OpenClovis Software Development Kit (SDK) Service Description and API Reference for Operating System Abstraction Layer (OSAL) Service

For OpenClovis SDK Release2.3 V0.3

Document Revision Date: December 19, 2006

Contents

Chapter 1

Functional Overview

The OpenClovis Operating System Abstraction Layer (OSAL) provides a standard interface to commonly used operating system functions. OSAL supports target operating systems like most variations of Carrier Grade Linux. It is easily adaptable to any proprietary target operating system.

All OpenClovis ASP components are developed based on OSAL that provide an OS agnostic function to all system calls, such as memory management functions and thread management functions. Internally, OSAL maps such functions to the respective equivalent system calls provided by the underlying OS. This allows OpenClovis ASP as well as all OpenClovis ASP based applications to be ported to new operating systems with no modifications.

OpenClovis ASP is delivered with a Posix-compliant adaptation module that allows it to operate to any Posix-compliant system, such as Linux and most Unix systems.

Software components use the signal handlers provided by OSAL to handle critical Unix signals. This can automatically inform OpenClovis ASP Component Manager (CPM) about the signal and generate a fault or trigger necessary recovery actions. OSAL is currently designed as a standalone library with no dependencies on any other OpenClovis ASP components.

1.1 Interaction with other components

This component is used by all the components that need the OS services. The back-end interface is tightly integrated to the underlying OS for which the OSAL is implemented.

Chapter 2

Service APIs

2.1 Type Definitions

2.1.1 ClOsalSchedulePolicyT

typedef enum {
CL_OSAL_SCHED_OTHER,
CL_OSAL_SCHED_FIFO,
CL_OSAL_SCHED_RR
} ClOsalSchedulePolicyT;

The values of the *ClOsalSchedulePolicyT* enumeration type contains schedule policy of the tasks that will be created.

- *CL_OSAL_SCHED_OTHER* is the default scheduling policy.
- *CL_OSAL_SCHED_FIFO* and
- *CL_OSAL_SCHED_RR* is used for real time threads.

2.1.2 ClOsalProcessFlagT

typedef enum {
CL_OSAL_PROCESS_WITH_NEW_SESSION,
CL_OSAL_PROCESS_WITH_NEW_GROUP
} ClOsalProcessFlagT;

The values of the *ClOsalProcessFlagT* enumeration type contains the flags for the process creation. The values of this enumeration are:

- *CL_OSAL_PROCESS_WITH_NEW_SESSION*: This would create a process with new session.
- *CL_OSAL_PROCESS_WITH_NEW_GROUP*: This would create a new process group.

2.1.3 ClOsalThreadPriorityT

typedef enum {
CL_OSAL_THREAD_PRI_HIGH,
CL_OSAL_THREAD_PRI_MEDIUM,
CL_OSAL_THREAD_PRI_LOW
} ClOsalThreadPriorityT;

The *ClOsalThreadPriorityT* enumeration type contains the various thread priorities. The values of the ClOsalThreadPriorityT enumeration type have the following interpretation:

- *CL_OSAL_THREAD_PRI_HIGH* - Highest thread priority.
- *CL_OSAL_THREAD_PRI_MEDIUM* - Medium thread priority.
- *CL_OSAL_THREAD_PRI_LOW* - Lowest thread priority.

2.1.4 ClOsalPidT

typedef ClUint32T ClOsalPidT;

The type of an identifier for the OSAL Process ID.

2.1.5 ClOsalProcessFuncT

*typedef void(*ClOsalProcessFuncT)(void *);*

The type of the callback function invoked when a process is created.

2.1.6 ClOsalSemIdT

typedef ClUint32T ClOsalSemIdT;

The type the handle for the OSAL Semaphore ID.

2.1.7 ClOsalShmIdT

typedef ClUint32T ClOsalShmIdT;

The type of an identifier for the OSAL Shared Memory ID.

2.1.8 ClOsalTaskDataT

typedef ClHandleT ClOsalTaskDataT;

The type the handle for the OSAL Task Data type.

2.1.9 ClOsalTaskIdT

typedef ClUint64T ClOsalTaskIdT;

The type of an identifier for the OSAL Task ID.

2.1.10 ClOsalTaskKeyDeleteCallBackT

*typedef void(*ClOsalTaskKeyDeleteCallBackT)(void *);*

This is required while Thread or Task creation. Its value is 64 kilobytes.

2.1.11 ClOsalCondIdT

typedef ClHandleT ClOsalCondIdT;

The type of the handle for the OSAL condition ID.

2.1.12 ClOsalMutexIdT

typedef ClHandleT ClOsalMutexIdT;

The type of the handle for the OSAL Mutex ID.

2.1.13 ClOsalMutexT

typedef pthread_mutex_t ClOsalMutexT;

The type of the mutex that is to be initialized.

2.2 Library Life Cycle APIs

2.2.1 clOsalInitialize

clOsalInitialize

Synopsis:
Initializes the Operating System Abstraction Layer (OSAL).

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalInitialize(const ClPtrT pConfig);
```

Parameters:
None.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_NO_MEM: On memory allocation failure.

CL_ERR_MUTEX_CREATE: On failure in creating a mutex.

Description:
This function is used to initialize the OSAL. This must be the first function to be called before any of the other OSAL functions are invoked.

Library File:
ClOsal

Related Functions(s):
clOsalFinalize

2.2.2 clOsalFinalize

clOsalFinalize

Synopsis:
Cleans up the OSAL.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalFinalize(void);
```

Parameters:
Parameters: None.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_CL_OSAL_CLEANUP: On failure to clean up OSAL.

CL_ERR_MUTEX_DELETE: On failure in deleting mutex.

Description:
This function is used to clean-up OSAL. This must be the last OSAL function that is invoked, and it is typically called during system shutdown.

Library File:
ClOsal

Related Function(s):
clOsalInitialize

2.3 Functional APIs

2.3.1 clOsalTaskCreate

clOsalTaskCreate

Synopsis:
Creates a task.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalTaskCreate (CL_IN ClCharT *taskName,
       CL_IN ClOsalSchedulePolicyT schedulePolicy,
       CL_IN ClUint32T priority,
       CL_IN ClUint32T stackSize,
       CL_IN void* (*fpTaskFunction)(void*),
       CL_IN void* pTaskFuncArgument,
       CL_OUT ClOsalTaskIdT* pTaskId);
```

Parameters:

taskName: (in) Name of the task. This must be a valid string. NULL is regarded as invalid but the task creation does not fail.

schedulePolicy: (in) Schedule policy can be set as one of the following:

- `CL_OSAL_SCHED_RR`: For this, you must be logged in as super-user. It supports priority-based realtime round-robin scheduling.
- `CL_OSAL_SCHED_FIFO`: For this, you must be logged in as super-user. It supports priority based pre-emptive scheduling. This parameter is ignored in case of VxWorks.
- `CL_OSAL_SCHED_OTHER`

priority: (in) Priority at which the tasks is executed. The priority must be between 1 and 160 (1 - lowest priority 160 - highest priority). Any other value is regarded as wrong and task creation fails.

stackSize: (in) Size (in bytes) of the user stack that must be created when the task is executed.The stack size must be a positive integer. If zero is mentioned, then the default stack size of 4096 bytes would be allocated. If the stack size mentioned is less than the default stack size then the default stack size is assigned.

fpTaskFunction: (in) Entry point of the task. This function is executed when the task is executed. This is a function pointer and NULL argument would result in error.

pTaskFuncArgument: (in) The argument to be passed to *fpTaskFuncArgument* when the function is executed. This must be a valid pointer and if there are no arguments to be passed then NULL can be passed. Any other value will be invalid.

pTaskId: (out) The identifier of the task started is stored here. This shall be any valid pointer, NULL will be invalid.

Return values:

CL_RC_OK: This API executed successfully.

CL_ERR_NULL_PTR: On passing a NULL pointer.

CL_ERR_INVLD_PARAM: On passing invalid parameters.

CL_ERR_TASK_ATTRIBUTE_INIT: On failure in task attribute initialization.

CL_ERR_TASK_CREATE: On failure to create task.

CL_ERR_TASK_ATTRIBUTE_SET: On failure in setting task attributes.

Description:

This function is used to spawn a new task that runs concurrently with the calling task. The new task invokes the user function *fpTaskFunction* with the argument *pTaskFuncArgument* to the function. The new task terminates explicitly by calling clOsalTaskDelete or implicitly, by returning from the *fpTaskFunction*.

Library File:

ClOsal

Related Function(s):

clOsalTaskDelete, clOsalSelfTaskIdGet , clOsalTaskNameGet , clOsalTaskPriorityGet , clOsalTaskPrioritySet , clOsalTaskDelay

2.3.2 clOsalTaskDelete

clOsalTaskDelete

Synopsis:
Deletes a task.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalTaskDelete (
                    CL_IN ClOsalTaskIdT taskId);
```

Parameters:
TaskId: (in) Identifier of the task to be deleted. The *taskId* must be same as what was returned when the task was created. All other values will be invalid.

Return values:
CL_RC_OK: This API executed successfully. (Not applicable for deleting the currently executing task)

CL_ERR_NO_TASK_EXIST: On deleting a task which does not exist.

CL_ERR_TASK_DELETE: On failure to delete a task.

Description:
This function is used to delete a task identified by the *taskId*. If *taskId* is 0 the currently running task is terminated. The resources are released for the particular task deleted.

Library File:
ClOsal

Related Function(s):
clOsalTaskCreate, clOsalSelfTaskIdGet , clOsalTaskNameGet , clOsalTaskPriorityGet , clOsalTaskPrioritySet , clOsalTaskDelay

2.3.3 clOsalSelfTaskIdGet

clOsalSelfTaskIdGet

Synopsis:
Retrieves task id.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalSelfTaskIdGet (
                        CL_OUT ClOsalTaskIdT* pTaskId);
```

Parameters:
pTaskId: (out) Task ID of the calling task is stored here. This must be a valid pointer and cannot be NULL.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_NULL_PTR: On passing a NULL pointer.

Description:
This function is used to obtain the task Id of the calling task.

Library File:
ClOsal

Related Function(s):
clOsalTaskCreate, clOsalTaskDelete , clOsalTaskNameGet , clOsalTaskPriorityGet , clOsalTaskPrioritySet , clOsalTaskDelay

2.3.4 clOsalTaskNameGet

clOsalTaskNameGet

Synopsis:
Retrieves task name.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalTaskNameGet (CL_IN ClOsalTaskIdT taskId,
    CL_OUT ClUint8T** ppTaskName);
```

Parameters:

TaskId: (in) Task ID of the task for the which the name is to be found. If the task ID is zero then the name of the current task is found.

ppTaskName: (out) Name of the task is stored here. This must be a valid pointer and cannot be NULL.

Return values:

CL_RC_OK: This API executed successfully.

CL_ERR_NULL_PTR: On passing a NULL pointer.

CL_ERR_NO_TASK_EXIST: On requesting for a task name which does not exist.

Description:
This function is used to obtain the name of the task.

Library File:
ClOsal

Related Function(s):
clOsalTaskCreate , clOsalTaskDelete, clOsalSelfTaskIdGet , clOsalTaskPriorityGet, clOsalTaskPrioritySet , clOsalTaskDelay

2.3.5 clOsalTaskPriorityGet

clOsalTaskPriorityGet

Synopsis:
Retrieves the priority of the task.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalTaskPriorityGet(
          CL_IN ClOsalTaskIdT taskId,
          CL_OUT ClUint32T* pTaskPriority);
```

Parameters:

TaskId: (in) Task ID of the task for which the priority is to obtained. The task ID must be same as the one that was returned when the task was created. Any other value will be invalid.

pTaskPriority: (out) Priority of the task is stored here. This must be a valid pointer and cannot be NULL.

Return values:

CL_RC_OK: This API executed successfully.

CL_ERR_NULL_PTR: On passing a NULL pointer.

CL_ERR_TASK_ATTRIBUTE_GET: On failure in retrieving the priority of the task.

Description:
This function is used to obtain the priority of a task.

Library File:
ClOsal

Related Function(s):
clOsalTaskCreate, clOsalTaskDelete, clOsalSelfTaskIdGet, clOsalTaskNameGet , clOsalTaskPrioritySet , clOsalTaskDelay

2.3.6 clOsalTaskPrioritySet

clOsalTaskPrioritySet

Synopsis:
Sets the priority of the task.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalTaskPrioritySet (CL_IN ClOsalTaskIdT taskId,
    CL_IN ClUint32T taskPriority);
```

Parameters:

TaskId: (in) Task ID of the task for which the priority is to be set. The task ID must be same as the one that was returned when the task was created. Any other value will be invalid.

pTaskPriority: (in) New priority of the task. This must be between 1 and 160, 1 being the lowest and 160 highest priority. This parameter is ignored for CL_OSAL_SCHED_-OTHER. It applies only for the other scheduling policies.

Return values:

CL_RC_OK: This API executed successfully.

CL_ERR_NULL_PTR: On passing a NULL pointer.

CL_ERR_TASK_ATTRIBUTE_SET: On failure in setting the priority of the task.

Description:
This function is used to set the priority of a task.

Library File:
ClOsal

Related Function(s):
clOsalTaskCreate, clOsalTaskDelete, clOsalSelfTaskIdGet , clOsalTaskNameGet , clOsal-TaskPriorityGet, clOsalTaskDelay

2.3.7 clOsalTaskDelay

clOsalTaskDelay

Synopsis:
Delays a task for a specified duration.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalTaskDelay (
                    CL_IN ClTimerTimeOutT time);
```

Parameters:
time: (in) Time duration for which the task is delayed. This value is in milliseconds. Any positive value will be valid.

Return values:
CL_RC_OK: The API executed successfully.

CL_ERR_TASK_DELAY: On failure in delaying a task.

Description:
This function causes the calling task to relinquish the CPU for the duration specified. For certain Operating Systems like Linux the tasks may not get scheduled back after the time delay mentioned. It may be re-scheduled by the scheduler, based on the scheduling time interval. It can be interrupted by signals received by the process and could return before the delay time period.

Library File:
ClOsal

Related Function(s):
clOsalTaskCreate, clOsalTaskDelete, clOsalSelfTaskIdGet, clOsalTaskNameGet , clOsalTaskPriorityGet , clOsalTaskPrioritySet

2.3.8 clOsalTimeOfDayGet

clOsalTimeOfDayGet

Synopsis:
Retrieves the current time.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalTimeOfDayGet (
                        CL_OUT ClTimerTimeOutT* pTime);
```

Parameters:
pTime: (out) Pointer to variable of type *ClTimerTimeOutT*, in which the time is returned.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_NULL_PTR: On passing a NULL pointer.

CL_ERR_TIME_OF_DAY: On failure in retrieving the time of day.

Description:
This function is used to return the current time (in number of seconds and milliseconds) since the Epoch.

Library File:
ClOsal

Related Function(s):
None.

2.3.9 clOsalMutexInit

clOsalMutexInit

Synopsis:
Initializes a mutex.

Header File:
clOsalApi.h

Syntax:

```
   ClRcT clOsalMutexInit (
ClOsalMutexT* pMutex);
```

Parameters:
pMutex: (out) Mutex initialization data is stored here. This must be a valid pointer and cannot be NULL.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_NULL_PTR: On passing a NULL pointer.

CL_ERR_MUTEX_CREATE: On failure in creating a Mutex.

Description:
This API is used to initialize a mutual exclusion semaphore.

Library File:
ClOsal

Related Function(s):
clOsalMutexCreateAndLock , clOsalMutexCreate clOsalMutexDelete

2.3.10 clOsalMutexCreate

clOsalMutexCreate

Synopsis:
Creates a mutex.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalMutexCreate (
          CL_OUT ClOsalMutexIdT* pMutexId);
```

Parameters:
pMutexId: (out) Identifier of the mutex created is stored here. This must be a valid pointer and cannot be NULL.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_NULL_PTR: On passing a NULL pointer.

CL_ERR_NO_MEM: On memory allocation failure.

CL_ERR_MUTEX_CREATE: On failure in creating a Mutex.

Description:
This function is used to create and initialize a mutual exclusion semaphore.

Library File:
ClOsal

Related Function(s):
clOsalMutexCreateAndLock , clOsalMutexDelete

2.3.11 clOsalMutexCreateAndLock

clOsalMutexCreateAndLock

Synopsis:
Creates a mutex in locked state.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalMutexCreateAndLock (
                          CL_OUT ClOsalMutexIdT* pMutexId);
```

Parameters:
pMutexId: (out) Identifier of the mutex created is stored here. This must be a valid pointer and cannot be NULL.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_NULL_PTR: On passing a NULL pointer.

CL_ERR_NO_MEM: On memory allocation failure.

CL_ERR_MUTEX_CREATE: On failure in creating a mutex.

CL_ERR_MUTEX_LOCK: On failure in locking a mutex.

Description:
This function is used to create and initialize a mutual exclusion semaphore in locked state.

Library File:
ClOsal

Related Function(s):
clOsalMutexCreate , clOsalMutexDelete

2.3.12 clOsalMutexLock

clOsalMutexLock

Synopsis:
Locks a mutex.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalMutexLock (
                            ClOsalMutexIdT mutexId);
```

Parameters:
mutexId: (in) Identifier of the mutex to be locked. The mutex id must be same as one returned when the mutex was created. All other values will be invalid.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_INVLD_PARAM: On passing an invalid parameter.

CL_ERR_MUTEX_LOCK: On failure in locking a mutex.

Description:
This function is used to lock a mutex.

Library File:
ClOsal

Related Function(s):
clOsalMutexUnlock

2.3.13 clOsalMutexUnlock

clOsalMutexUnlock

Synopsis:
Unlocks a mutex.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalMutexUnlock (
                            CL_IN ClOsalMutexIdT mutexId);
```

Parameters:
mutexId: (in) Identifier of the mutex to be unlocked. The mutex ID must be same as one returned when the mutex was created. All other values will be invalid.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_INVLD_PARAM: On passing an invalid parameter.

CL_ERR_MUTEX_UNLOCK: On failure in unlocking a mutex.

Description:
This function is used to unlock a mutex.

Library File:
ClOsal

Related Function(s):
clOsalMutexLock

2.3.14 clOsalMutexDelete

clOsalMutexDelete

Synopsis:
Deletes a mutex.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalMutexDelete (
                          CL_IN ClOsalMutexIdT mutexId);
```

Parameters:
mutexId: (in) Identifier of the mutex to be deleted. The mutex id must be same as one returned when the mutex was created. All other values will be invalid.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_INVLD_PARAM: On passing an invalid parameter.

CL_ERR_MUTEX_DELETE: On failure in deleting a mutex.

Description:
This function is used to delete a mutex. Any other operation on the mutex is undefined after this invocation.

Library File:
ClOsal

Related Function(s):
clOsalMutexCreate , clOsalMutexCreateAndLock

2.3.15 clOsalMutexDestroy

clOsalMutexDestroy

Synopsis:
Destroys a mutex.

Header File:
clOsalApi.h

Syntax:
```
ClRcT clOsalMutexDestroy (
                              ClOsalMutexT *pMutex);
```

Parameters:
p_Mutex: (in) Mutex to be destroyed. The mutex must be same as one returned when the mutex was initialized. All other values will be invalid.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_INVLD_PARAM: On passing an invalid parameter.

CL_ERR_MUTEX_DELETE: On failure in destroying the mutex.

Description:
This API is used to destroy a mutex.

Library File:
ClOsal

Related Function(s):
clOsalMutexCreate, clOsalMutexInit, clOsalMutexCreateAndLock

2.3.16 clOsalNumMemoryAllocGet

clOsalNumMemoryAllocGet

Synopsis:
Returns the number of memory allocations.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalNumMemoryAllocGet (
                          CL_OUT ClSizeT* pMemAlloc);
```

Parameters:
pMemAlloc: (out) Number of memory allocated is stored here. This must be a valid pointer and cannot be NULL.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_NULL_PTR: On passing a NULL pointer.

Description:
This function is used to obtain the number of memory allocations performed by the component.

Library File:
ClOsal

Related Function(s):
clOsalNumMemoryDeallocedGet

2.3.17 clOsalNumMemoryDeallocedGet

clOsalNumMemoryDeallocedGet

Synopsis:
Retrieves the number of memory blocks freed.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalNumMemoryDeallocedGet (
                              CL_OUT ClSizeT* pMemFreed);
```

Parameters:
pMemFreed: (out) Number of memory blocks freed is stored here. This must be a valid pointer and cannot be NULL.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_NULL_PTR: On passing a NULL pointer.

Description:
This function is used to obtain the number of memory blocks freed by the component.

Library File:
ClOsal

Related Function(s):
clOsalNumMemoryAllocGet

2.3.18 clOsalCondCreate

clOsalCondCreate

Synopsis:
Creates a condition variable.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalCondCreate (
                        CL_OUT ClOsalCondIdT* pConditionId);
```

Parameters:
pConditionId: (out) Identifier of the condition variable created is stored here. This must be a valid pointer and cannot be NULL.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_NULL_PTR: On passing a NULL pointer.

CL_ERR_NO_MEM: On memory allocation failure.

CL_ERR_CONDITION_CREATE: On failure in creating a condition variable.

Description:
This function is used to create a condition variable.

Library File:
ClOsal

Related Function(s):
clOsalCondDelete , clOsalCondWait , clOsalCondBroadcast , clOsalCondSignal

2.3.19 clOsalCondDelete

clOsalCondDelete

Synopsis:
Deletes a condition variable if no task is waiting on this condition.

Header File:
clOsalApi.h

Syntax:
```
ClRcT clOsalCondDelete (
                        CL_IN ClOsalCondIdT conditionId);
```

Parameters:
conditionId: (in) Identifier of the condition variable to be deleted. No task must be waiting on this condition variable when delete is invoked. This must be the same as one returned when the condition variable was created. Any other value will be invalid.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_CONDITION_TIMEDOUT: On condition timedout

CL_ERR_INVLD_PARAM: On passing an invalid parameter.

CL_ERR_CONDITION_DELETE: On failure in deleting the condition variable.

Description:
This function is used to delete a condition variable. The condition variable becomes invalid after this invocation and the behavior of any task depending on this condition variable is undefined.

Library File:
ClOsal

Related Function(s):
clOsalCondCreate , clOsalCondWait , clOsalCondBroadcast , clOsalCondSignal

2.3.20 clOsalCondWait

clOsalCondWait

Synopsis:
Waits for a condition to be signalled.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalCondWait (
                      CL_IN ClOsalCondIdT conditionId,
                      CL_IN ClOsalMutexIdT mutexId,
                      CL_IN ClTimerTimeOutT time);
```

Parameters:

conditionId: (in) Identifier of the condition variable. This must be the same as one returned when the condition variable was created. Any other value can will be invalid.

mutexId: (in) Identifier of the mutex. This must be the same as one returned when the mutex was created. Any other value will be invalid.

time: (in) Duration for which the task waits for a condition. The time structure must be filled in with valid values. If the time-out is zero then the task waits indefinitely for the signal.

Return values:

CL_RC_OK: This API executed successfully.

CL_ERR_INVLD_PARAM: On passing an invalid parameter.

CL_ERR_CONDITION_WAIT: On failure to wait for a condition.

Description:
This function is used to unlock a mutex and wait on a condition variable to be signalled as the task execution is suspended. The mutex must be locked before entering into wait.

- If the time out value is specified then the task would wait for the specified time.
- If the condition is not signalled within the specified time then an error is returned.
- If the timeout specified is zero then the task waits indefinitely until it receives a signal. The mutex is re-acquired before returning to the calling task.

Library File:
ClOsal

Related Function(s):
clOsalCondCreate , clOsalCondDelete , clOsalCondBroadcast , clOsalCondSignal

2.3.21 clOsalCondBroadcast

clOsalCondBroadcast

Synopsis:
Broadcasts a condition.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalCondBroadcast (
                        CL_IN ClOsalCondIdT conditionId);
```

Parameters:
conditionId: (in) Identifier of the condition variable. This must be the same as one returned when the condition variable was created. Any other value will be invalid.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_INVLD_PARAM: On passing an invalid parameter.

CL_ERR_CONDITION_BROADCAST: On failure to broadcast the condition.

Description:
This function is used to resume all the tasks that are waiting on the condition variable. Nothing happens if no task is waiting on that condition.

Library File:
ClOsal

Related Function(s):
clOsalCondCreate , clOsalCondDelete , clOsalCondWait , clOsalCondSignal

2.3.22 clOsalCondSignal

clOsalCondSignal

Synopsis:
Signals the occurrence of a condition.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalCondSignal (
                        CL_IN ClOsalCondIdT conditionId);
```

Parameters:
conditionId: (in) Identifier of the condition variable. This must be the same as one returned when the condition variable was created. Any other value will be invalid.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_INVLD_PARAM: On passing an invalid parameter.

CL_ERR_CONDITION_SIGNAL: On failure to signal a condition.

Description:
This function is used to resume one of the tasks that are waiting on the condition variable. If no tasks are waiting on that condition then nothing happens. If there are several threads waiting on the condition then exactly one task is resumed.

Library File:
ClOsal

Related Function(s):
clOsalCondCreate , clOsalCondDelete , clOsalCondWait , clOsalCondBroadcast

2.3.23 clOsalTaskKeyCreate

clOsalTaskKeyCreate

Synopsis:
Creates a key for thread-specific data.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalTaskKeyCreate(
                              CL_OUT ClUint32T* pKey,
                              CL_IN void(*ClOsalTaskKeyDeleteCallBackT)(void*));
```

Parameters:

pKey: (out) Pointer to ClUint32T, in which the created key is returned.

ClOsalTaskKeyDeleteCallBackT: (in) Callback function, which is called when the key is deleted using clOsalTaskKeyDelete API.

Return values:

CL_RC_OK: This API executed successfully.

CL_ERR_NULL_PTR: On passing a NULL pointer.

CL_OS_ERROR: On failure to create a key for thread-specific data.

Description:
This function is used to create a key for thread-specific data. Different threads can use this key to store their thread-specific data.

Library File:
ClOsal

Related Function(s):
clOsalTaskKeyDelete, clOsalTaskDataSet , clOsalTaskDataGet

2.3.24 clOsalTaskKeyDelete

clOsalTaskKeyDelete

Synopsis:
Deletes the key for thread-specific data.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalTaskKeyDelete(
                        CL_IN ClUint32T key);
```

Parameters:
key: (in) Key to be deleted.

Return values:
CL_RC_OK: This API executed successfully.
CL_OS_ERROR: On failure in deleting the key for thread-specific data.

Description:
This function is used to delete the key for thread-specific data, which is returned by clOsalTaskKeyCreate.

Library File:
ClOsal

Related Function(s):
clOsalTaskKeyCreate , clOsalTaskDataSet , clOsalTaskDataGet

2.3.25 clOsalTaskDataSet

clOsalTaskDataSet

Synopsis:
Sets the thread-specific data.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalTaskDataSet(
       CL_IN ClUint32T key,
           CL_IN ClOsalTaskDataT threadData);
```

Parameters:
key: (in) Key for the thread-specific data returned by clOsalTaskKeyCreate API.

threadData: (in) *threadData* to be assigned to the calling task.

Return values:
CL_RC_OK: This API executed successfully.

CL_OS_ERROR: On failure to set thread-specific data.

Description:
This function is used to set the thread-specific data for the calling task.

Library File:
ClOsal

Related Function(s):
clOsalTaskKeyCreate, clOsalTaskKeyDelete , clOsalTaskDataGet

2.3.26 clOsalTaskDataGet

clOsalTaskDataGet

Synopsis:
Retrieves the thread-specific data.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalTaskDataGet(
                        CL_IN ClUint32T key,
                        CL_OUT ClOsalTaskDataT* pThreadData);
```

Parameters:

key: (in) Key for the thread-specific data returned by clOsalTaskKeyCreate API.

pThreadData: (out) Pointer to ClUint32T in which the thread-specific data is returned on success.

Return values:

CL_RC_OK: This API executed successfully.

CL_ERR_NULL_PTR: On passing a NULL pointer.

CL_OS_ERROR: On failure in retrieving the thread-specific data.

Description:
This function is used to return the thread-specific data of the calling task associated with this key.

Library File:
ClOsal

Related Function(s):
clOsalTaskKeyCreate, clOsalTaskKeyDelete , clOsalTaskDataSet

2.3.27 clOsalPrintf

clOsalPrintf

Synopsis:
Prints to the standard output.

Header File:
clOsalApi.h

Syntax:

```
clRcT clOsalPrintf(
                    CL_IN const ClCharT* fmt, ...);
```

Parameters:
fmt: (in) Format string to be printed.

Return values:
CL_RC_OK: This API executed successfully.

CL_OS_ERROR: On failure to print to standard output.

Description:
This function is used to print to the standard output.

Library File:
ClOsal

Related Function(s):
None.

2.3.28 clOsalSemCreate

clOsalSemCreate

Synopsis:
Creates a semaphore.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalSemCreate(
       CL_IN ClUint8T* pName,
       CL_IN ClUint32T value,
       CL_OUT ClOsalSemIdT* pSemId);
```

Parameters:

pName: (in)Name of the semaphore to be created. If the same name is specified then the same semaphore ID is returned. Name must not be more than 20 characters and cannot be NULL.

value: (in) Value of the semaphore. This is required to set the semaphore before it is used first time. This must be a positive integer and must be less than `CL_SEM_MAX_VALUE`. Zero will be invalid.

pSemId: (out) Identifier of the semaphore created is stored in this location. This must be a valid pointer and cannot be NULL.

Return values:

CL_RC_OK: This API executed successfully.

CL_ERR_INVLD_PARAM: On passing an invalid parameter.

CL_ERR_SEM_CREATE: On failure in creating a semaphore.

CL_ERR_NAME_TOO_LONG: If semaphore name is too long.

CL_ERR_NULL_PTR: On passing a NULL pointer.

Description:
This function is used to create an counting semaphore. If the semaphore with the same name exists then the existing semaphore is obtained else a new semaphore is created.

Library File:
ClOsal

Related Function(s):
clOsalSemIdGet , clOsalSemLock , clOsalSemTryLock , clOsalSemUnlock , clOsalSemValueGet , clOsalSemDelete

2.3.29 clOsalSemIdGet

clOsalSemIdGet

Synopsis:
Retrieves the semaphore id.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalSemIdGet(
                    CL_IN ClUint8T* pName,
                    CL_OUT ClOsalSemIdT* pSemId);
```

Parameters:

pName: (in) Name of the semaphore for the which the ID is required.NULL is not valid.

pSemId: (out) Name of the semaphore for the which the ID is required. NULL will not be valid.

Return values:

CL_RC_OK: This API executed successfully.

CL_ERR_INVLD_PARAM: On passing an invalid parameter.

CL_ERR_NULL_PTR: On passing a NULL pointer.

CL_ERR_SEM_ID_GET: On failure in retrieving the semaphore ID.

Description:
This function is used to obtain the ID of the semaphore if the name is specified.

Library File:
ClOsal

Related Function(s):
clOsalSemCreate , clOsalSemLock , clOsalSemTryLock , clOsalSemUnlock , clOsalSemValueGet, clOsalSemDelete

2.3.30 clOsalSemLock

clOsalSemLock

Synopsis:
Locks a semaphore.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalSemLock(
                    CL_IN ClOsalSemIdT semId);
```

Parameters:
semId: (in) Identifier of the semaphore to be locked. This must be the same as what was returned when the semaphore was created. Any other value will be invalid.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_INVLD_PARAM: On passing an invalid parameter.

CL_ERR_SEM_LOCK: On failure in locking a semaphore.

Description:
This function is used to lock a semaphore. If the semaphore is already locked (unavailable) then trying to lock the same semaphore suspends the execution of the process until the semaphore is available. The value of the semaphore is decremented.

Library File:
ClOsal

Related Function(s):
clOsalSemCreate , clOsalSemIdGet, clOsalSemTryLock , clOsalSemUnlock , clOsalSemValueGet, clOsalSemDelete

2.3.31 clOsalSemTryLock

clOsalSemTryLock

Synopsis:
Locks a semaphore if it is available.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalSemTryLock(
                          CL_IN ClOsalSemIdT semId);
```

Parameters:
semId: (in) Identifier of the semaphore to be locked. This must be the same as one returned when the semaphore was created. Any other value will be invalid.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_INVLD_PARAM: On passing an invalid parameter.

CL_ERR_SEM_LOCK: On failure in try-locking a semaphore.

Description:
This function is used to lock a semaphore if the semaphore is available. If the semaphore is unavailable then the call just returns without blocking. The call returns immediately (non-blocking) on both the cases. If the semaphore is available, then semaphore is locked and will remain locked until it is unlocked explicitly. The value of the semaphore is decremented.

Library File:
ClOsal

Related Function(s):
clOsalSemCreate , clOsalSemIdGet, clOsalSemLock , clOsalSemUnlock, clOsalSemValue-Get , clOsalSemDelete

2.3.32 clOsalSemUnlock

clOsalSemUnlock

Synopsis:
Unlocks a semaphore.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalSemUnlock(
                        CL_IN ClOsalSemIdT semId);
```

Parameters:
semId: (in) Identifier of the semaphore to be locked. This must be the same as one returned when the semaphore was created. Any other value will be invalid.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_INVLD_PARAM: On passing an invalid parameter.

CL_ERR_SEM_UNLOCK: On failure in unlocking a semaphore.

Description:
This function is used to unlock a semaphore.Any process that is blocked on the semaphore resumes execution. If no process is blocked then the semaphore is made available for the others. The value of the semaphore is incremented.

Library File:
ClOsal

Related Function(s):
clOsalSemCreate, clOsalSemIdGet , clOsalSemLock, clOsalSemTryLock , clOsalSemValue-Get, clOsalSemDelete

2.3.33 clOsalSemValueGet

clOsalSemValueGet

Synopsis:
Retrieves the value of a semaphore.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalSemValueGet(
                    CL_IN ClOsalSemIdT semId,
                    CL_OUT ClUint32T* pSemValue);
```

Parameters:

semId: (in) Identifier of the semaphore to be locked. This must be the same as one returned when the semaphore was created. Any other value will be invalid.

pSemValue: (out) Value of the semaphore is stored in the location specified. This must be a valid pointer and cannot be NULL.

Return values:

CL_RC_OK: This API executed successfully.

CL_ERR_NULL_PTR: On passing a NULL pointer.

CL_ERR_INVLD_PARAM: On passing an invalid parameter.

CL_ERR_SEM_GET_VALUE: On failure in retrieving the value of the semaphore.

Description:
This function is used to return the value of a semaphore.

Library File:
ClOsal

Related Function(s):
clOsalSemCreate, clOsalSemIdGet , clOsalSemLock, clOsalSemTryLock , clOsalSemUnlock , clOsalSemDelete

2.3.34 clOsalSemDelete

clOsalSemDelete

Synopsis:
Deletes a semaphore.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalSemDelete(
                    CL_IN ClOsalSemIdT semId);
```

Parameters:
semId: (in) Identifier of the semaphore to be locked. This must be the same as one returned when the semaphore was created. Any other value will be invalid.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_INVLD_PARAM: On passing an invalid parameter.

CL_ERR_SEM_DELETE: On failure in deleting a semaphore.

Description:
This function is used to destroy a semaphore. No process must be waiting for the semaphore while the semaphore is being destroyed. If one or more processes are waiting on the semaphore then the semaphore does not get destroyed.

Library File:
ClOsal

Related Function(s):
clOsalSemCreate, clOsalSemIdGet, clOsalSemLock , clOsalSemTryLock, clOsalSemUnlock , clOsalSemValueGet

2.3.35 clOsalProcessCreate

clOsalProcessCreate

Synopsis:
Creates a process.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalProcessCreate(
           CL_IN ClOsalProcessFuncT fpFunction,
           CL_IN void* functionArg,
           CL_IN ClOsalProcessFlagT creationFlags,
           CL_OUT ClOsalPidT* pProcessId);
```

Parameters:

fpFunction: (in) Function to be invoked when the process is created. This must be a valid pointer and cannot be NULL.

functionArg: (in) Argument passed to the *fpFunction* that is created along with the creation of the process. This can be a valid pointer or NULL.

creationFlags: (in) Flags that control certain properties including creation of new session, process group and creating the process in a suspended state.

pProcessId: (out) Id of the process created is stored in this location. This must be a valid pointer and cannot be NULL.

Return values:

CL_RC_OK: This API executed successfully.

CL_ERR_NULL_PTR: On passing a NULL pointer.

CL_ERR_PROCESS_CREATE: On failure to create a process.

Description:
This function is used to create a process.

Library File:
ClOsal

Related Function(s):
clOsalProcessDelete, clOsalProcessWait , clOsalProcessSelfIdGet

2.3.36 cosProcessResume

cosProcessResume

Synopsis:
Resumes a process.

Header File:
clOsalApi.h

Syntax:

```
ClRcT cosProcessResume(
                        ClOsalPidT processId);
```

Parameters:
processId: (in) Id of the process to be resumed.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_INVALID_PARAM: On passing an invalid parameter.

Description:
This API is used to resume the process that is suspended.

Library File:
ClOsal

Related Function(s):
None.

2.3.37 cosProcessSuspend

cosProcessSuspend

Synopsis:
Suspends a process.

Header File:
clOsalApi.h

Syntax:

```
ClRcT cosProcessSuspend(
                            ClOsalPidT processId);
```

Parameters:
processId: (in) Id of the process to be resumed.

Return values:
CL_RC_OK: This API executed successfully.
CL_ERR_INVALID_PARAM: On passing an invalid parameter.

Description:
This API is used to suspend any process from execution.

Library File:
ClOsal

Related Function(s):
None.

2.3.38 clOsalProcessDelete

clOsalProcessDelete

Synopsis:
Deletes a process.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalProcessDelete(
                        CL_IN ClOsalPidT processId);
```

Parameters:
processId: (in) Id of the process to be deleted. This must be the same as one returned when the process was created.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_PROCESS_DELETE: On failure in deleting a process.

Description:
This function is used to delete a process. The invocation of this function sends a `SIGKILL` signal to the process identified by the *processId*.

Library File:
ClOsal

Related Function(s):
clOsalProcessCreate, clOsalProcessWait , clOsalProcessSelfIdGet

2.3.39 clOsalProcessWait

clOsalProcessWait

Synopsis:
Waits for a process to exit.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalProcessWait(
                        CL_IN ClOsalPidT processId);
```

Parameters:
processId: (in) Id of the process for which the calling process needs to wait. This must be the same as one returned when the process was created. If zero is passed then the calling process waits for all the processes, that it created, to exit.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_PROCESS_WAIT: On failure to wait for a process to exit.

Description:
This function is used to suspend the execution of a process until the processes created from within the process has exited.

$<$ -1 which means to wait for any child process whose process group ID is equal to the absolute value of pid

-1 which means to wait for any child process; this is the same behavior which wait exhibits.

0 which means to wait for any child process whose process group ID is equal to that of the calling process.

$>$ 0 which means to wait for the child whose process ID is equal to the value of pid.

Library File:
ClOsal

Related Function(s):
clOsalProcessCreate , clOsalProcessDelete, clOsalProcessSelfIdGet

2.3.40 clOsalProcessSelfIdGet

clOsalProcessSelfIdGet

Synopsis:
Retrieves the processId.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalProcessSelfIdGet(
                    CL_OUT ClOsalPidT* pProcessId);
```

Parameters:
pProcessId: (out) ProcessId of the caller is stored in this location.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_NULL_PTR: On passing a NULL pointer.

Description:
This function is used to obtain the *processId* of the caller.

Library File:
ClOsal

Related Function(s):
clOsalProcessCreate , clOsalProcessDelete , clOsalProcessWait

2.3.41 clOsalShmCreate

clOsalShmCreate

Synopsis:
Creates a shared memory.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalShmCreate(
            CL_IN ClUint8T* pName,
            CL_IN ClUint32T       size,
            CL_OUT ClOsalShmIdT    *pShmId);
```

Parameters:

pName: (in) Name for the shared memory region to be created. The same name must be specified if the same shared memory is to be used from some other process. This must be a valid pointer and cannot be NULL. Name must not be more than 20 characters.

size: (in) Size of the shared memory to be created. This must be a positive integer.

pShmId: (out) Identifier of the shared memory created is stored in this location. This must be a valid pointer and cannot be NULL.

Return values:

CL_RC_OK: This API executed successfully.

CL_ERR_INVLD_PARAM: On passing an invalid parameter.

CL_ERR_NULL_PTR: On passing a NULL pointer.

CL_ERR_NAME_TOO_LONG: If the name specified is too long.

CL_ERR_SHM_ID_GET: On failure to retrieve the ID of the shared memory.

CL_ERR_SHM_CREATE: On failure to create a shared memory.

Description:
This function is used to create a shared memory region. If the shared memory region with the same name exists then the existing shared memory is obtained, else a new shared memory region is created.

Library File:
ClOsal

Related Function(s):
clOsalShmIdGet , clOsalShmDelete, clOsalShmAttach , clOsalShmDetach , clOsalShmSecurityModeSet , clOsalShmSecurityModeGet, clOsalShmSizeGet

2.3.42 clOsalShmIdGet

clOsalShmIdGet

Synopsis:
Retrieves a shared memory Id.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalShmIdGet(
                CL_IN ClUint8T    *pName,
                CL_OUT ClOsalShmIdT    *pShmId)
```

Parameters:

pName: (in) Name of the shared memory region for which the Id is to be obtained. This must be a valid pointer and cannot be NULL.

pShmId: (out) Id of the shared memory is copied into this location. NULL will be invalid.

Return values:

CL_RC_OK: The API executed successfully.

CL_ERR_INVLD_PARAM: On passing an invalid parameter.

CL_ERR_NULL_PTR: On passing a NULL pointer.

CL_ERR_SHM_ID_GET: On failure in obtaining the shared memory ID.

Description:
This function returns the identifier of the shared memory segment associated with *pName*.

Library File:
ClOsal

Related Function(s):
clOsalShmCreate , clOsalShmDelete , clOsalShmAttach , clOsalShmDetach , clOsalShmSecurityModeSet , clOsalShmSecurityModeGet , clOsalShmSizeGet

2.3.43 clOsalShmDelete

clOsalShmDelete

Synopsis:
Deletes a shared memory.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalShmDelete(
                        CL_IN ClOsalShmIdT shmId);
```

Parameters:
shmId: (in) Id of the shared memory to be deleted. This must be the same as the one that was passed when the shared memory was created. Any other value will be invalid.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_INVLD_PARAM: On passing an invalid parameter.

CL_ERR_SHM_DELETE: On failure in deleting the shared memory.

Description:
This function is used to delete a shared memory region. If more than one process has attached this shared memory then *clOsalShmDelete()* would just mark it for destroy. The shared memory is actually destroyed only after the last detach.

Library File:
ClOsal

Related Function(s):
clOsalShmCreate , clOsalShmIdGet , clOsalShmAttach , clOsalShmDetach , clOsalShmSecurityModeSet , clOsalShmSecurityModeGet , clOsalShmSizeGet

2.3.44 clOsalShmAttach

clOsalShmAttach

Synopsis:
Attaches a shared memory.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalShmAttach(
                    CL_IN ClOsalShmIdT shmId,
                    CL_IN void* pInMem,
                    CL_OUT void** ppOutMem);
```

Parameters:

shmId: (in) Identifier of the shared memory region to be attached. This must be same as one returned the shared memory was created. All other values will be invalid.

pInMem: (in) Pointer to the shared memory region to be attached to the address space of the calling processes.

- If *pInMem* is NULL then any unused address is attached.
- If a valid address is passed then the attach occurs at the address equal to *pInMem* rounded to the nearest multiple of pagesize.

ppOutMem: (out) Attached shared memory region is copied to this location. This must be a valid pointer and cannot be NULL.

Return values:

CL_RC_OK: This API executed successfully.

CL_ERR_INVLD_PARAM: On passing an invalid parameter.

CL_ERR_NULL_PTR: On passing a NULL pointer.

CL_ERR_SHM_ATTACH: On failure in attaching the shared memory.

Description:
This function is used to attach a shared memory region. The attach occurs at the address equal to the address specified by you rounded to the nearest multiple of the page size.

Library File:
ClOsal

Related Function(s):
clOsalShmCreate , clOsalShmIdGet, clOsalShmDelete, clOsalShmDetach , clOsalShmSecurityModeSet, clOsalShmSecurityModeGet, clOsalShmSizeGet

2.3.45 clOsalShmDetach

clOsalShmDetach

Synopsis:
Detaches a shared memory.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalShmDetach(
                        CL_IN void* pMem);
```

Parameters:
pMem: (in) Pointer to the shared memory region to be detached. The memory must have been attached to the process already. NULL will be invalid.

Return values:
CL_RC_OK: This API executed successfully.

CL_ERR_NULL_PTR: On passing a NULL pointer.

CL_ERR_SHM_DETACH: On failure in detaching shared memory.

Description:
This function is used to detach a shared memory region from the address space of the calling process. The memory segment to be detached must have been attached to the process by a call to *clOsalShmAttach*.

Library File:
ClOsal

Related Function(s):
clOsalShmCreate, clOsalShmIdGet , clOsalShmDelete , clOsalShmAttach , clOsalShmSecurityModeSet , clOsalShmSecurityModeGet , clOsalShmSizeGet

2.3.46 clOsalShmSecurityModeSet

clOsalShmSecurityModeSet

Synopsis:
Sets permissions to shared memory.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalShmSecurityModeSet(
                            CL_IN ClOsalShmIdT shmId,
                            CL_IN ClUint32T mode);
```

Parameters:

shmId: (in) Identifier of the shared memory region. This must be same as one returned when the shared memory was created. All other values will be invalid.

mode: (in) Permission to be set. This can be any of the flags specified in *ClOsalShmSecurityModeFlagT*.

Return values:

CL_RC_OK: This API executed successfully.

CL_ERR_INVLD_PARAM: On passing an invalid parameter.

CL_ERR_SHM_MODE_SET: On failure in setting permissions to shared memory.

Description:
This function is used to set permissions for a shared memory region. This allows only the access modes to be changed. Mode is invalid if it is anything above 0777(read, write, execute permission for user, group and others).

Library File:
ClOsal

Related Function(s):
clOsalShmCreate , clOsalShmIdGet , clOsalShmDelete , clOsalShmAttach , clOsalShmDetach clOsalShmSecurityModeGet , clOsalShmSizeGet

2.3.47 clOsalShmSecurityModeGet

clOsalShmSecurityModeGet

Synopsis:
Retrieves permissions of shared memory.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalShmSecurityModeGet(
                CL_IN ClOsalShmIdT shmId,
                CL_OUT ClUint32T* pMode);
```

Parameters:

shmId: (in) Identifier of the shared memory region. This must be same as one returned when the shared memory was created. All other values will be invalid.

pMode: (out) Permission of the shared memory will be copied to this location. This can be any of the flags specified in *ClOsalShmSecurityModeFlagT*. NULL will be invalid.

Return values:

CL_RC_OK: This API executed successfully.

CL_ERR_INVLD_PARAM: On passing an invalid parameter.

CL_ERR_NULL_PTR: On passing a NULL pointer.

CL_ERR_SHM_MODE_GET: On failure in retrieving permissions of the shared memory.

Description:
This function is used to return the permissions of the shared memory region.

Library File:
ClOsal

Related Function(s):
clOsalShmCreate , clOsalShmIdGet , clOsalShmDelete , clOsalShmAttach , clOsalShm-Detach clOsalShmSecurityModeGet , clOsalShmSizeGet

2.3.48 clOsalShmSizeGet

clOsalShmSizeGet

Synopsis:
Retrieves the size of shared memory.

Header File:
clOsalApi.h

Syntax:

```
ClRcT clOsalShmSizeGet(
                            CL_IN ClOsalShmIdT shmId,
                            CL_OUT ClUint32T* pSize);
```

Parameters:

shmId: (in) Identifier of the shared memory region. This must be same as one returned when the shared memory was created. All other values will be invalid.

pSize: (out) Size of the shared memory is copied into the specified location.

Return values:

CL_RC_OK: This API executed successfully.

CL_ERR_INVLD_PARAM: On passing an invalid parameter.

CL_ERR_NULL_PTR: On passing a NULL pointer.

CL_ERR_SHM_SIZE: On failure in retrieving size of the shared memory.

Description:
This function is used to obtain the size of the shared memory that was allocated.

Library File:
ClOsal

Related Function(s):,
clOsalShmCreate clOsalShmIdGet , clOsalShmDelete , clOsalShmAttach , clOsalShmDetach , clOsalShmSecurityModeSet , clOsalShmSecurityModeGet

2.3.49 clOsalSigHandlerInitialize

clOsalSigHandlerInitialize

Synopsis:
Installs and initializes the signal handler.

Header File:
clOsalApi.h

Syntax:

```
void clOsalSigHandlerInitialize();
```

Parameters:
None.

Return Values:
None.

Description:
This function is used to install and initialize the signal handler.

Library File:
ClOsal

Related Function(s):
None.

2.3.50 clOsalErrorReportHandler

clOsalErrorReportHandler

Synopsis:
Reports the message provided by the components to the proper channel.

Header File:
clOsalApi.h

Syntax:

```
void clOsalErrorReportHandler(
                            CL_IN void *info);
```

Parameters:
info: (in) Information to be reported.

Return Values:
None.

Description:
This function is used to report the message provided by the components to the proper channel.

Library File:
ClOsal

Related Function(s):
None.

Index