.h.

7.1.1.26 OS_PRINTF_CONSOLE_NAME `#define OS_PRINTF_CONSOLE_NAME ""`

The name of the primary console device.

This is the device to which OS_printf() output is written. The output may be configured to tag each line with this prefix for identification.

Based on the OSAL_CONFIG_PRINTF_CONSOLE_NAME configuration option

Definition at line 245 of file osconfig.h.

7.1.1.27 OS_QUEUE_MAX_DEPTH `#define OS_QUEUE_MAX_DEPTH 50`

The maximum depth of OSAL queues.

Based on the OSAL_CONFIG_QUEUE_MAX_DEPTH configuration option

Definition at line 225 of file osconfig.h.

7.1.1.28 OS_SHELL_CMD_INPUT_FILE_NAME `#define OS_SHELL_CMD_INPUT_FILE_NAME ""`

The name of the temporary file used to store shell commands.

This configuration is only applicable if shell support is enabled, and only necessary/relevant on some OS implementations.

Based on the OSAL_CONFIG_SHELL_CMD_INPUT_FILE_NAME configuration option

Definition at line 235 of file osconfig.h.

7.1.1.29 OS_SOCKADDR_MAX_LEN `#define OS_SOCKADDR_MAX_LEN 28`

The maximum size of the socket address structure.

This is part of the Socket API, and should be set large enough to hold the largest address type in use on the target system.

Based on the OSAL_CONFIG_SOCKADDR_MAX_LEN configuration option

Definition at line 173 of file osconfig.h.

7.1.1.30 OS_UTILITYTASK_PRIORITY `#define OS_UTILITYTASK_PRIORITY 245`

Priority level of the background utility task.

This task is responsible for writing buffered output of OS_printf to the actual console device, and any other future maintenance task.

Based on the OSAL_CONFIG_UTILITYTASK_PRIORITY configuration option

Definition at line 197 of file osconfig.h.

7.1.1.31 OS_UTILITYTASK_STACK_SIZE `#define OS_UTILITYTASK_STACK_SIZE 2048`

The stack size of the background utility task.

This task is responsible for writing buffered output of OS_printf to the actual console device, and any other future maintenance task.

Based on the OSAL_CONFIG_UTILITYTASK_STACK_SIZE configuration option

Definition at line 207 of file osconfig.h.

7.1.1.32 OSAL_CONFIG_CONSOLE_ASYNC `#define OSAL_CONFIG_CONSOLE_ASYNC`

Definition at line 27 of file osconfig.h.

7.1.1.33 OSAL_CONFIG_INCLUDE_DYNAMIC_LOADER `#define OSAL_CONFIG_INCLUDE_DYNAMIC_LOADER`

Configuration file Operating System Abstraction Layer.

The specific definitions in this file may only be modified by setting the respective OSAL configuration options in the CMake build.

Any direct modifications to the generated copy will be overwritten each time CMake executes.

Note

This file was automatically generated by CMake from /home/runner/work/cFS/cFS/osal/default_config.cmake

Definition at line 21 of file osconfig.h.

7.1.1.34 OSAL_CONFIG_INCLUDE_NETWORK `#define OSAL_CONFIG_INCLUDE_NETWORK`

Definition at line 22 of file osconfig.h.

7.1.1.35 OSAL_CONFIG_INCLUDE_STATIC_LOADER `#define OSAL_CONFIG_INCLUDE_STATIC_LOADER`

Definition at line 23 of file osconfig.h.

7.2 osal/docs/src/osal_frontpage.dox File Reference

7.3 osal/docs/src/osal_fs.dox File Reference

7.4 osal/docs/src/osal_timer.dox File Reference

7.5 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_USED
- #define OS_PRINTF(n, m)
- #define OSAL_SIZE_C(X) ((size_t)(X))
- #define OSAL_BLOCKCOUNT_C(X) ((osal_blockcount_t)(X))
- #define OSAL_INDEX_C(X) ((osal_index_t)(X))
- #define OSAL_OBJTYPE_C(X) ((osal_objtype_t)(X))
- #define OSAL_STATUS_C(X) ((osal_status_t)(X))

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 uint32 osal_id_t
- typedef size_t osal_blockcount_t
- typedef uint32 osal_index_t
- typedef uint32 osal_objtype_t
- typedef int32 osal_status_t
- typedef void(∗ OS_ArgCallback_t) (osal_id_t object_id, void ∗arg)

  General purpose OSAL callback function.

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)

7.5.1 Detailed Description

Purpose: Unit specification for common types.

Design Notes: Assumes make file has defined processor family

7.5.2 Macro Definition Documentation

7.5.2.1 _EXTENSION_ `#define _EXTENSION_`

Definition at line 65 of file common_types.h.

7.5.2.2 CompileTimeAssert

```
#define CompileTimeAssert(
        Condition,
        Message ) typedef char Message[(Condition) ?  1 :  -1]
```

Definition at line 48 of file common_types.h.

7.5.2.3 OS_PRINTF

```
#define OS_PRINTF(
        n,
        m )
```

Definition at line 67 of file common_types.h.

7.5.2.4 OS_USED `#define OS_USED`

Definition at line 66 of file common_types.h.

7.5.2.5 OSAL_BLOCKCOUNT_C

```
#define OSAL_BLOCKCOUNT_C(
        X ) ((osal_blockcount_t)(X))
```

Definition at line 172 of file common_types.h.

7.5.2.6 OSAL_INDEX_C #define OSAL_INDEX_C(X) ((osal_index_t)(X))

Definition at line 173 of file common_types.h.

7.5.2.7 OSAL_OBJTYPE_C #define OSAL_OBJTYPE_C(X) ((osal_objtype_t)(X))

Definition at line 174 of file common_types.h.

7.5.2.8 OSAL_SIZE_C #define OSAL_SIZE_C(X) ((size_t)(X))

Definition at line 171 of file common_types.h.

7.5.2.9 OSAL_STATUS_C #define OSAL_STATUS_C(X) ((osal_status_t)(X))

Definition at line 175 of file common_types.h.

7.5.3 Typedef Documentation

7.5.3.1 cpuaddr typedef uintptr_t cpuaddr

Definition at line 88 of file common_types.h.

7.5.3.2 cpudiff typedef ptrdiff_t cpudiff

Definition at line 90 of file common_types.h.

7.5.3.3 cpusize typedef size_t cpusize

Definition at line 89 of file common_types.h.

7.5.3.4 int16 `typedef int16_t int16`

Definition at line 80 of file common_types.h.

7.5.3.5 int32 `typedef int32_t int32`

Definition at line 81 of file common_types.h.

7.5.3.6 int64 `typedef int64_t int64`

Definition at line 82 of file common_types.h.

7.5.3.7 int8 `typedef int8_t int8`

Definition at line 79 of file common_types.h.

7.5.3.8 intptr `typedef intptr_t intptr`

Definition at line 87 of file common_types.h.

7.5.3.9 OS_ArgCallback_t `typedef void(* OS_ArgCallback_t) (osal_id_t object_id, void *arg)`

General purpose OSAL callback function.

This may be used by multiple APIS

Definition at line 143 of file common_types.h.

7.5.3.10 osal_blockcount_t `typedef size_t osal_blockcount_t`

A type used to represent a number of blocks or buffers

This is used with file system and queue implementations.

Definition at line 116 of file common_types.h.

7.5.3.11 osal_id_t `typedef uint32 osal_id_t`

A type to be used for OSAL resource identifiers. This typedef is backward compatible with the IDs from older versions of OSAL

Definition at line 108 of file common_types.h.

7.5.3.12 osal_index_t `typedef uint32 osal_index_t`

A type used to represent an index into a table structure

This is used when referring directly to a table index as opposed to an object ID. It is primarily intended for internal use, but is also output from public APIs such as OS_ObjectIdToArrayIndex().

Definition at line 126 of file common_types.h.

7.5.3.13 osal_objtype_t `typedef uint32 osal_objtype_t`

A type used to represent the runtime type or category of an OSAL object

Definition at line 131 of file common_types.h.

7.5.3.14 osal_status_t `typedef int32 osal_status_t`

The preferred type to represent OSAL status codes defined in osapi-error.h

Definition at line 136 of file common_types.h.

7.5.3.15 uint16 `typedef uint16_t uint16`

Definition at line 84 of file common_types.h.

7.5.3.16 uint32 `typedef uint32_t uint32`

Definition at line 85 of file common_types.h.

7.5.3.17 uint64 typedef uint64_t uint64

Definition at line 86 of file common_types.h.

7.5.3.18 uint8 typedef uint8_t uint8

Definition at line 83 of file common_types.h.

7.5.4 Function Documentation

7.5.4.1 CompileTimeAssert() [1/9] CompileTimeAssert (

```
sizeof(cpuaddr) >=sizeof(void *) ,
TypePtrWrongSize  )
```

7.5.4.2 CompileTimeAssert() [2/9] CompileTimeAssert (

```
sizeof(int16)  = =2,
Typeint16WrongSize  )
```

7.5.4.3 CompileTimeAssert() [3/9] CompileTimeAssert (

```
sizeof(int32)  = =4,
Typeint32WrongSize  )
```

7.5.4.4 CompileTimeAssert() [4/9] CompileTimeAssert (

```
sizeof(int64)  = =8,
Typeint64WrongSize  )
```

7.5.4.5 CompileTimeAssert() [5/9] CompileTimeAssert (

```
sizeof(int8)  = =1,
Typeint8WrongSize  )
```

7.5.4.6 CompileTimeAssert() [6/9] CompileTimeAssert (

```
sizeof(uint16)  = =2,
TypeUint16WrongSize  )
```

7.5.4.7 CompileTimeAssert() [7/9] CompileTimeAssert (

```
sizeof(uint32)  = =4,
TypeUint32WrongSize  )
```

7.5.4.8 CompileTimeAssert() [8/9] CompileTimeAssert (

```
sizeof(uint64)  = =8,
TypeUint64WrongSize  )
```

7.5.4.9 CompileTimeAssert() [9/9] CompileTimeAssert (

```
sizeof(uint8)  = =1,
TypeUint8WrongSize  )
```

7.6 osal/src/os/inc/osapi-binsem.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
```

Data Structures

- struct OS_bin_sem_prop_t

  OSAL binary semaphore properties.

Macros

- #define OS_SEM_FULL 1

  Semaphore full state.
- #define OS_SEM_EMPTY 0

  Semaphore empty state.

Functions

- int32 OS_BinSemCreate (osal_id_t *sem_id, const char *sem_name, uint32 sem_initial_value, uint32 options)
  Creates a binary semaphore.
- int32 OS_BinSemFlush (osal_id_t sem_id)
  Unblock all tasks pending on the specified semaphore.
- int32 OS_BinSemGive (osal_id_t sem_id)
  Increment the semaphore value.
- int32 OS_BinSemTake (osal_id_t sem_id)
  Decrement the semaphore value.
- int32 OS_BinSemTimedWait (osal_id_t sem_id, uint32 msecs)
  Decrement the semaphore value with a timeout.
- int32 OS_BinSemDelete (osal_id_t sem_id)
  Deletes the specified Binary Semaphore.
- int32 OS_BinSemGetIdByName (osal_id_t *sem_id, const char *sem_name)
  Find an existing semaphore ID by name.
- int32 OS_BinSemGetInfo (osal_id_t sem_id, OS_bin_sem_prop_t *bin_prop)
  Fill a property object buffer with details regarding the resource.

7.6.1 Detailed Description

Declarations and prototypes for binary semaphores

7.7 osal/src/os/inc/osapi-bsp.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
```

Functions

- void OS_BSP_SetResourceTypeConfig (uint32 ResourceType, uint32 ConfigOptionValue)
- uint32 OS_BSP_GetResourceTypeConfig (uint32 ResourceType)
- uint32 OS_BSP_GetArgC (void)
- char *const * OS_BSP_GetArgV (void)
- void OS_BSP_SetExitCode (int32 code)

7.7.1 Detailed Description

Declarations and prototypes for OSAL BSP

7.8 osal/src/os/inc/osapi-clock.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
```

Data Structures

- struct OS_time_t

  OSAL time interval structure.

Macros

- #define OS_TIME_MAX ((OS_time_t) {INT64_MAX})

  The maximum value for OS_time_t.
- #define OS_TIME_ZERO ((OS_time_t) {0})

  The zero value for OS_time_t.
- #define OS_TIME_MIN ((OS_time_t) {INT64_MIN})

  The minimum value for OS_time_t.

Enumerations

- enum { OS_TIME_TICK_RESOLUTION_NS = 100 , OS_TIME_TICKS_PER_SECOND = 1000000000 / OS_TIME_TICK_RESOLUTION_NS , OS_TIME_TICKS_PER_MSEC = 1000000 / OS_TIME_TICK_↩ RESOLUTION_NS , OS_TIME_TICKS_PER_USEC = 1000 / OS_TIME_TICK_RESOLUTION_NS }

  Multipliers/divisors to convert ticks into standardized units.

Functions

- int32 OS_GetLocalTime (OS_time_t *time_struct)

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

  Set the local time.
- OS_time_t OS_TimeFromRelativeMilliseconds (int32 relative_msec)

  Gets an absolute time value relative to the current time.
- int32 OS_TimeToRelativeMilliseconds (OS_time_t time)

  Gets a relative time value from an absolute time.
- static int64 OS_TimeGetTotalSeconds (OS_time_t tm)

  Get interval from an OS_time_t object normalized to whole number of seconds.
- static OS_time_t OS_TimeFromTotalSeconds (int64 tm)

  Get an OS_time_t interval object from an integer number of seconds.
- static int64 OS_TimeGetTotalMilliseconds (OS_time_t tm)

  Get interval from an OS_time_t object normalized to millisecond units.
- static OS_time_t OS_TimeFromTotalMilliseconds (int64 tm)

  Get an OS_time_t interval object from a integer number of milliseconds.

- static int64 OS_TimeGetTotalMicroseconds (OS_time_t tm)

  Get interval from an OS_time_t object normalized to microsecond units.
- static OS_time_t OS_TimeFromTotalMicroseconds (int64 tm)

  Get an OS_time_t interval object from a integer number of microseconds.
- static int64 OS_TimeGetTotalNanoseconds (OS_time_t tm)

  Get interval from an OS_time_t object normalized to nanosecond units.
- static OS_time_t OS_TimeFromTotalNanoseconds (int64 tm)

  Get an OS_time_t interval object from a integer number of nanoseconds.
- static int64 OS_TimeGetFractionalPart (OS_time_t tm)

  Get subseconds portion (fractional part only) from an OS_time_t object.
- static uint32 OS_TimeGetSubsecondsPart (OS_time_t tm)

  Get 32-bit normalized subseconds (fractional part only) from an OS_time_t object.
- static uint32 OS_TimeGetMillisecondsPart (OS_time_t tm)

  Get milliseconds portion (fractional part only) from an OS_time_t object.
- static uint32 OS_TimeGetMicrosecondsPart (OS_time_t tm)

  Get microseconds portion (fractional part only) from an OS_time_t object.
- static uint32 OS_TimeGetNanosecondsPart (OS_time_t tm)

  Get nanoseconds portion (fractional part only) from an OS_time_t object.
- static OS_time_t OS_TimeAssembleFromNanoseconds (int64 seconds, uint32 nanoseconds)

  Assemble/Convert a number of seconds + nanoseconds into an OS_time_t interval.
- static OS_time_t OS_TimeAssembleFromMicroseconds (int64 seconds, uint32 microseconds)

  Assemble/Convert a number of seconds + microseconds into an OS_time_t interval.
- static OS_time_t OS_TimeAssembleFromMilliseconds (int64 seconds, uint32 milliseconds)

  Assemble/Convert a number of seconds + milliseconds into an OS_time_t interval.
- static OS_time_t OS_TimeAssembleFromSubseconds (int64 seconds, uint32 subseconds)

  Assemble/Convert a number of seconds + subseconds into an OS_time_t interval.
- static OS_time_t OS_TimeAdd (OS_time_t time1, OS_time_t time2)

  Computes the sum of two time intervals.
- static OS_time_t OS_TimeSubtract (OS_time_t time1, OS_time_t time2)

  Computes the difference between two time intervals.
- static bool OS_TimeEqual (OS_time_t time1, OS_time_t time2)

  Checks if two time values are equal.
- static int8_t OS_TimeGetSign (OS_time_t time)

  Checks the sign of the time value.
- static int8_t OS_TimeCompare (OS_time_t time1, OS_time_t time2)

  Compares two time values.

7.8.1 Detailed Description

Declarations and prototypes for osapi-clock module

7.8.2 Macro Definition Documentation

7.8.2.1 OS_TIME_MAX `#define OS_TIME_MAX ((OS_time_t) {INT64_MAX})`

The maximum value for OS_time_t.

This is the largest positive (future) time that is representable in an OS_time_t value.

Definition at line 56 of file osapi-clock.h.

7.8.2.2 OS_TIME_MIN `#define OS_TIME_MIN ((OS_time_t) {INT64_MIN})`

The minimum value for OS_time_t.

This is the largest negative (past) time that is representable in an OS_time_t value.

Definition at line 71 of file osapi-clock.h.

7.8.2.3 OS_TIME_ZERO `#define OS_TIME_ZERO ((OS_time_t) {0})`

The zero value for OS_time_t.

This is a reasonable initializer/placeholder value for an OS_time_t

Definition at line 63 of file osapi-clock.h.

7.8.3 Enumeration Type Documentation

7.8.3.1 anonymous enum `anonymous enum`

Multipliers/divisors to convert ticks into standardized units.

Various fixed conversion factor constants used by the conversion routines

A 100ns tick time allows max intervals of about +/- 14000 years in a 64-bit signed integer value.

Note

Applications should not directly use these values, but rather use conversion routines below to obtain standardized units (seconds/microseconds/etc).

Enumerator

OS_TIME_TICK_RESOLUTION_NS	
OS_TIME_TICKS_PER_SECOND	
OS_TIME_TICKS_PER_MSEC	
OS_TIME_TICKS_PER_USEC	

Definition at line 84 of file osapi-clock.h.

7.9 osal/src/os/inc/osapi-common.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
```

Typedefs

- typedef int32(∗ OS_EventHandler_t) (OS_Event_t event, osal_id_t object_id, void ∗data)

  A callback routine for event handling.

Enumerations

- enum OS_Event_t {
  OS_EVENT_RESERVED = 0 , OS_EVENT_RESOURCE_ALLOCATED , OS_EVENT_RESOURCE_CREATED , OS_EVENT_RESOURCE_DELETED ,
  OS_EVENT_TASK_STARTUP , OS_EVENT_MAX }

  A set of events that can be used with BSP event callback routines.

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_API_Teardown (void)

  Teardown/de-initialization of OSAL 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.
- int32 OS_RegisterEventHandler (OS_EventHandler_t handler)

  Callback routine registration.
- size_t OS_strnlen (const char ∗s, size_t maxlen)

  get string length

7.9.1 Detailed Description

Declarations and prototypes for general OSAL functions that are not part of a subsystem

7.9.2 Typedef Documentation

7.9.2.1 OS_EventHandler_t `typedef int32(* OS_EventHandler_t) (OS_Event_t event, osal_id_t object_id, void *data)`

A callback routine for event handling.

Parameters

in	*event*	The event that occurred
in	*object_id*	The associated object_id, or 0 if not associated with an object
in,out	*data*	An abstract data/context object associated with the event, or NULL.

Returns

status Execution status, see OSAL Return Code Defines.

Definition at line 98 of file osapi-common.h.

7.9.3 Enumeration Type Documentation

7.9.3.1 OS_Event_t `enum OS_Event_t`

A set of events that can be used with BSP event callback routines.

Enumerator

OS_EVENT_RESERVED	no-op/reserved event id value
OS_EVENT_RESOURCE_ALLOCATED	resource/id has been newly allocated but not yet created. This event is invoked from WITHIN the locked region, in the context of the task which is allocating the resource. If the handler returns non-success, the error will be returned to the caller and the creation process is aborted.

Enumerator

OS_EVENT_RESOURCE_CREATED	resource/id has been fully created/finalized. Invoked outside locked region, in the context of the task which created the resource. Data object is not used, passed as NULL. Return value is ignored - this is for information purposes only.
OS_EVENT_RESOURCE_DELETED	resource/id has been deleted. Invoked outside locked region, in the context of the task which deleted the resource. Data object is not used, passed as NULL. Return value is ignored - this is for information purposes only.
OS_EVENT_TASK_STARTUP	New task is starting. Invoked outside locked region, in the context of the task which is currently starting, before the entry point is called. Data object is not used, passed as NULL. If the handler returns non-success, task startup is aborted and the entry point is not called.
OS_EVENT_MAX	placeholder for end of enum, not used

Definition at line 34 of file osapi-common.h.

7.10 osal/src/os/inc/osapi-condvar.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
#include "osapi-clock.h"
```

Data Structures

- struct OS_condvar_prop_t

  OSAL condition variable properties.

Functions

- int32 OS_CondVarCreate (osal_id_t *var_id, const char *var_name, uint32 options)

  Creates a condition variable resource.
- int32 OS_CondVarLock (osal_id_t var_id)

  Locks/Acquires the underlying mutex associated with a condition variable.
- int32 OS_CondVarUnlock (osal_id_t var_id)

  Unlocks/Releases the underlying mutex associated with a condition variable.
- int32 OS_CondVarSignal (osal_id_t var_id)

  Signals the condition variable resource referenced by var_id.
- int32 OS_CondVarBroadcast (osal_id_t var_id)

  Broadcasts the condition variable resource referenced by var_id.

- int32 OS_CondVarWait (osal_id_t var_id)

  Waits on the condition variable object referenced by var_id.
- int32 OS_CondVarTimedWait (osal_id_t var_id, const OS_time_t *abs_wakeup_time)

  Time-limited wait on the condition variable object referenced by var_id.
- int32 OS_CondVarDelete (osal_id_t var_id)

  Deletes the specified condition variable.
- int32 OS_CondVarGetIdByName (osal_id_t *var_id, const char *var_name)

  Find an existing condition variable ID by name.
- int32 OS_CondVarGetInfo (osal_id_t var_id, OS_condvar_prop_t *condvar_prop)

  Fill a property object buffer with details regarding the resource.

7.10.1 Detailed Description

Declarations and prototypes for condition variables

7.11 osal/src/os/inc/osapi-constants.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
```

Macros

- #define OS_PEND (-1)
- #define OS_CHECK (0)
- #define OS_OBJECT_ID_UNDEFINED ((osal_id_t) {0})

  Initializer for the osal_id_t type which will not match any valid value.
- #define OS_OBJECT_CREATOR_ANY OS_OBJECT_ID_UNDEFINED

  Constant that may be passed to OS_ForEachObject()/OS_ForEachObjectOfType() to match any creator (i.e. get all objects)
- #define OS_MAX_LOCAL_PATH_LEN (OS_MAX_PATH_LEN + OS_FS_PHYS_NAME_LEN)

  Maximum length of a local/native path name string.

7.11.1 Detailed Description

General constants for OSAL that are shared across subsystems

7.11.2 Macro Definition Documentation

7.11.2.1 OS_CHECK `#define OS_CHECK (0)`

Definition at line 35 of file osapi-constants.h.

7.11.2.2 OS_MAX_LOCAL_PATH_LEN `#define OS_MAX_LOCAL_PATH_LEN (OS_MAX_PATH_LEN + OS_FS_PHYS_NAME_LEN)`

Maximum length of a local/native path name string.

This is a concatenation of the OSAL virtual path with the system mount point or device name

Definition at line 54 of file osapi-constants.h.

7.11.2.3 OS_OBJECT_CREATOR_ANY `#define OS_OBJECT_CREATOR_ANY OS_OBJECT_ID_UNDEFINED`

Constant that may be passed to OS_ForEachObject()/OS_ForEachObjectOfType() to match any creator (i.e. get all objects)

Definition at line 46 of file osapi-constants.h.

7.11.2.4 OS_OBJECT_ID_UNDEFINED `#define OS_OBJECT_ID_UNDEFINED ((osal_id_t) {0})`

Initializer for the osal_id_t type which will not match any valid value.

Definition at line 40 of file osapi-constants.h.

7.11.2.5 OS_PEND `#define OS_PEND (-1)`

Definition at line 34 of file osapi-constants.h.

7.12 osal/src/os/inc/osapi-countsem.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
```

Data Structures

- struct OS_count_sem_prop_t

  OSAL counting semaphore properties.

Functions

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

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

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

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

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

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

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

  Fill a property object buffer with details regarding the resource.

7.12.1 Detailed Description

Declarations and prototypes for counting semaphores

7.13 osal/src/os/inc/osapi-dir.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
```

Data Structures

- struct os_dirent_t

  Directory entry.

Macros

- #define OS_DIRENTRY_NAME(x) ((x).FileName)

  Access filename part of the dirent structure.

Functions

- int32 OS_DirectoryOpen (osal_id_t *dir_id, const char *path)

  Opens a directory.
- int32 OS_DirectoryClose (osal_id_t dir_id)

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

  Rewinds an open directory.
- int32 OS_DirectoryRead (osal_id_t 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.

7.13.1 Detailed Description

Declarations and prototypes for directories

7.13.2 Macro Definition Documentation

7.13.2.1 OS_DIRENTRY_NAME `#define OS_DIRENTRY_NAME( x ) ((x).FileName)`

Access filename part of the dirent structure.

Definition at line 38 of file osapi-dir.h.

7.14 osal/src/os/inc/osapi-error.h File Reference

```
#include "common_types.h"
```

Macros

- #define OS_ERROR_NAME_LENGTH 35

  Error string name length.
- #define OS_STATUS_STRING_LENGTH 12

  Status converted to string length limit.
- #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 length including null terminator greater than OS_MAX_API_NAME
- #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_ERR_OPERATION_NOT_SUPPORTED (-38)

  Requested operation not support on supplied object(s)
- #define OS_ERR_INVALID_SIZE (-40)

  Invalid Size.
- #define OS_ERR_OUTPUT_TOO_LARGE (-41)

  Size of output exceeds limit
- #define OS_ERR_INVALID_ARGUMENT (-42)

  Invalid argument value (other than ID or size)
- #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.

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 char os_status_string_t[OS_STATUS_STRING_LENGTH]

  For the OS_StatusToString() function, to ensure everyone is making an array of the same length.

Functions

- static long OS_StatusToInteger (osal_status_t Status)

  Convert a status code to a native "long" type.
- int32 OS_GetErrorName (int32 error_num, os_err_name_t *err_name)

  Convert an error number to a string.
- char * OS_StatusToString (osal_status_t status, os_status_string_t *status_string)

  Convert status to a string.

7.14.1 Detailed Description

OSAL error code definitions

7.14.2 Macro Definition Documentation

7.14.2.1 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 35 of file osapi-error.h.

7.14.2.2 OS_STATUS_STRING_LENGTH `#define OS_STATUS_STRING_LENGTH 12`

Status converted to string length limit.

Used for sizing os_status_string_t intended for use in printing osal_status_t values Sized to fit LONG_MIN including NULL termination

Definition at line 55 of file osapi-error.h.

7.14.3 Typedef Documentation

7.14.3.1 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 47 of file osapi-error.h.

7.14.3.2 os_status_string_t typedef char os_status_string_t[OS_STATUS_STRING_LENGTH]

For the OS_StatusToString() function, to ensure everyone is making an array of the same length.

Definition at line 61 of file osapi-error.h.

7.15 osal/src/os/inc/osapi-file.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
#include "osapi-clock.h"
```

Data Structures

- struct OS_file_prop_t
  OSAL file properties.
- struct os_fstat_t
  File system status.

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_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) (OS_TimeGetTotalSeconds((x).FileTime))
  Access file stat time field as a whole number of seconds.

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.
- enum OS_file_flag_t { OS_FILE_FLAG_NONE = 0x00 , OS_FILE_FLAG_CREATE = 0x01 , OS_FILE_FLAG_TRUNCATE = 0x02 }

  Flags that can be used with opening of a file (bitmask)

Functions

- int32 OS_OpenCreate (osal_id_t *filedes, const char *path, int32 flags, int32 access_mode)

  Open or create a file.
- int32 OS_close (osal_id_t filedes)

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

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

  Write to a file handle.
- int32 OS_TimedReadAbs (osal_id_t filedes, void *buffer, size_t nbytes, OS_time_t abstime)

  File/Stream input read with a timeout.
- int32 OS_TimedRead (osal_id_t filedes, void *buffer, size_t nbytes, int32 timeout)

  File/Stream input read with a timeout.
- int32 OS_TimedWriteAbs (osal_id_t filedes, const void *buffer, size_t nbytes, OS_time_t abstime)

  File/Stream output write with a timeout.
- int32 OS_TimedWrite (osal_id_t filedes, const void *buffer, size_t nbytes, int32 timeout)

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

  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 (osal_id_t 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 (osal_id_t 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.

7.15.1 Detailed Description

Declarations and prototypes for file objects

7.15.2 Macro Definition Documentation

7.15.2.1 OS_FILESTAT_EXEC

```
#define OS_FILESTAT_EXEC(
        x ) ((x).FileModeBits & OS_FILESTAT_MODE_EXEC)
```

File stat is executable logical.

Definition at line 92 of file osapi-file.h.

7.15.2.2 OS_FILESTAT_ISDIR

```
#define OS_FILESTAT_ISDIR(
        x ) ((x).FileModeBits & OS_FILESTAT_MODE_DIR)
```

File stat is directory logical.

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

7.15.2.3 OS_FILESTAT_MODE

```
#define OS_FILESTAT_MODE(
        x ) ((x).FileModeBits)
```

Access file stat mode bits.

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

7.15.2.4 OS_FILESTAT_READ

```
#define OS_FILESTAT_READ(
        x ) ((x).FileModeBits & OS_FILESTAT_MODE_READ)
```

File stat is read enabled logical.

Definition at line 96 of file osapi-file.h.

7.15.2.5 OS_FILESTAT_SIZE `#define OS_FILESTAT_SIZE( x ) ((x).FileSize)`

Access file stat size field.

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

7.15.2.6 OS_FILESTAT_TIME `#define OS_FILESTAT_TIME( x ) (OS_TimeGetTotalSeconds((x).FileTime))`

Access file stat time field as a whole number of seconds.

Definition at line 100 of file osapi-file.h.

7.15.2.7 OS_FILESTAT_WRITE `#define OS_FILESTAT_WRITE( x ) ((x).FileModeBits & OS_FILESTAT_MODE_WRITE)`

File stat is write enabled logical.

Definition at line 94 of file osapi-file.h.

7.15.3 Enumeration Type Documentation

7.15.3.1 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 79 of file osapi-file.h.

7.15.3.2 OS_file_flag_t enum OS_file_flag_t

Flags that can be used with opening of a file (bitmask)

Enumerator

OS_FILE_FLAG_NONE	
OS_FILE_FLAG_CREATE	
OS_FILE_FLAG_TRUNCATE	

Definition at line 105 of file osapi-file.h.

7.16 osal/src/os/inc/osapi-filesys.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
```

Data Structures

- struct os_fsinfo_t

  OSAL file system info.
- struct OS_statvfs_t

Macros

- #define OS_CHK_ONLY 0
- #define OS_REPAIR 1

Functions

- int32 OS_FileSysAddFixedMap (osal_id_t *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, size_t blocksize, osal_blockcount_t 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, size_t blocksize, osal_blockcount_t 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_FileSysStatVolume (const char *name, OS_statvfs_t *statbuf)

  Obtains information about size and free space 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 an OSAL Virtual file system path to a host Local path.
- int32 OS_GetFsInfo (os_fsinfo_t *filesys_info)

  Returns information about the file system.

7.16.1 Detailed Description

Declarations and prototypes for file systems

7.16.2 Macro Definition Documentation

7.16.2.1 OS_CHK_ONLY `#define OS_CHK_ONLY 0`

Unused, API takes bool

Definition at line 31 of file osapi-filesys.h.

7.16.2.2 OS_REPAIR `#define OS_REPAIR 1`

Unused, API takes bool

Definition at line 32 of file osapi-filesys.h.

7.17 osal/src/os/inc/osapi-heap.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
```

Data Structures

- struct OS_heap_prop_t

  OSAL heap properties.

Functions

- int32 OS_HeapGetInfo (OS_heap_prop_t *heap_prop)

  Return current info on the heap.

7.17.1 Detailed Description

Declarations and prototypes for heap functions

7.18 osal/src/os/inc/osapi-idmap.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
```

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_OS_CONDVAR 0x0D

  Object condition variable type.
- #define OS_OBJECT_TYPE_USER 0x10

  Object user type.

Functions

- static unsigned long OS_ObjectIdToInteger (osal_id_t object_id)

  Obtain an integer value corresponding to an object ID.
- static osal_id_t OS_ObjectIdFromInteger (unsigned long value)

  Obtain an osal ID corresponding to an integer value.
- static bool OS_ObjectIdEqual (osal_id_t object_id1, osal_id_t object_id2)

  Check two OSAL object ID values for equality.
- static bool OS_ObjectIdDefined (osal_id_t object_id)

  Check if an object ID is defined.
- int32 OS_GetResourceName (osal_id_t object_id, char *buffer, size_t buffer_size)

  Obtain the name of an object given an arbitrary object ID.
- osal_objtype_t OS_IdentifyObject (osal_id_t object_id)

  Obtain the type of an object given an arbitrary object ID.
- int32 OS_ConvertToArrayIndex (osal_id_t object_id, osal_index_t *ArrayIndex)

  Converts an abstract ID into a number suitable for use as an array index.
- int32 OS_ObjectIdToArrayIndex (osal_objtype_t idtype, osal_id_t object_id, osal_index_t *ArrayIndex)

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

  call the supplied callback function for all valid object IDs
- void OS_ForEachObjectOfType (osal_objtype_t objtype, osal_id_t creator_id, OS_ArgCallback_t callback_ptr, void *callback_arg)

  call the supplied callback function for valid object IDs of a specific type

7.18.1 Detailed Description

Declarations and prototypes for object IDs

7.18.2 Macro Definition Documentation

7.18.2.1 OS_OBJECT_INDEX_MASK `#define OS_OBJECT_INDEX_MASK 0xFFFF`

Object index mask.

Definition at line 32 of file osapi-idmap.h.

7.18.2.2 OS_OBJECT_TYPE_SHIFT `#define OS_OBJECT_TYPE_SHIFT 16`

Object type shift.

Definition at line 33 of file osapi-idmap.h.

7.19 osal/src/os/inc/osapi-macros.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "osconfig.h"
#include "common_types.h"
#include "osapi-printf.h"
```

Macros

- #define BUGREPORT(...) OS_printf(__VA_ARGS__)
- #define BUGCHECK(cond, errcode)

  Basic Bug-Checking macro.
- #define ARGCHECK(cond, errcode)

  Generic argument checking macro for non-critical values.
- #define LENGTHCHECK(str, len, errcode) ARGCHECK(memchr(str, '\0', len), errcode)

  String length limit check macro.
- #define BUGCHECK_VOID(cond) BUGCHECK(cond,)

  Bug-Check macro for void functions.

7.19.1 Detailed Description

Macro definitions that are used across all OSAL subsystems

7.19.2 Macro Definition Documentation

7.19.2.1 ARGCHECK `#define ARGCHECK(`

```
        cond,
        errcode )
```

Value:

```
if (!(cond))                \
{                           \
    return errcode;         \
}
```

Generic argument checking macro for non-critical values.

This macro checks a conditional that is expected to be true, and return a value if it evaluates false.

ARGCHECK can be used to check for out of range or other invalid argument conditions which may (validly) occur at runtime and do not necessarily indicate bugs in the application.

These argument checks are NOT considered fatal errors. The application continues to run normally. This does not report the error on the console.

As such, ARGCHECK actions are always compiled in - not selectable at compile-time.

See also

BUGCHECK for checking critical values that indicate bugs

Definition at line 131 of file osapi-macros.h.

7.19.2.2 BUGCHECK `#define BUGCHECK(`

```
        cond,
        errcode )
```

Value:

```
if (!(cond))                                                                                    \
{                                                                                               \
    BUGREPORT("\n**BUG** %s():%d:check \'%s\' FAILED --> %s\n\n", __func__, __LINE__, #cond, #errcode); \
    return errcode;                                                                             \
}
```

Basic Bug-Checking macro.

This macro checks a conditional, and if it is FALSE, then it generates a report - which may in turn contain additional actions.

BUGCHECK should only be used for conditions which are critical and must always be true. If such a condition is ever false then it indicates a bug in the application which must be resolved. It may or may not be possible to continue operation if a bugcheck fails.

See also

ARGCHECK for checking non-critical values

Definition at line 105 of file osapi-macros.h.

7.19.2.3 BUGCHECK_VOID

```
#define BUGCHECK_VOID(
        cond ) BUGCHECK(cond, )
```

Bug-Check macro for void functions.

The basic BUGCHECK macro returns a value, which needs to be empty for functions that do not have a return value. In this case the second argument (errcode) is intentionally left blank.

Definition at line 155 of file osapi-macros.h.

7.19.2.4 BUGREPORT

```
#define BUGREPORT(
        ... ) OS_printf(__VA_ARGS__)
```

Definition at line 88 of file osapi-macros.h.

7.19.2.5 LENGTHCHECK

```
#define LENGTHCHECK(
        str,
        len,
        errcode ) ARGCHECK(memchr(str, '\0', len), errcode)
```

String length limit check macro.

This macro is a specialized version of ARGCHECK that confirms a string will fit into a buffer of the specified length, and return an error code if it will not.

Note

this uses ARGCHECK, thus treating a string too long as a normal runtime (i.e. non-bug) error condition with a typical error return to the caller.

Definition at line 146 of file osapi-macros.h.

7.20 osal/src/os/inc/osapi-module.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
```

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.

Macros

- #define OS_MODULE_FLAG_GLOBAL_SYMBOLS 0x00

  Requests OS_ModuleLoad() to add the symbols to the global symbol table.
- #define OS_MODULE_FLAG_LOCAL_SYMBOLS 0x01

  Requests OS_ModuleLoad() to keep the symbols local/private to this module.

Functions

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

  Find the Address of a Symbol.
- int32 OS_ModuleSymbolLookup (osal_id_t module_id, cpuaddr *symbol_address, const char *symbol_name)

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

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

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

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

  Obtain information about a module.

7.20.1 Detailed Description

Declarations and prototypes for module subsystem

7.20.2 Macro Definition Documentation

7.20.2.1 OS_MODULE_FLAG_GLOBAL_SYMBOLS `#define OS_MODULE_FLAG_GLOBAL_SYMBOLS 0x00`

Requests OS_ModuleLoad() to add the symbols to the global symbol table.

When supplied as the "flags" argument to OS_ModuleLoad(), this indicates that the symbols in the loaded module should be added to the global symbol table. This will make symbols in this library available for use when resolving symbols in future module loads.

This is the default mode of operation for OS_ModuleLoad().

Note

On some operating systems, use of this option may make it difficult to unload the module in the future, if the symbols are in use by other entities.

Definition at line 49 of file osapi-module.h.

7.20.2.2 OS_MODULE_FLAG_LOCAL_SYMBOLS `#define OS_MODULE_FLAG_LOCAL_SYMBOLS 0x01`

Requests OS_ModuleLoad() to keep the symbols local/private to this module.

When supplied as the "flags" argument to OS_ModuleLoad(), this indicates that the symbols in the loaded module should NOT be added to the global symbol table. This means the symbols in the loaded library will not be available for use by other modules.

Use this option is recommended for cases where no other entities will need to reference symbols within this module. This helps ensure that the module can be more safely unloaded in the future, by preventing other modules from binding to it. It also helps reduce the likelihood of symbol name conflicts among modules.

Note

To look up symbols within a module loaded with this flag, use OS_SymbolLookupInModule() instead of OS_SymbolLookup(). Also note that references obtained using this method are not tracked by the OS; the application must ensure that all references obtained in this manner have been cleaned up/released before unloading the module.

Definition at line 71 of file osapi-module.h.

7.21 osal/src/os/inc/osapi-mutex.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
```

Data Structures

- struct OS_mut_sem_prop_t

  OSAL mutex properties.

Functions

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

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

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

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

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

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

  Fill a property object buffer with details regarding the resource.

7.21.1 Detailed Description

Declarations and prototypes for mutexes

7.22 osal/src/os/inc/osapi-network.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
```

Functions

- int32 OS_NetworkGetID (void)

  Gets the network ID of the local machine.
- int32 OS_NetworkGetHostName (char *host_name, size_t name_len)

  Gets the local machine network host name.

7.22.1 Detailed Description

Declarations and prototypes for network subsystem

7.23 osal/src/os/inc/osapi-printf.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
```

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.

7.23.1 Detailed Description

Declarations and prototypes for printf/console output

7.24 osal/src/os/inc/osapi-queue.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
```

Data Structures

- struct OS_queue_prop_t

  OSAL queue properties.

Functions

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

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

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

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

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

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

  Fill a property object buffer with details regarding the resource.

7.24.1 Detailed Description

Declarations and prototypes for queue subsystem

7.25 osal/src/os/inc/osapi-select.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
#include "osapi-clock.h"
```

Data Structures

- struct OS_FdSet

  An abstract structure capable of holding several OSAL IDs.

Enumerations

- enum OS_StreamState_t {
  OS_STREAM_STATE_BOUND = 0x01 , OS_STREAM_STATE_CONNECTED = 0x02 , OS_STREAM_STATE_READABLE = 0x04 , OS_STREAM_STATE_WRITABLE = 0x08 ,
  OS_STREAM_STATE_LISTENING = 0x10 }

  For the OS_SelectSingle() function's in/out StateFlags parameter, the state(s) of the stream and the result of the select is a combination of one or more of these states.

Functions

- int32 OS_SelectMultipleAbs (OS_FdSet *ReadSet, OS_FdSet *WriteSet, OS_time_t abs_timeout)

  Wait for events across multiple file handles.
- int32 OS_SelectMultiple (OS_FdSet *ReadSet, OS_FdSet *WriteSet, int32 msecs)

  Wait for events across multiple file handles.
- int32 OS_SelectSingleAbs (osal_id_t objid, uint32 *StateFlags, OS_time_t abs_timeout)

  Wait for events on a single file handle.
- int32 OS_SelectSingle (osal_id_t 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, osal_id_t objid)

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

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

  Check if an FdSet structure contains a given ID.

7.25.1 Detailed Description

Declarations and prototypes for select abstraction

7.25.2 Enumeration Type Documentation

7.25.2.1 OS_StreamState_t `enum OS_StreamState_t`

For the OS_SelectSingle() function's in/out StateFlags parameter, the state(s) of the stream and the result of the select is a combination of one or more of these states.

See also

OS_SelectSingle()

Enumerator

OS_STREAM_STATE_BOUND	whether the stream is bound
OS_STREAM_STATE_CONNECTED	whether the stream is connected
OS_STREAM_STATE_READABLE	whether the stream is readable
OS_STREAM_STATE_WRITABLE	whether the stream is writable
OS_STREAM_STATE_LISTENING	whether the stream is listening

Definition at line 56 of file osapi-select.h.

7.26 osal/src/os/inc/osapi-shell.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
```

Functions

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

  Executes the command and sends output to a file.

7.26.1 Detailed Description

Declarations and prototypes for shell abstraction

7.27 osal/src/os/inc/osapi-sockets.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
#include "osapi-clock.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_INET6 , OS_SocketDomain_MAX }

  Socket domain.
- enum OS_SocketType_t { OS_SocketType_INVALID , OS_SocketType_DATAGRAM , OS_SocketType_STREAM , OS_SocketType_MAX }

  Socket type.
- enum OS_SocketShutdownMode_t { OS_SocketShutdownMode_NONE = 0 , OS_SocketShutdownMode_SHUT_READ = 1 , OS_SocketShutdownMode_SHUT_WRITE = 2 , OS_SocketShutdownMode_SHUT_READWRITE = 3 }

  Shutdown Mode.

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, size_t 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 (osal_id_t *sock_id, OS_SocketDomain_t Domain, OS_SocketType_t Type)

  Opens a socket.
- int32 OS_SocketBind (osal_id_t sock_id, const OS_SockAddr_t *Addr)

  Binds a socket to a given local address and enter listening (server) mode.
- int32 OS_SocketListen (osal_id_t sock_id)

  Places the specified socket into a listening state.
- int32 OS_SocketBindAddress (osal_id_t sock_id, const OS_SockAddr_t *Addr)

  Binds a socket to a given local address.
- int32 OS_SocketConnectAbs (osal_id_t sock_id, const OS_SockAddr_t *Addr, OS_time_t abs_timeout)

  Connects a socket to a given remote address.
- int32 OS_SocketConnect (osal_id_t sock_id, const OS_SockAddr_t *Addr, int32 timeout)

  Connects a socket to a given remote address.
- int32 OS_SocketShutdown (osal_id_t sock_id, OS_SocketShutdownMode_t Mode)

  Implement graceful shutdown of a stream socket.
- int32 OS_SocketAcceptAbs (osal_id_t sock_id, osal_id_t *connsock_id, OS_SockAddr_t *Addr, OS_time_t abs↩_timeout)

  Waits for and accept the next incoming connection on the given socket.
- int32 OS_SocketAccept (osal_id_t sock_id, osal_id_t *connsock_id, OS_SockAddr_t *Addr, int32 timeout)

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

- int32 OS_SocketRecvFromAbs (osal_id_t sock_id, void *buffer, size_t buflen, OS_SockAddr_t *RemoteAddr, OS_time_t abs_timeout)

  Reads data from a message-oriented (datagram) socket.
- int32 OS_SocketRecvFrom (osal_id_t sock_id, void *buffer, size_t buflen, OS_SockAddr_t *RemoteAddr, int32 timeout)

  Reads data from a message-oriented (datagram) socket.
- int32 OS_SocketSendTo (osal_id_t sock_id, const void *buffer, size_t buflen, const OS_SockAddr_t *RemoteAddr)

  Sends data to a message-oriented (datagram) socket.
- int32 OS_SocketGetIdByName (osal_id_t *sock_id, const char *sock_name)

  Gets an OSAL ID from a given name.
- int32 OS_SocketGetInfo (osal_id_t sock_id, OS_socket_prop_t *sock_prop)

  Gets information about an OSAL Socket ID.

7.27.1 Detailed Description

Declarations and prototypes for sockets abstraction

7.27.2 Macro Definition Documentation

7.27.2.1 OS_SOCKADDR_MAX_LEN `#define OS_SOCKADDR_MAX_LEN 28`

Definition at line 46 of file osapi-sockets.h.

7.27.3 Enumeration Type Documentation

7.27.3.1 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 61 of file osapi-sockets.h.

7.27.3.2 OS_SocketShutdownMode_t enum OS_SocketShutdownMode_t

Shutdown Mode.

Enumerator

OS_SocketShutdownMode_NONE	Reserved value, no effect.
OS_SocketShutdownMode_SHUT_READ	Disable future reading.
OS_SocketShutdownMode_SHUT_WRITE	Disable future writing.
OS_SocketShutdownMode_SHUT_READWRITE	Disable future reading or writing.

Definition at line 80 of file osapi-sockets.h.

7.27.3.3 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 70 of file osapi-sockets.h.

7.28 osal/src/os/inc/osapi-task.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
```

Data Structures

- struct OS_task_prop_t

  OSAL task properties.

Macros

- #define OS_MAX_TASK_PRIORITY 255

  Upper limit for OSAL task priorities.
- #define OS_FP_ENABLED 1

  Floating point enabled state for a task.
- #define OSAL_PRIORITY_C(X) ((osal_priority_t) {X})
- #define OSAL_STACKPTR_C(X) ((osal_stackptr_t) {X})
- #define OSAL_TASK_STACK_ALLOCATE OSAL_STACKPTR_C(NULL)

Typedefs

- typedef uint8_t osal_priority_t

  Type to be used for OSAL task priorities.
- typedef void * osal_stackptr_t

  Type to be used for OSAL stack pointer.
- typedef void osal_task

  For task entry point.

Functions

- typedef osal_task ((*osal_task_entry)(void))

  For task entry point.
- int32 OS_TaskCreate (osal_id_t *task_id, const char *task_name, osal_task_entry function_pointer, osal_stackptr_t stack_pointer, size_t stack_size, osal_priority_t priority, uint32 flags)

  Creates a task and starts running it.
- int32 OS_TaskDelete (osal_id_t 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 (osal_id_t task_id, osal_priority_t new_priority)

  Sets the given task to a new priority.
- osal_id_t OS_TaskGetId (void)

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

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

  Fill a property object buffer with details regarding the resource.
- int32 OS_TaskFindIdBySystemData (osal_id_t *task_id, const void *sysdata, size_t sysdata_size)

  Reverse-lookup the OSAL task ID from an operating system ID.

7.28.1 Detailed Description

Declarations and prototypes for task abstraction

7.28.2 Macro Definition Documentation

7.28.2.1 OS_FP_ENABLED `#define OS_FP_ENABLED 1`

Floating point enabled state for a task.

Definition at line 35 of file osapi-task.h.

7.28.2.2 OS_MAX_TASK_PRIORITY `#define OS_MAX_TASK_PRIORITY 255`

Upper limit for OSAL task priorities.

Definition at line 32 of file osapi-task.h.

7.28.2.3 OSAL_PRIORITY_C `#define OSAL_PRIORITY_C( X ) ((osal_priority_t) {X})`

Definition at line 46 of file osapi-task.h.

7.28.2.4 OSAL_STACKPTR_C `#define OSAL_STACKPTR_C( X ) ((osal_stackptr_t) {X})`

Definition at line 53 of file osapi-task.h.

7.28.2.5 OSAL_TASK_STACK_ALLOCATE `#define OSAL_TASK_STACK_ALLOCATE OSAL_STACKPTR_C(NULL)`

Definition at line 54 of file osapi-task.h.

7.28.3 Typedef Documentation

7.28.3.1 osal_priority_t typedef uint8_t osal_priority_t

Type to be used for OSAL task priorities.

OSAL priorities are in reverse order, and range from 0 (highest; will preempt all other tasks) to 255 (lowest; will not preempt any other task).

Definition at line 44 of file osapi-task.h.

7.28.3.2 osal_stackptr_t typedef void* osal_stackptr_t

Type to be used for OSAL stack pointer.

Definition at line 51 of file osapi-task.h.

7.28.3.3 osal_task typedef void osal_task

For task entry point.

Definition at line 68 of file osapi-task.h.

7.28.4 Function Documentation

7.28.4.1 osal_task() typedef osal_task ((*)(void) *osal_task_entry*)

For task entry point.

7.29 osal/src/os/inc/osapi-timebase.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
```

Data Structures

- struct OS_timebase_prop_t

  Time base properties.

Typedefs

- typedef uint32(* OS_TimerSync_t) (osal_id_t timer_id)

  Timer sync.

Functions

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

  Create an abstract Time Base resource.
- int32 OS_TimeBaseSet (osal_id_t timebase_id, uint32 start_time, uint32 interval_time)

  Sets the tick period for simulated time base objects.
- int32 OS_TimeBaseDelete (osal_id_t timebase_id)

  Deletes a time base object.
- int32 OS_TimeBaseGetIdByName (osal_id_t *timebase_id, const char *timebase_name)

  Find the ID of an existing time base resource.
- int32 OS_TimeBaseGetInfo (osal_id_t timebase_id, OS_timebase_prop_t *timebase_prop)

  Obtain information about a timebase resource.
- int32 OS_TimeBaseGetFreeRun (osal_id_t timebase_id, uint32 *freerun_val)

  Read the value of the timebase free run counter.

7.29.1 Detailed Description

Declarations and prototypes for timebase abstraction

7.29.2 Typedef Documentation

7.29.2.1 OS_TimerSync_t `typedef uint32(* OS_TimerSync_t) (osal_id_t timer_id)`

Timer sync.

Definition at line 34 of file osapi-timebase.h.

7.30 osal/src/os/inc/osapi-timer.h File Reference

```
#include "osconfig.h"
#include "common_types.h"
```

Data Structures

- struct OS_timer_prop_t

  Timer properties.

Typedefs

- typedef void(∗ OS_TimerCallback_t) (osal_id_t timer_id)

  Timer callback.

Functions

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

  Create a timer object.
- int32 OS_TimerAdd (osal_id_t ∗timer_id, const char ∗timer_name, osal_id_t timebase_id, OS_ArgCallback_t callback_ptr, void ∗callback_arg)

  Add a timer object based on an existing TimeBase resource.
- int32 OS_TimerSet (osal_id_t timer_id, uint32 start_time, uint32 interval_time)

  Configures a periodic or one shot timer.
- int32 OS_TimerDelete (osal_id_t timer_id)

  Deletes a timer resource.
- int32 OS_TimerGetIdByName (osal_id_t ∗timer_id, const char ∗timer_name)

  Locate an existing timer resource by name.
- int32 OS_TimerGetInfo (osal_id_t timer_id, OS_timer_prop_t ∗timer_prop)

  Gets information about an existing timer.

7.30.1 Detailed Description

Declarations and prototypes for timer abstraction (app callbacks)

7.30.2 Typedef Documentation

7.30.2.1 OS_TimerCallback_t `typedef void(* OS_TimerCallback_t) (osal_id_t timer_id)`

Timer callback.

Definition at line 34 of file osapi-timer.h.

7.31 osal/src/os/inc/osapi-version.h File Reference

```
#include "common_types.h"
```

Macros

- #define OS_BUILD_NUMBER 123
- #define OS_BUILD_BASELINE "equuleus-rc1"
- #define OS_BUILD_DEV_CYCLE "equuleus-rc2"

  Development: Release name for current development cycle.
- #define OS_BUILD_CODENAME "Equuleus"

  : Development: Code name for the current build
- #define OS_MAJOR_VERSION 5

  Major version number.
- #define OS_MINOR_VERSION 0

  Minor version number.
- #define OS_REVISION 0

  Revision version number. Value of 99 indicates a development version.
- #define OS_LAST_OFFICIAL "v5.0.0"

  Last official release.
- #define OS_MISSION_REV 0xFF

  Mission revision.
- #define OS_STR_HELPER(x) #x

  Helper function to concatenate strings from integer.
- #define OS_STR(x) OS_STR_HELPER(x)

  Helper function to concatenate strings from integer.
- #define OS_VERSION OS_BUILD_BASELINE "+dev" OS_STR(OS_BUILD_NUMBER)

  Development Build Version Number.
- #define OSAL_API_VERSION ((OS_MAJOR_VERSION * 10000) + (OS_MINOR_VERSION * 100) + OS_REVISION)

  Combines the revision components into a single value.
- #define OS_CFG_MAX_VERSION_STR_LEN 256

  Max Version String length.

Functions

- const char * OS_GetVersionString (void)
- const char * OS_GetVersionCodeName (void)
- void OS_GetVersionNumber (uint8 VersionNumbers[4])

  Obtain the OSAL numeric version number.
- uint32 OS_GetBuildNumber (void)

  Obtain the OSAL library numeric build number.

7.31.1 Detailed Description

Provide version identifiers for Operating System Abstraction Layer

Note

OSAL follows the same version semantics as cFS, which in turn is based on the Semantic Versioning 2.0 Specification. For more information, see the documentation provided with cFE.

7.31.2 Macro Definition Documentation

7.31.2.1 OS_BUILD_BASELINE `#define OS_BUILD_BASELINE "equuleus-rc1"`

Definition at line 38 of file osapi-version.h.

7.31.2.2 OS_BUILD_CODENAME `#define OS_BUILD_CODENAME "Equuleus"`

: Development: Code name for the current build

Definition at line 40 of file osapi-version.h.

7.31.2.3 OS_BUILD_DEV_CYCLE `#define OS_BUILD_DEV_CYCLE "equuleus-rc2"`

Development: Release name for current development cycle.

Definition at line 39 of file osapi-version.h.

7.31.2.4 OS_BUILD_NUMBER `#define OS_BUILD_NUMBER 123`

Definition at line 37 of file osapi-version.h.

7.31.2.5 OS_CFG_MAX_VERSION_STR_LEN `#define OS_CFG_MAX_VERSION_STR_LEN 256`

Max Version String length.

Maximum length that an OSAL version string can be.

Definition at line 154 of file osapi-version.h.

7.31.2.6 OS_LAST_OFFICIAL `#define OS_LAST_OFFICIAL "v5.0.0"`

Last official release.

Definition at line 52 of file osapi-version.h.

7.31.2.7 OS_MAJOR_VERSION `#define OS_MAJOR_VERSION 5`

Major version number.

Definition at line 45 of file osapi-version.h.

7.31.2.8 OS_MINOR_VERSION `#define OS_MINOR_VERSION 0`

Minor version number.

Definition at line 46 of file osapi-version.h.

7.31.2.9 OS_MISSION_REV `#define OS_MISSION_REV 0xFF`

Mission revision.

Reserved for mission use to denote patches/customizations as needed. Values 1-254 are reserved for mission use to denote patches/customizations as needed. NOTE: Reserving 0 and 0xFF for cFS open-source development use (pending resolution of nasa/cFS#440)

Definition at line 61 of file osapi-version.h.

7.31.2.10 OS_REVISION `#define OS_REVISION 0`

Revision version number. Value of 99 indicates a development version.

Definition at line 47 of file osapi-version.h.

7.31.2.11 OS_STR `#define OS_STR( x ) OS_STR_HELPER(x)`

Helper function to concatenate strings from integer.

Definition at line 67 of file osapi-version.h.

7.31.2.12 OS_STR_HELPER `#define OS_STR_HELPER( x ) #x`

Helper function to concatenate strings from integer.

Definition at line 66 of file osapi-version.h.

7.31.2.13 OS_VERSION `#define OS_VERSION OS_BUILD_BASELINE "+dev" OS_STR(OS_BUILD_NUMBER)`

Development Build Version Number.

Baseline git tag + Number of commits since baseline.

Definition at line 72 of file osapi-version.h.

7.31.2.14 OSAL_API_VERSION `#define OSAL_API_VERSION ((OS_MAJOR_VERSION * 10000) + (OS_MINOR_VERSION * 100) + OS_REVISION)`

Combines the revision components into a single value.

Applications can check against this number
e.g. "#if OSAL_API_VERSION >= 40100" would check if some feature added in OSAL 4.1 is present.

Definition at line 79 of file osapi-version.h.

7.31.3 Function Documentation

7.31.3.1 OS_GetBuildNumber() `uint32 OS_GetBuildNumber ( void )`

Obtain the OSAL library numeric build number.

The build number is a monotonically increasing number that (coarsely) reflects the number of commits/changes that have been merged since the epoch release. During development cycles this number should increase after each subsequent merge/modification.

Like other version information, this is a fixed number assigned at compile time.

Returns

The OSAL library build number

7.31.3.2 OS_GetVersionCodeName() `const char* OS_GetVersionCodeName ( void )`

Gets the OSAL version code name

All NASA CFE/CFS components (including CFE framework, OSAL and PSP) that work together will share the same code name.

Returns

OSAL code name. This is a fixed value string and is never NULL.

7.31.3.3 OS_GetVersionNumber() `void OS_GetVersionNumber ( uint8 VersionNumbers[4] )`

Obtain the OSAL numeric version number.

This retrieves the numeric OSAL version identifier as an array of 4 uint8 values.

The array of numeric values is in order of precedence: [0] = Major Number [1] = Minor Number [2] = Revision Number [3] = Mission Revision

The "Mission Revision" (last output) also indicates whether this is an official release, a patched release, or a development version. 0 indicates an official release 1-254 local patch level (reserved for mission use) 255 indicates a development build

Parameters

out	*VersionNumbers*	A fixed-size array to be filled with the version numbers

7.31.3.4 OS_GetVersionString() `const char* OS_GetVersionString ( void )`

Gets the OSAL version/baseline ID as a string

This returns the content of the OS_VERSION macro defined above, and is specifically just the baseline and development build ID (if applicable), without any extra info.

Returns

Basic version identifier. This is a fixed value string and is never NULL.

7.32 osal/src/os/inc/osapi.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include "common_types.h"
#include "osapi-version.h"
#include "osconfig.h"
#include "osapi-binsem.h"
#include "osapi-clock.h"
#include "osapi-common.h"
#include "osapi-condvar.h"
#include "osapi-constants.h"
#include "osapi-countsem.h"
#include "osapi-dir.h"
#include "osapi-error.h"
#include "osapi-file.h"
#include "osapi-filesys.h"
#include "osapi-heap.h"
#include "osapi-macros.h"
#include "osapi-idmap.h"
#include "osapi-module.h"
#include "osapi-mutex.h"
#include "osapi-network.h"
#include "osapi-printf.h"
#include "osapi-queue.h"
#include "osapi-select.h"
#include "osapi-shell.h"
#include "osapi-sockets.h"
#include "osapi-task.h"
#include "osapi-timebase.h"
#include "osapi-timer.h"
#include "osapi-bsp.h"
```

7.32.1 Detailed Description

Purpose: Contains functions prototype definitions and variables declarations for the OS Abstraction Layer, Core OS module

Index