HeliOS

Kernel 0.4.0

HeliOS Developer's Guide

1 Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

2 File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

3 Data Structure Documentation

3.1 MemoryRegionStats_s Struct Reference

Data structure for memory region statistics.

Data Fields

- Word_t largestFreeEntryInBytes
- Word_t smallestFreeEntryInBytes
- Word_t numberOfFreeBlocks
- Word_t availableSpaceInBytes
- Word_t successfulAllocations
- Word_t successfulFrees
- Word_t minimumEverFreeBytesRemaining

3.1.1 Detailed Description

The MemoryRegionStats_t data structure is used by xMemGetHeapStats() and xMemGetKernelStats() to obtain statistics about either memory region.

See also

xMemoryRegionStats
xMemGetHeapStats()
xMemGetKernelStats()
xMemFree()

3.1.2 Field Documentation

3.1.2.1 availableSpaceInBytes `Word_t MemoryRegionStats_s::availableSpaceInBytes`

The amount of free memory in bytes (i.e., numberOfFreeBlocks * CONFIG_MEMORY_REGION_BLOCK_SIZE).

3.1.2.2 largestFreeEntryInBytes `Word_t MemoryRegionStats_s::largestFreeEntryInBytes`

The largest free entry in bytes.

3.1.2.3 minimumEverFreeBytesRemaining `Word_t MemoryRegionStats_s::minimumEverFreeBytesRemaining`

Lowest water lever since system initialization of free bytes of memory.

3.1.2.4 numberOfFreeBlocks `Word_t MemoryRegionStats_s::numberOfFreeBlocks`

The number of free blocks. See CONFIG_MEMORY_REGION_BLOCK_SIZE for block size in bytes.

3.1.2.5 smallestFreeEntryInBytes `Word_t MemoryRegionStats_s::smallestFreeEntryInBytes`

The smallest free entry in bytes.

3.1.2.6 successfulAllocations `Word_t MemoryRegionStats_s::successfulAllocations`

Number of successful memory allocations.

3.1.2.7 successfulFrees `Word_t MemoryRegionStats_s::successfulFrees`

Number of successful memory "frees".

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

- HeliOS.h

3.2 QueueMessage_s Struct Reference

Data structure for a queue message.

Data Fields

- Base_t messageBytes
- Byte_t messageValue [0x8u]

3.2.1 Detailed Description

The QueueMessage_t stucture is used to store a queue message and is returned by xQueueReceive() and xQueuePeek().

See also

xQueueMessage

xQueueReceive()

xQueuePeek()

CONFIG_MESSAGE_VALUE_BYTES

xMemFree()

3.2.2 Field Documentation

3.2.2.1 messageBytes `Base_t QueueMessage_s::messageBytes`

The number of bytes contained in the message value which cannot exceed CONFIG_MESSAGE_VALUE_BYTES.

3.2.2.2 messageValue `Byte_t QueueMessage_s::messageValue[0x8u]`

The queue message value.

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

- HeliOS.h

3.3 SystemInfo_s Struct Reference

Data structure for information about the HeliOS system.

Data Fields

- Byte_t productName [0x6u]
- Base_t majorVersion
- Base_t minorVersion
- Base_t patchVersion
- Base_t numberOfTasks

3.3.1 Detailed Description

The SystemInfo_t data structure is used to store information about the HeliOS system and is returned by xSystemGetSystemInfo().

See also

xSystemInfo

xSystemGetSystemInfo()

OS_PRODUCT_NAME_SIZE

xMemFree()

3.3.2 Field Documentation

3.3.2.1 majorVersion `Base_t SystemInfo_s::majorVersion`

The SemVer major version number of HeliOS.

3.3.2.2 minorVersion `Base_t SystemInfo_s::minorVersion`

The SemVer minor version number of HeliOS.

3.3.2.3 numberOfTasks `Base_t SystemInfo_s::numberOfTasks`

The number of tasks regardless of their state.

3.3.2.4 patchVersion `Base_t SystemInfo_s::patchVersion`

The SemVer patch version number of HeliOS.

3.3.2.5 productName `Byte_t SystemInfo_s::productName[0x6u]`

The product name of the operating system (always "HeliOS").

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

- HeliOS.h

3.4 TaskInfo_s Struct Reference

Data structure for information about a task.

Data Fields

- Base_t id
- Byte_t name [0x8u]
- TaskState_t state
- Ticks_t lastRunTime
- Ticks_t totalRunTime

3.4.1 Detailed Description

The TaskInfo_t structure is similar to xTaskRuntimeStats_t in that it contains runtime statistics for a task. However, TaskInfo_t also contains additional details about a task such as its name and state. The TaskInfo_t structure is returned by xTaskGetTaskInfo() and xTaskGetAllTaskInfo(). If only runtime statistics are needed, then TaskRun↩ TimeStats_t should be used because of its smaller memory footprint.

See also

xTaskInfo

xTaskGetTaskInfo()

xTaskGetAllTaskInfo()

CONFIG_TASK_NAME_BYTES

xMemFree()

3.4.2 Field Documentation

3.4.2.1 id `Base_t TaskInfo_s::id`

The ID of the task.

3.4.2.2 lastRunTime `Ticks_t TaskInfo_s::lastRunTime`

The duration in ticks of the task's last runtime.

3.4.2.3 name `Byte_t TaskInfo_s::name[0x8u]`

The name of the task which must be exactly CONFIG_TASK_NAME_BYTES bytes in length. Shorter task names must be padded.

3.4.2.4 state `TaskState_t TaskInfo_s::state`

The state the task is in which is one of four states specified in the TaskState_t enumerated data type.

3.4.2.5 totalRunTime `Ticks_t TaskInfo_s::totalRunTime`

The duration in ticks of the task's total runtime.

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

- HeliOS.h

3.5 TaskNotification_s Struct Reference

Data structure for a direct to task notification.

Data Fields

- Base_t notificationBytes
- Byte_t notificationValue [0x8u]

3.5.1 Detailed Description

The TaskNotification_t data structure is used by xTaskNotifyGive() and xTaskNotifyTake() to send and receive direct to task notifications. Direct to task notifications are part of the event-driven multitasking model. A direct to task notification may be received by event-driven and co-operative tasks alike. However, the benefit of direct to task notifications may only be realized by tasks scheduled as event-driven. In order to wait for a direct to task notification, the task must be in a "waiting" state which is set by xTaskWait().

See also

xTaskNotification
xMemFree()
xTaskNotifyGive()
xTaskNotifyTake()
xTaskWait()

3.5.2 Field Documentation

3.5.2.1 notificationBytes `Base_t TaskNotification_s::notificationBytes`

The length in bytes of the notification value which cannot exceed CONFIG_NOTIFICATION_VALUE_BYTES.

3.5.2.2 notificationValue `Byte_t TaskNotification_s::notificationValue[0x8u]`

The notification value whose length is specified by the notification bytes member.

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

- HeliOS.h

3.6 TaskRunTimeStats_s Struct Reference

Data structure for task runtime statistics.

Data Fields

- Base_t id
- Ticks_t lastRunTime
- Ticks_t totalRunTime

3.6.1 Detailed Description

The TaskRunTimeStats_t data structure is used by xTaskGetTaskRunTimeStats() and xTaskGetAllRuntimeStats() to obtain runtime statistics about a task.

See also

xTaskRunTimeStats

xTaskGetTaskRunTimeStats()

xTaskGetAllRunTimeStats()

xMemFree()

3.6.2 Field Documentation

3.6.2.1 id `Base_t TaskRunTimeStats_s::id`

The ID of the task.

3.6.2.2 lastRunTime `Ticks_t TaskRunTimeStats_s::lastRunTime`

The duration in ticks of the task's last runtime.

3.6.2.3 totalRunTime `Ticks_t TaskRunTimeStats_s::totalRunTime`

The duration in ticks of the task's total runtime.

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

- HeliOS.h

4 File Documentation

4.1 config.h File Reference

Kernel source for build configuration.

Macros

- #define CONFIG_ENABLE_ARDUINO_CPP_INTERFACE

  Define to enable the Arduino API C++ interface.
- #define CONFIG_ENABLE_SYSTEM_ASSERT

  Define to enable system assertions.
- #define CONFIG_SYSTEM_ASSERT_BEHAVIOR(f, l) __ArduinoAssert__(f, l)

  Define the system assertion behavior.
- #define CONFIG_MESSAGE_VALUE_BYTES 0x8u /* 8 */

  Define the size in bytes of the message queue message value.
- #define CONFIG_NOTIFICATION_VALUE_BYTES 0x8u /* 8 */

  Define the size in bytes of the direct to task notification value.
- #define CONFIG_TASK_NAME_BYTES 0x8u /* 8 */

  Define the size in bytes of the task name.
- #define CONFIG_MEMORY_REGION_SIZE_IN_BLOCKS 0x18u /* 24 */

  Define the number of memory blocks available in all memory regions.
- #define CONFIG_MEMORY_REGION_BLOCK_SIZE 0x20u /* 32 */

  Define the memory block size in bytes for all memory regions.
- #define CONFIG_QUEUE_MINIMUM_LIMIT 0x5u /* 5 */

  Define the minimum value for a message queue limit.
- #define CONFIG_STREAM_BUFFER_BYTES 0x20u /* 32 */

  Define the length of the stream buffer.
- #define CONFIG_TASK_WD_TIMER_ENABLE

  Enable task watchdog timers.
- #define CONFIG_DEVICE_NAME_BYTES 0x8u /* 8 */

  Define the length of a device driver name.

4.1.1 Detailed Description

Author

Manny Peterson manny@heliosproj.org

Version

0.4.0

Date

2023-03-19

Copyright

4.1.2 Macro Definition Documentation

4.1.2.1 CONFIG_DEVICE_NAME_BYTES `#define CONFIG_DEVICE_NAME_BYTES 0x8u /* 8 */`

Setting CONFIG_DEVICE_NAME_BYTES will define the length of a device driver name. The name of device drivers should be exactly this length. There really isn't a reason to change this and doing so may break existing device drivers. The default length is 8 bytes.

4.1.2.2 CONFIG_ENABLE_ARDUINO_CPP_INTERFACE `#define CONFIG_ENABLE_ARDUINO_CPP_INTERFACE`

Because HeliOS kernel is written in C, the Arduino API cannot be called directly from the kernel. For example, assertions are unable to be written to the serial bus in applications using the Arduino platform/tool-chain. The CONFIG_ENABLE_ARDUINO_CPP_INTERFACE builds the included arduino.cpp file to allow the kernel to call the Arduino API through wrapper functions such as **ArduinoAssert**(). The arduino.cpp file can be found in the /extras directory. It must be copied into the /src directory to be built.

4.1.2.3 CONFIG_ENABLE_SYSTEM_ASSERT `#define CONFIG_ENABLE_SYSTEM_ASSERT`

The CONFIG_ENABLE_SYSTEM_ASSERT setting allows the end-user to enable system assertions in HeliOS. Once enabled, the end-user must define CONFIG_SYSTEM_ASSERT_BEHAVIOR for there to be an effect. By default the CONFIG_ENABLE_SYSTEM_ASSERT setting is not defined.

See also

CONFIG_SYSTEM_ASSERT_BEHAVIOR

4.1.2.4 CONFIG_MEMORY_REGION_BLOCK_SIZE `#define CONFIG_MEMORY_REGION_BLOCK_SIZE 0x20u /* 32 */`

Setting CONFIG_MEMORY_REGION_BLOCK_SIZE allows the end-user to define the size of a memory region block in bytes. The memory region block size should be set to achieve the best possible utilization of the available memory. The CONFIG_MEMORY_REGION_BLOCK_SIZE setting effects both the heap and kernel memory regions. The default value is 32 bytes.

See also

xMemAlloc()

xMemFree()

CONFIG_MEMORY_REGION_SIZE_IN_BLOCKS

4.1.2.5 CONFIG_MEMORY_REGION_SIZE_IN_BLOCKS `#define CONFIG_MEMORY_REGION_SIZE_IN_↩ BLOCKS 0x18u /* 24 */`

The heap memory region is used by tasks. Whereas the kernel memory region is used solely by the kernel for kernel objects. The CONFIG_MEMORY_REGION_SIZE_IN_BLOCKS setting allows the end-user to define the size, in blocks, of all memory regions thus effecting both the heap and kernel memory regions. The size of a memory block is defined by the CONFIG_MEMORY_REGION_BLOCK_SIZE setting. The size of all memory regions needs to be adjusted to fit the memory requirements of the end-user's application. The default value is 24 blocks.

4.1.2.6 CONFIG_MESSAGE_VALUE_BYTES `#define CONFIG_MESSAGE_VALUE_BYTES 0x8u /* 8 */`

Setting the CONFIG_MESSAGE_VALUE_BYTES allows the end-user to define the size of the message queue message value. The larger the size of the message value, the greater impact there will be on system performance. The default size is 8 bytes.

See also

xQueueMessage

4.1.2.7 CONFIG_NOTIFICATION_VALUE_BYTES `#define CONFIG_NOTIFICATION_VALUE_BYTES 0x8u /* 8 */`

Setting the CONFIG_NOTIFICATION_VALUE_BYTES allows the end-user to define the size of the direct to task notification value. The larger the size of the notification value, the greater impact there will be on system performance. The default size is 8 bytes.

See also

xTaskNotification

4.1.2.8 CONFIG_QUEUE_MINIMUM_LIMIT `#define CONFIG_QUEUE_MINIMUM_LIMIT 0x5u /* 5 */`

Setting the CONFIG_QUEUE_MINIMUM_LIMIT allows the end-user to define the MINIMUM length limit a message queue can be created with xQueueCreate(). When a message queue length equals its limit, the message queue will be considered full and return true when xQueueIsQueueFull() is called. A full queue will also not accept messages from xQueueSend(). The default value is 5.

See also

xQueueIsQueueFull()

xQueueSend()

xQueueCreate()

4.1.2.9 CONFIG_STREAM_BUFFER_BYTES `#define CONFIG_STREAM_BUFFER_BYTES 0x20u /* 32 */`

Setting CONFIG_STREAM_BUFFER_BYTES will define the length of stream buffers created by xStreamCreate(). When the length of the stream buffer reaches this value, it is considered full and can no longer be written to by calling xStreamSend(). The default value is 32.

4.1.2.10 CONFIG_SYSTEM_ASSERT_BEHAVIOR `#define CONFIG_SYSTEM_ASSERT_BEHAVIOR( f, l ) __ArduinoAssert__(f, l)`

The CONFIG_SYSTEM_ASSERT_BEHAVIOR setting allows the end-user to specify the behavior (code) of the assertion which is called when CONFIG_ENABLE_SYSTEM_ASSERT is defined. Typically some sort of output is generated over a serial or other interface. By default the CONFIG_SYSTEM_ASSERT_BEHAVIOR is not defined.

Note

In order to use the **ArduinoAssert**() functionality, the CONFIG_ENABLE_ARDUINO_CPP_INTERFACE setting must be enabled.

See also

CONFIG_ENABLE_SYSTEM_ASSERT

CONFIG_ENABLE_ARDUINO_CPP_INTERFACE

```
#define CONFIG_SYSTEM_ASSERT_BEHAVIOR(f, l) __ArduinoAssert__( f , l )
```

4.1.2.11 CONFIG_TASK_NAME_BYTES `#define CONFIG_TASK_NAME_BYTES 0x8u /* 8 */`

Setting the CONFIG_TASK_NAME_BYTES allows the end-user to define the size of the task name. The larger the size of the task name, the greater impact there will be on system performance. The default size is 8 bytes.

See also

xTaskInfo

4.1.2.12 CONFIG_TASK_WD_TIMER_ENABLE `#define CONFIG_TASK_WD_TIMER_ENABLE`

Defining CONFIG_TASK_WD_TIMER_ENABLE will enable the task watchdog timer feature. The default is enabled.

4.2 HeliOS.h File Reference

Kernel source for user application header.

Data Structures

- struct TaskNotification_s

  Data structure for a direct to task notification.
- struct TaskRunTimeStats_s

  Data structure for task runtime statistics.
- struct MemoryRegionStats_s

  Data structure for memory region statistics.
- struct TaskInfo_s

  Data structure for information about a task.
- struct QueueMessage_s

  Data structure for a queue message.
- struct SystemInfo_s

  Data structure for information about the HeliOS system.

Typedefs

- typedef enum TaskState_e TaskState_t

  Enumerated type for task states.
- typedef TaskState_t xTaskState

  Enumerated type for task states.
- typedef enum SchedulerState_e SchedulerState_t

  Enumerated type for scheduler state.
- typedef SchedulerState_t xSchedulerState

  Enumerated type for scheduler state.
- typedef enum Return_e Return_t

  Enumerated type for syscall return type.
- typedef Return_t xReturn

  Enumerated type for syscall return type.
- typedef void TaskParm_t

  Data type for the task paramater.
- typedef TaskParm_t * xTaskParm

  Data type for the task paramater.
- typedef uint8_t Base_t

  Data type for the base type.
- typedef Base_t xBase

  Data type for the base type.
- typedef uint8_t Byte_t

  Data type for an 8-bit wide byte.
- typedef Byte_t xByte

  Data type for an 8-bit wide byte.
- typedef void Addr_t

  Data type for a pointer to a memory address.
- typedef Addr_t * xAddr

  Data type for a pointer to a memory address.
- typedef size_t Size_t

  Data type for the storage requirements of an object in memory.
- typedef Size_t xSize

  Data type for the storage requirements of an object in memory.
- typedef uint16_t HalfWord_t

Data type for a 16-bit half word.

- typedef HalfWord_t xHalfWord

  Data type for a 16-bit half word.

- typedef uint32_t Word_t

  Data type for a 32-bit word.

- typedef Word_t xWord

  Data type for a 32-bit word.

- typedef uint32_t Ticks_t

  Data type for system ticks.

- typedef Ticks_t xTicks

  Data type for system ticks.

- typedef void Task_t

  Data type for a task.

- typedef Task_t * xTask

  Data type for a task.

- typedef void Timer_t

  Data type for a timer.

- typedef Timer_t * xTimer

  Data type for a timer.

- typedef void Queue_t

  Data type for a queue.

- typedef Queue_t * xQueue

  Data type for a queue.

- typedef void StreamBuffer_t

  Data type for a stream buffer.

- typedef StreamBuffer_t * xStreamBuffer

  Data type for a stream buffer.

- typedef struct TaskNotification_s TaskNotification_t

  Data structure for a direct to task notification.

- typedef TaskNotification_t * xTaskNotification

  Data structure for a direct to task notification.

- typedef struct TaskRunTimeStats_s TaskRunTimeStats_t

  Data structure for task runtime statistics.

- typedef TaskRunTimeStats_t * xTaskRunTimeStats

  Data structure for task runtime statistics.

- typedef struct MemoryRegionStats_s MemoryRegionStats_t

  Data structure for memory region statistics.

- typedef MemoryRegionStats_t * xMemoryRegionStats

  Data structure for memory region statistics.

- typedef struct TaskInfo_s TaskInfo_t

  Data structure for information about a task.

- typedef TaskInfo_t xTaskInfo

  Data structure for information about a task.

- typedef struct QueueMessage_s QueueMessage_t

  Data structure for a queue message.

- typedef QueueMessage_t * xQueueMessage

  Data structure for a queue message.

- typedef struct SystemInfo_s SystemInfo_t

  Data structure for information about the HeliOS system.

- typedef SystemInfo_t * xSystemInfo

  Data structure for information about the HeliOS system.

Enumerations

- enum TaskState_e { TaskStateSuspended , TaskStateRunning , TaskStateWaiting }
  Enumerated type for task states.
- enum SchedulerState_e { SchedulerStateSuspended , SchedulerStateRunning }
  Enumerated type for scheduler state.
- enum Return_e { ReturnOK , ReturnError }
  Enumerated type for syscall return type.

Functions

- xReturn xDeviceRegisterDevice (xReturn(∗device_self_register_)())
  Syscall to register a device driver with the kernel.
- xReturn xDeviceIsAvailable (const xHalfWord uid_, xBase ∗res_)
  Syscall to query the device driver about the availability of a device.
- xReturn xDeviceSimpleWrite (const xHalfWord uid_, xByte data_)
  Syscall to write a byte of data to the device.
- xReturn xDeviceWrite (const xHalfWord uid_, xSize ∗size_, xAddr data_)
  Syscall to write multiple bytes of data to a device.
- xReturn xDeviceSimpleRead (const xHalfWord uid_, xByte ∗data_)
  Syscall to read a byte of data from the device.
- xReturn xDeviceRead (const xHalfWord uid_, xSize ∗size_, xAddr ∗data_)
  Syscall to read multiple bytes from a device.
- xReturn xDeviceInitDevice (const xHalfWord uid_)
  Syscall to initialize a device.
- xReturn xDeviceConfigDevice (const xHalfWord uid_, xSize ∗size_, xAddr config_)
  Syscall to configure a device.
- xReturn xMemAlloc (volatile xAddr ∗addr_, const xSize size_)
  Syscall to request memory from the heap.
- xReturn xMemFree (const volatile xAddr addr_)
  Syscall to free heap memory allocated by xMemAlloc()
- xReturn xMemGetUsed (xSize ∗size_)
  Syscall to obtain the amount of in-use heap memory.
- xReturn xMemGetSize (const volatile xAddr addr_, xSize ∗size_)
  Syscall to obtain the amount of heap memory allocated at a specific address.
- xReturn xMemGetHeapStats (xMemoryRegionStats ∗stats_)
  Syscall to get memory statistics on the heap memory region.
- xReturn xMemGetKernelStats (xMemoryRegionStats ∗stats_)
  Syscall to get memory statistics on the kernel memory region.
- xReturn xQueueCreate (xQueue ∗queue_, const xBase limit_)
  Syscall to create a message queue.
- xReturn xQueueDelete (xQueue queue_)
  Syscall to delete a message queue.
- xReturn xQueueGetLength (const xQueue queue_, xBase ∗res_)
  Syscall to get the length of a message queue.
- xReturn xQueueIsQueueEmpty (const xQueue queue_, xBase ∗res_)
  Syscall to inquire as to whether a message queue is empty.
- xReturn xQueueIsQueueFull (const xQueue queue_, xBase ∗res_)
  Syscall to inquire as to whether a message queue is full.
- xReturn xQueueMessagesWaiting (const xQueue queue_, xBase ∗res_)

Syscall to inquire as to whether a message queue has one or more messages waiting.

- xReturn xQueueSend (xQueue queue_, const xBase bytes_, const xByte *value_)

  Syscall to send a message to a message queue.
- xReturn xQueuePeek (const xQueue queue_, xQueueMessage *message_)

  Syscall to retrieve a message from a message queue without dropping the message.
- xReturn xQueueDropMessage (xQueue queue_)

  Syscall to drop a message from a message queue without retrieving the message.
- xReturn xQueueReceive (xQueue queue_, xQueueMessage *message_)

  Syscall to retrieve and drop the next message from a message queue.
- xReturn xQueueLockQueue (xQueue queue_)

  Syscall to lock a message queue.
- xReturn xQueueUnLockQueue (xQueue queue_)

  Syscall to unlock a message queue.
- xReturn xStreamCreate (xStreamBuffer *stream_)

  Syscall to create a stream buffer.
- xReturn xStreamDelete (const xStreamBuffer stream_)

  Syscall to delete a stream buffer.
- xReturn xStreamSend (xStreamBuffer stream_, const xByte byte_)

  Syscall to send a byte to a stream buffer.
- xReturn xStreamReceive (const xStreamBuffer stream_, xHalfWord *bytes_, xByte **data_)

  Syscall to retrieve all waiting bytes from a stream buffer.
- xReturn xStreamBytesAvailable (const xStreamBuffer stream_, xHalfWord *bytes_)

  Syscall to inquire about the number of bytes waiting in a stream buffer.
- xReturn xStreamReset (const xStreamBuffer stream_)

  Syscall to reset a stream buffer.
- xReturn xStreamIsEmpty (const xStreamBuffer stream_, xBase *res_)

  Syscall to inquire as to whether a stream buffer is empty.
- xReturn xStreamIsFull (const xStreamBuffer stream_, xBase *res_)

  Syscall to inquire as to whether a stream buffer is full.
- xReturn xSystemAssert (const char *file_, const int line_)

  Syscall to to raise a system assert.
- xReturn xSystemInit (void)

  Syscall to bootstrap HeliOS.
- xReturn xSystemHalt (void)

  Syscall to halt HeliOS.
- xReturn xSystemGetSystemInfo (xSystemInfo *info_)

  Syscall to inquire about the system.
- xReturn xTaskCreate (xTask *task_, const xByte *name_, void(*callback_)(xTask task_, xTaskParm parm_), xTaskParm taskParameter_)

  Syscall to create a new task.
- xReturn xTaskDelete (const xTask task_)

  Syscall to delete a task.
- xReturn xTaskGetHandleByName (xTask *task_, const xByte *name_)

  Syscall to get the task handle by name.
- xReturn xTaskGetHandleById (xTask *task_, const xBase id_)

  Syscall to get the task handle by task id.
- xReturn xTaskGetAllRunTimeStats (xTaskRunTimeStats *stats_, xBase *tasks_)

  Syscall to get obtain the runtime statistics of all tasks.
- xReturn xTaskGetTaskRunTimeStats (const xTask task_, xTaskRunTimeStats *stats_)

  Syscall to get the runtime statistics for a single task.
- xReturn xTaskGetNumberOfTasks (xBase *tasks_)

Syscall to get the number of tasks.

- xReturn xTaskGetTaskInfo (const xTask task_, xTaskInfo *info_)

  Syscall to get info about a task.
- xReturn xTaskGetAllTaskInfo (xTaskInfo *info_, xBase *tasks_)

  Syscall to get info about all tasks.
- xReturn xTaskGetTaskState (const xTask task_, xTaskState *state_)

  Syscall to get the state of a task.
- xReturn xTaskGetName (const xTask task_, xByte **name_)

  Syscall to get the name of a task.
- xReturn xTaskGetId (const xTask task_, xBase *id_)

  Syscall to get the task id of a task.
- xReturn xTaskNotifyStateClear (xTask task_)

  Syscall to clear a waiting direct-to-task notification.
- xReturn xTaskNotificationIsWaiting (const xTask task_, xBase *res_)

  Syscall to inquire as to whether a direct-to-task notification is waiting.
- xReturn xTaskNotifyGive (xTask task_, const xBase bytes_, const xByte *value_)

  Syscall to give (i.e., send) a task a direct-to-task notification.
- xReturn xTaskNotifyTake (xTask task_, xTaskNotification *notification_)

  Syscall to take (i.e. receive) a waiting direct-to-task notification.
- xReturn xTaskResume (xTask task_)

  Syscall to place a task in the "running" state.
- xReturn xTaskSuspend (xTask task_)

  Syscall to place a task in the "suspended" state.
- xReturn xTaskWait (xTask task_)

  Syscall to place a task in the "waiting" state.
- xReturn xTaskChangePeriod (xTask task_, const xTicks period_)

  Syscall to change the interval period of a task timer.
- xReturn xTaskChangeWDPeriod (xTask task_, const xTicks period_)

  Syscall to change the task watchdog timer period.
- xReturn xTaskGetPeriod (const xTask task_, xTicks *period_)

  Syscall to obtain the task timer period.
- xReturn xTaskResetTimer (xTask task_)

  Syscall to set the task timer elapsed time to zero.
- xReturn xTaskStartScheduler (void)

  Syscall to start the HeliOS scheduler.
- xReturn xTaskResumeAll (void)

  Syscall to set the scheduler state to running.
- xReturn xTaskSuspendAll (void)

  Syscall to set the scheduler state to suspended.
- xReturn xTaskGetSchedulerState (xSchedulerState *state_)

  Syscall to get the scheduler state.
- xReturn xTaskGetWDPeriod (const xTask task_, xTicks *period_)

  Syscall to get the task watchdog timer period.
- xReturn xTimerCreate (xTimer *timer_, const xTicks period_)

  Syscall to create an application timer.
- xReturn xTimerDelete (const xTimer timer_)

  Syscall to delete an application timer.
- xReturn xTimerChangePeriod (xTimer timer_, const xTicks period_)

  Syscall to change the period on an application timer.
- xReturn xTimerGetPeriod (const xTimer timer_, xTicks *period_)

  Syscall to get the current period for an application timer.

- xReturn xTimerIsTimerActive (const xTimer timer_, xBase *res_)

  Syscall to inquire as to whether an application timer is active.
- xReturn xTimerHasTimerExpired (const xTimer timer_, xBase *res_)

  Syscall to inquire as to whether an application timer has expired.
- xReturn xTimerReset (xTimer timer_)

  Syscall to reset an application timer.
- xReturn xTimerStart (xTimer timer_)

  Syscall to place an application timer in the running state.
- xReturn xTimerStop (xTimer timer_)

  Syscall to place an application timer in the suspended state.

4.2.1 Detailed Description

Author

Manny Peterson manny@heliosproj.org

Version

0.4.0

Date

2023-03-19

Copyright

HeliOS Embedded Operating System Copyright (C) 2020-2023 HeliOS Project license@heliosproj.org

SPDX-License-Identifier: GPL-2.0-or-later

4.2.2 Typedef Documentation

4.2.2.1 Addr_t `typedef void Addr_t`

The Addr_t type is a pointer of type void and is used to pass addresses between the end-user application and syscalls. It is not necessary to use the Addr_t type within the end-user application as long as the type is not used to interact with the kernel through syscalls

See also

xAddr

4.2.2.2 Base_t `typedef uint8_t Base_t`

The Base_t type is a simple data type often used as an argument or result type for syscalls when the value is known not to exceed its 8-bit width and no data structure requirements exist. There are no guarantees the Base_t will always be 8-bits wide. If an 8-bit data type is needed that is guaranteed to remain 8-bits wide, the Byte_t data type should be used.

See also

xBase

Byte_t

4.2.2.3 Byte_t `typedef uint8_t Byte_t`

The Byte_t type is an 8-bit wide data type and is guaranteed to always be 8-bits wide.

See also

xByte

4.2.2.4 HalfWord_t `typedef uint16_t HalfWord_t`

The HalfWord_t type is a 16-bit wide data type and is guaranteed to always be 16-bits wide.

See also

xHalfWord

4.2.2.5 MemoryRegionStats_t `typedef struct MemoryRegionStats_s MemoryRegionStats_t`

The MemoryRegionStats_t data structure is used by xMemGetHeapStats() and xMemGetKernelStats() to obtain statistics about either memory region.

See also

xMemoryRegionStats

xMemGetHeapStats()

xMemGetKernelStats()

xMemFree()

4.2.2.6 Queue_t `typedef void Queue_t`

The Queue_t data type is used as a queue The queue is created when xQueueCreate() is called. For more information about queues, see xQueueCreate().

See also

xQueue
xQueueCreate()
xQueueDelete()

4.2.2.7 QueueMessage_t `typedef struct QueueMessage_s QueueMessage_t`

The QueueMessage_t stucture is used to store a queue message and is returned by xQueueReceive() and xQueuePeek().

See also

xQueueMessage
xQueueReceive()
xQueuePeek()
CONFIG_MESSAGE_VALUE_BYTES
xMemFree()

4.2.2.8 Return_t `typedef enum Return_e Return_t`

All HeliOS syscalls return the Return_t type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGet↩ Used(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

See also

OK()
ERROR()
xReturn

4.2.2.9 SchedulerState_t `typedef enum SchedulerState_e SchedulerState_t`

The scheduler can be in one of three possible states as defined by the SchedulerState_t enumerated data type. The state the scheduler is in is changed by calling xTaskSuspendAll() and xTaskResumeAll(). The state the scheduler is in can be obtained by calling xTaskGetSchedulerState().

See also

xSchedulerState
xTaskSuspendAll()
xTaskResumeAll()
xTaskGetSchedulerState()
xTaskStartScheduler()

4.2.2.10 Size_t `typedef size_t Size_t`

The Size_t type is used for the storage requirements of an object in memory and is always represented in bytes.

See also

xSize

4.2.2.11 StreamBuffer_t `typedef void StreamBuffer_t`

The StreamBuffer_t data type is used as a stream buffer. The stream buffer is created when xStreamCreate() is called. For more information about stream buffers, see xStreamCreate(). Stream_t should be declared as xStream.

See also

xStream

xStreamCreate()

xStreamDelete()

4.2.2.12 SystemInfo_t `typedef struct SystemInfo_s SystemInfo_t`

The SystemInfo_t data structure is used to store information about the HeliOS system and is returned by xSystemGetSystemInfo().

See also

xSystemInfo

xSystemGetSystemInfo()

OS_PRODUCT_NAME_SIZE

xMemFree()

4.2.2.13 Task_t `typedef void Task_t`

The Task_t data type is used as a task. The task is created when xTaskCreate() is called. For more information about tasks, see xTaskCreate().

See also

xTask

xTaskCreate()

xTaskDelete()

4.2.2.14 TaskInfo_t `typedef struct TaskInfo_s TaskInfo_t`

The TaskInfo_t structure is similar to xTaskRuntimeStats_t in that it contains runtime statistics for a task. However, TaskInfo_t also contains additional details about a task such as its name and state. The TaskInfo_t structure is returned by xTaskGetTaskInfo() and xTaskGetAllTaskInfo(). If only runtime statistics are needed, then TaskRun↩ TimeStats_t should be used because of its smaller memory footprint.

See also

- xTaskInfo
- xTaskGetTaskInfo()
- xTaskGetAllTaskInfo()
- CONFIG_TASK_NAME_BYTES
- xMemFree()

4.2.2.15 TaskNotification_t `typedef struct TaskNotification_s TaskNotification_t`

The TaskNotification_t data structure is used by xTaskNotifyGive() and xTaskNotifyTake() to send and receive direct to task notifications. Direct to task notifications are part of the event-driven multitasking model. A direct to task notification may be received by event-driven and co-operative tasks alike. However, the benefit of direct to task notifications may only be realized by tasks scheduled as event-driven. In order to wait for a direct to task notification, the task must be in a "waiting" state which is set by xTaskWait().

See also

- xTaskNotification
- xMemFree()
- xTaskNotifyGive()
- xTaskNotifyTake()
- xTaskWait()

4.2.2.16 TaskParm_t `typedef void TaskParm_t`

The TaskParm_t type is used to pass a paramater to a task at the time of task creation using xTaskCreate(). A task paramater is a pointer of type void and can point to any number of types, arrays and/or data structures that will be passed to the task. It is up to the end-user to manage, allocate and free the memory related to these objects using xMemAlloc() and xMemFree().

See also

- xTaskParm
- xTaskCreate()
- xMemAlloc()
- xMemFree()

4.2.2.17 TaskRunTimeStats_t `typedef struct TaskRunTimeStats_s TaskRunTimeStats_t`

The TaskRunTimeStats_t data structure is used by xTaskGetTaskRunTimeStats() and xTaskGetAllRuntimeStats() to obtain runtime statistics about a task.

See also

xTaskRunTimeStats

xTaskGetTaskRunTimeStats()

xTaskGetAllRunTimeStats()

xMemFree()

4.2.2.18 TaskState_t `typedef enum TaskState_e TaskState_t`

A task can be in one of four possible states as defined by the TaskState_t enumerated data type. The state a task is in is changed by calling xTaskResume(), xTaskSuspend() or xTaskWait(). The HeliOS scheduler will only schedule, for execution, tasks in either the TaskStateRunning or TaskStateWaiting state.

See also

xTaskState

xTaskResume()

xTaskSuspend()

xTaskWait()

xTaskGetTaskState()

4.2.2.19 Ticks_t `typedef uint32_t Ticks_t`

The Ticks_t type is used to store ticks from the system clock. Ticks is not bound to any one unit of measure for time though most systems are configured for millisecond resolution, milliseconds is not guaranteed and is dependent on the system clock frequency and prescaler.

See also

xTicks

4.2.2.20 Timer_t `typedef void Timer_t`

The Timer_t data type is used as a timer. The timer is created when xTimerCreate() is called. For more information about timers, see xTimerCreate().

See also

xTimer

xTimerCreate()

xTimerDelete()

4.2.2.21 Word_t `typedef uint32_t Word_t`

The Word_t type is a 32-bit wide data type and is guaranteed to always be 32-bits wide.

See also

xWord

4.2.2.22 xAddr `typedef Addr_t* xAddr`

See also

Addr_t

4.2.2.23 xBase `typedef Base_t xBase`

See also

Base_t

4.2.2.24 xByte `typedef Byte_t xByte`

See also

Byte_t

4.2.2.25 xHalfWord `typedef HalfWord_t xHalfWord`

See also

HalfWord_t

4.2.2.26 xQueue `typedef Queue_t* xQueue`

See also

Queue_t

4.2.2.27 xReturn `typedef Return_t xReturn`

See also

Return_t

4.2.2.28 xSchedulerState `typedef SchedulerState_t xSchedulerState`

See also

SchedulerState_t

4.2.2.29 xSize `typedef Size_t xSize`

See also

Size_t

4.2.2.30 xStreamBuffer `typedef StreamBuffer_t* xStreamBuffer`

See also

StreamBuffer_t

4.2.2.31 xTask `typedef Task_t* xTask`

See also

Task_t

4.2.2.32 xTaskNotification `typedef TaskNotification_t* xTaskNotification`

See also

TaskNotification_t

4.2.2.33 xTaskParm `typedef TaskParm_t* xTaskParm`

See also

TaskParm_t

4.2.2.34 xTaskState `typedef TaskState_t xTaskState`

See also

TaskState_t

4.2.2.35 xTicks `typedef Ticks_t xTicks`

See also

Ticks_t

4.2.2.36 xTimer `typedef Timer_t* xTimer`

See also

Timer_t

4.2.2.37 xWord `typedef Word_t xWord`

See also

Word_t

4.2.3 Enumeration Type Documentation

4.2.3.1 Return_e `enum Return_e`

All HeliOS syscalls return the Return_t type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGet↩ Used(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

See also

OK()

ERROR()

xReturn

Enumerator

ReturnOK	Return value if the syscall was successful.
ReturnError	Return value if the syscall failed.

4.2.3.2 SchedulerState_e `enum SchedulerState_e`

The scheduler can be in one of three possible states as defined by the SchedulerState_t enumerated data type. The state the scheduler is in is changed by calling xTaskSuspendAll() and xTaskResumeAll(). The state the scheduler is in can be obtained by calling xTaskGetSchedulerState().

See also

- xSchedulerState
- xTaskSuspendAll()
- xTaskResumeAll()
- xTaskGetSchedulerState()
- xTaskStartScheduler()

Enumerator

SchedulerStateSuspended	State the scheduler is in after calling xTaskSuspendAll(). TaskStartScheduler() will stop scheduling tasks for execution and relinquish control when xTaskSuspendAll() is called.
SchedulerStateRunning	State the scheduler is in after calling xTaskResumeAll(). xTaskStartScheduler() will continue to schedule tasks for execution until xTaskSuspendAll() is called.

4.2.3.3 TaskState_e `enum TaskState_e`

A task can be in one of four possible states as defined by the TaskState_t enumerated data type. The state a task is in is changed by calling xTaskResume(), xTaskSuspend() or xTaskWait(). The HeliOS scheduler will only schedule, for execution, tasks in either the TaskStateRunning or TaskStateWaiting state.

See also

- xTaskState
- xTaskResume()
- xTaskSuspend()
- xTaskWait()
- xTaskGetTaskState()

Enumerator

TaskStateSuspended	State a task is in after it is created OR after calling xTaskSuspend(). Tasks in the TaskStateSuspended state will not be scheduled for execution by the scheduler.
TaskStateRunning	State a task is in after calling xTaskResume(). Tasks in the TaskStateRunning state will be scheduled for execution by the scheduler.
TaskStateWaiting	State a task is in after calling xTaskWait(). Tasks in the TaskStateWaiting state will be scheduled for execution by the scheduler only when a task event has occurred.

4.2.4 Function Documentation

4.2.4.1 xDeviceConfigDevice()

```
xReturn xDeviceConfigDevice (
        const xHalfWord uid_,
        xSize * size_,
        xAddr config_ )
```

The xDeviceConfigDevice() will call the device driver's DEVICENAME_config() function to configure the device. The syscall is bi-directional (i.e., it will write configuration data to the device and read the same from the device before returning). The purpose of the bi-directional functionality is to allow the device's configuration to be set and queried using one syscall. The structure of the configuration data is left to the device driver's author. What is required is that the configuration data memory is allocated using xMemAlloc() and that the "size_" parameter is set to the size (i.e., amount) of the configuration data (e.g., sizeof(MyDeviceDriverConfig)) in bytes.

See also

xReturn

xMemAlloc()

xMemFree()

Parameters

*uid_*	The unique identifier ("UID") of the device driver to be operated on.
*size_*	The size (i.e., amount) of configuration data to bw written and read to and from the device, in bytes.
*config_*	The configuration data. The configuration data must have been allocated by xMemAlloc().

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.2 xDeviceInitDevice()

```
xReturn xDeviceInitDevice (
        const xHalfWord uid_ )
```

The xDeviceInitDevice() syscall will call the device driver's DRIVERNAME_init() function to bootstrap the device. For example, setting memory mapped registers to starting values or setting the device driver's state and mode. This syscall is optional and is dependent on the specifics of the device driver's implementation by its author.

See also

xReturn

Parameters

*uid_*	The unique identifier ("UID") of the device driver to be operated on.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.3 xDeviceIsAvailable()

```
xReturn xDeviceIsAvailable (
        const xHalfWord uid_,
        xBase * res_ )
```

The xDeviceIsAvailable() syscall queries the device driver about the availability of a device. Generally "available" means the that the device is available for read and/or write operations though the meaning is implementation specific and left up to the device driver's author.

See also

xReturn

Parameters

*uid_*	The unique identifier ("UID") of the device driver to be operated on.
*res_*	The result of the inquiry; here, taken to mean the availability of the device.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or

invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.4 xDeviceRead()

```
xReturn xDeviceRead (
    const xHalfWord uid_,
    xSize * size_,
    xAddr * data_ )
```

The xDeviceRead() syscall will read multiple bytes of data from a device into a data buffer. The data buffer must be freed by xMemFree(). Whether the data is read from the device is dependent on the device driver mode, state and implementation of these features by the device driver's author.

See also

xReturn

xMemFree()

Parameters

*uid_*	The unique identifier ("UID") of the device driver to be operated on.
*size_*	The number of bytes read from the device and contained in the data buffer.
*data_*	The data buffer containing the data read from the device which must be freed by xMemFree().

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.5 xDeviceRegisterDevice()

```
xReturn xDeviceRegisterDevice (
    xReturn(*)() device_self_register_ )
```

The xDeviceRegisterDevice() syscall is a component of the HeliOS device driver model which registers a device driver with the HeliOS kernel. This syscall must be made before a device driver can be called by xDeviceRead(), xDeviceWrite(), etc. Once a device is registered, it cannot be un-registered - it can only be placed in a suspended state which is done by calling xDeviceConfigDevice(). However, as with most aspects of the HeliOS device driver model , it is important to note that the implementation of and support for device state and mode is up to the device driver's author.

Note

A device driver's unique identifier ("UID") must be a globally unique identifier. No two device drivers in the same application can share the same UID. This is best achieved by ensuring the device driver author selects a UID for his device driver that is not in use by other device drivers. A device driver template and device drivers can be found in /drivers.

See also

CONFIG_DEVICE_NAME_BYTES

xReturn

Parameters

*device_self_↩ register_*	The device driver's self registration function, DRIVERNAME_self_register().

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.6 xDeviceSimpleRead()

```
xReturn xDeviceSimpleRead (
    const xHalfWord uid_,
    xByte * data_ )
```

The xDeviceSimpleRead() syscall will read a byte of data from a device. Whether the data is read from the device is dependent on the device driver mode, state and implementation of these features by the device driver's author.

See also

xReturn

Parameters

uid↩ _	The unique identifier ("UID") of the device driver to be operated on.
data↩ _	The byte of data read from the device.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or

invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.7 xDeviceSimpleWrite()

```
xReturn xDeviceSimpleWrite (
    const xHalfWord uid_,
    xByte data_ )
```

The xDeviceSimpleWrite() syscall will write a byte of data to a device. Whether the data is written to the device is dependent on the device driver mode, state and implementation of these features by the device driver's author.

See also

xReturn

Parameters

*uid_*	The unique identifier ("UID") of the device driver to be operated on.
*data_*	A byte of data to be written to the device.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.8 xDeviceWrite()

```
xReturn xDeviceWrite (
    const xHalfWord uid_,
    xSize * size_,
    xAddr data_ )
```

The xDeviceWrite() syscall will write multiple bytes of data contained in a data buffer to a device. The data buffer must have been allocated by xMemAlloc(). Whether the data is written to the device is dependent on the device driver mode, state and implementation of these features by the device driver's author.

See also

xReturn

xMemAlloc()

xMemFree()

Parameters

*uid_*	The unique identifier ("UID") of the device driver to be operated on.
*size_*	The size of the data buffer, in bytes.
*data_*	The data buffer containing the data to be written to the device. The data buffer must have been allocated by xMemAlloc().

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.9 xMemAlloc()

```
xReturn xMemAlloc (
    volatile xAddr * addr_,
    const xSize size_ )
```

The xMemAlloc() syscall allocates heap memory for user's application. The amount of available heap memory is dependent on the CONFIG_MEMORY_REGION_SIZE_IN_BLOCKS and CONFIG_MEMORY_REGION_BLOCK_SIZE settings. Similar to libc calloc(), xMemAlloc() clears (i.e., zeros out) the allocated memory it allocates. Because the address of the newly allocated heap memory is handed back through the "addr_" argument, the argument must be cast to "volatile xAddr *" to avoid compiler warnings.

See also

xReturn

CONFIG_MEMORY_REGION_SIZE_IN_BLOCKS

CONFIG_MEMORY_REGION_BLOCK_SIZE

xMemFree()

Parameters

*addr_*	The address of the allocated memory. For example, if heap memory for a structure called mystruct (MyStruct *) needs to be allocated, the call to xMemAlloc() would be written as follows if(OK(xMemAlloc((volatile xAddr *) &mystruct, sizeof(MyStruct)))) {}.
*size_*	The amount of heap memory, in bytes, being requested.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or

invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.10 xMemFree() `xReturn xMemFree ( const volatile xAddr addr_ )`

The xMemFree() syscall frees (i.e., de-allocates) heap memory allocated by xMemAlloc(). xMemFree() is also used to free heap memory allocated by syscalls including xTaskGetAllRunTimeStats().

See also

xReturn

xMemAlloc()

Parameters

*addr_*	The address of the allocated memory to be freed.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.11 xMemGetHeapStats() `xReturn xMemGetHeapStats ( xMemoryRegionStats * stats_ )`

The xMemGetHeapStats() syscall is used to obtain detailed statistics about the heap memory region which can be used by the application to monitor memory utilization.

See also

xReturn

xMemoryRegionStats

xMemFree()

Parameters

*stats_*	The memory region statistics. The memory region statistics must be freed by xMemFree().

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.12 xMemGetKernelStats()

```
xReturn xMemGetKernelStats (
    xMemoryRegionStats * stats_ )
```

The xMemGetKernelStats() syscall is used to obtain detailed statistics about the kernel memory region which can be used by the application to monitor memory utilization.

See also

xReturn

xMemoryRegionStats

xMemFree()

Parameters

*stats_*	The memory region statistics. The memory region statistics must be freed by xMemFree().

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.13 xMemGetSize()

```
xReturn xMemGetSize (
    const volatile xAddr addr_,
    xSize * size_ )
```

The xMemGetSize() syscall can be used to obtain the amount, in bytes, of heap memory allocated at a specific address. The address must be the address obtained from xMemAlloc().

See also

xReturn

Parameters

*addr_*	The address of the heap memory for which the size (i.e., amount) allocated, in bytes, is being sought.
*size_*	The size (i.e., amount), in bytes, of heap memory allocated to the address.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.14 xMemGetUsed()

```
xReturn xMemGetUsed (
        xSize * size_ )
```

The xMemGetUsed() syscall will update the "size_" argument with the amount, in bytes, of in-use heap memory. If more memory statistics are needed, xMemGetHeapStats() provides a more complete picture of the heap memory region.

See also

xReturn

xMemGetHeapStats()

Parameters

*size_*	The size (i.e., amount), in bytes, of in-use heap memory.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.15 xQueueCreate()

```
xReturn xQueueCreate (
        xQueue * queue_,
        const xBase limit_ )
```

The xQueueCreate() syscall will create a new message queue for inter-task communication.

See also

xReturn

xQueue

CONFIG_QUEUE_MINIMUM_LIMIT

xQueueDelete()

Parameters

*queue_*	The message queue to be operated on.
*limit_*	The message limit for the queue. When this value is reached, the message queue is considered to be full. The minimume message limit is configured using the CONFIG_QUEUE_MINIMUM_LIMIT (default is 5) setting.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.16 xQueueDelete()

```
xReturn xQueueDelete (
        xQueue queue_ )
```

The xQueueDelete() syscall will delete a message queue used for inter-task communication.

See also

xReturn

xQueue

xQueueCreate()

Parameters

*queue_*	The message queue to be operated on.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.17 xQueueDropMessage()

```
xReturn xQueueDropMessage (
        xQueue queue_ )
```

The xQueueDropMessage() syscall is used to drop the next message from a message queue without retrieving the message.

See also

xReturn

xQueue

Parameters

*queue_*	The message queue to be operated on.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.18 xQueueGetLength()

```
xReturn xQueueGetLength (
        const xQueue queue_,
        xBase * res_ )
```

The xQueueGetLength() syscall is used to inquire about the length (i.e., the number of messages) of a message queue.

See also

xReturn

xQueue

Parameters

*queue_*	The message queue to be operated on.
*res_*	The result of the inquiry; taken here to mean the number of messages a message queue contains.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.19 xQueueIsQueueEmpty()

```
xReturn xQueueIsQueueEmpty (
    const xQueue queue_,
    xBase * res_ )
```

The xQueueIsQueueEmpty() syscall is used to inquire as to whether a message queue is empty. A message queue is considered empty if the length (i.e., number of messages) of a queue is zero.

See also

xReturn

xQueue

Parameters

*queue_*	The message queue to be operated on.
*res_*	The result of the inquiry; taken here to mean "true" if the queue is empty, "false" if it contains one or more messages.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.20 xQueueIsQueueFull() `xReturn xQueueIsQueueFull (`

```
const xQueue queue_,
xBase * res_ )
```

The xQueueIsQueueFull() syscall is used to inquire as to whether a message queue is full. A message queue is considered full if the length (i.e., number of messages) of a queue has reached its message limit which is configured using the CONFIG_QUEUE_MINIMUM_LIMIT (default is 5) setting.

See also

xReturn

xQueue

CONFIG_QUEUE_MINIMUM_LIMIT

Parameters

*queue_*	The message queue to be operated on.
*res_*	The result of the inquiry; taken here to mean "true" if the queue is full, "false" if it contains less than "limit" messages.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.21 xQueueLockQueue() `xReturn xQueueLockQueue (`

```
xQueue queue_ )
```

The xQueueLockQueue() syscall is used to lock a message queue. Locking a message queue prevents tasks from sending messages to the queue but does not prevent tasks from peeking, receiving or dropping messages from a message queue.

See also

xReturn

xQueue

xQueueUnLockQueue()

Parameters

*queue_*	The message queue to be operated on.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.22 xQueueMessagesWaiting()

```
xReturn xQueueMessagesWaiting (
    const xQueue queue_,
    xBase * res_ )
```

The xQueueMessagesWaiting() syscall is used to inquire as to whether a message queue has one or more messages waiting.

See also

xReturn

xQueue

Parameters

*queue_*	The message queue to be operated on.
*res_*	The result of the inquiry; taken here to mean "true" if there is one or more messages waiting.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.23 xQueuePeek()

```
xReturn xQueuePeek (
    const xQueue queue_,
    xQueueMessage * message_ )
```

The xQueuePeek() syscall is used to retrieve the next message from a message queue without dropping the message (i.e., peek at the message).

See also

xReturn

xQueue

xQueueMessage

xMemFree()

Parameters

*queue_*	The message queue to be operated on.
*message_*	The message retrieved from the message queue. The message must be freed by xMemFree().

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.24 xQueueReceive()

```
xReturn xQueueReceive (
    xQueue queue_,
    xQueueMessage * message_ )
```

The xQueueReceive() syscall has the effect of calling xQueuePeek() followed by xQueueDropMessage(). The syscall will receive the next message from the message queue if there is a waiting message.

See also

xReturn

xQueue

xQueueMessage

xMemFree()

Parameters

*queue_*	The message queue to be operated on.
*message_*	The message retrieved from the message queue. The message must be freed by xMemFree().

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.25 xQueueSend()

```
xReturn xQueueSend (
        xQueue queue_,
        const xBase bytes_,
        const xByte * value_ )
```

The xQueueSend() syscall is used to send a message to a message queue. The message value is an array of bytes (i.e., xByte) and cannot exceed CONFIG_MESSAGE_VALUE_BYTES (default is 8) bytes in size.

See also

xReturn

xQueue

xByte

CONFIG_MESSAGE_VALUE_BYTES

Parameters

*queue_*	The message queue to be operated on.
*bytes_*	The size, in bytes, of the message to send to the message queue. The size of the message cannot exceed the CONFIG_MESSAGE_VALUE_BYTES (default is 8) setting.
*value_*	The message to be sent to the queue.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.26 xQueueUnLockQueue()

```
xReturn xQueueUnLockQueue (
        xQueue queue_ )
```

The xQueueUnLockQueue() syscall is used to unlock a message queue that was previously locked by xQueueLockQueue(). Once a message queue is unlocked, tasks may resume sending messages to the message queue.

See also

xReturn

xQueue

xQueueLockQueue()

Parameters

*queue_*	The message queue to be operated on.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.27 xStreamBytesAvailable()

```
xReturn xStreamBytesAvailable (
    const xStreamBuffer stream_,
    xHalfWord * bytes_ )
```

The xStreamBytesAvailable() syscall is used to obtain the number of waiting (i.e., available) bytes in a stream buffer.

See also

xReturn

xStreamBuffer

Parameters

*stream_*	The stream buffer to be operated on.
*bytes_*	The number of available bytes in the stream buffer.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.28 xStreamCreate()

```
xReturn xStreamCreate (
    xStreamBuffer * stream_ )
```

The xStreamCreate() syscall is used to create a stream buffer which is used for inter-task communications. A stream buffer is similar to a message queue, however, it operates only on one byte at a time.

See also

xReturn

xStreamBuffer

xStreamDelete()

Parameters

*stream_*	The stream buffer to be operated on.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.29 xStreamDelete()

```
xReturn xStreamDelete (
        const xStreamBuffer stream_ )
```

The xStreamDelete() syscall is used to delete a stream buffer created by xStreamCreate().

See also

xReturn

xStreamBuffer

xStreamCreate()

Parameters

*stream_*	The stream buffer to be operated on.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.30 xStreamIsEmpty()

```
xReturn xStreamIsEmpty (
    const xStreamBuffer stream_,
    xBase * res_ )
```

The xStreamIsEmpty() syscall is used to inquire as to whether a stream buffer is empty. An empty stream buffer has zero waiting (i.e.,available) bytes.

See also

xReturn

xStreamBuffer

Parameters

*stream_*	The stream buffer to be operated on.
*res_*	The result of the inquiry; taken here to mean "true" if the length (i.e., number of waiting bytes) is zero.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.31 xStreamIsFull()

```
xReturn xStreamIsFull (
    const xStreamBuffer stream_,
    xBase * res_ )
```

The xStreamIsFull() syscall is used to inquire as to whether a stream buffer is full. An full stream buffer has CONFIG_STREAM_BUFFER_BYTES (default is 32) bytes waiting.

See also

xReturn

xStreamBuffer

CONFIG_STREAM_BUFFER_BYTES

Parameters

*stream_*	The stream buffer to be operated on.
*res_*	The result of the inquiry; taken here to mean "true" if the length (i.e., number of waiting bytes) is CONFIG_STREAM_BUFFER_BYTES bytes.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.32 xStreamReceive()

```
xReturn xStreamReceive (
    const xStreamBuffer stream_,
    xHalfWord * bytes_,
    xByte ** data_ )
```

The xStreamReceive() syscall is used to retrieve all waiting bytes from a stream buffer.

See also

xReturn

xByte

xStreamBuffer

xMemFree()

Parameters

*stream_*	The stream buffer to be operated on.
*bytes_*	The number of bytes retrieved from the stream buffer.
*data_*	The bytes retrieved from the stream buffer. The data must be freed by xMemFree().

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.33 xStreamReset()

```
xReturn xStreamReset (
    const xStreamBuffer stream_ )
```

The xStreamReset() syscall is used to reset a stream buffer. Resetting a stream buffer has the effect of clearing the stream buffer such that xStreamBytesAvailable() would return zero bytes available.

See also

xReturn

xStreamBuffer

Parameters

*stream_*	The stream buffer to be operated on.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.34 xStreamSend()

```
xReturn xStreamSend (
    xStreamBuffer stream_,
    const xByte byte_ )
```

The xStreamSend() syscall is used to send one byte to a stream buffer.

See also

xReturn

xByte

xStreamBuffer

Parameters

*stream_*	The stream buffer to be operated on.
*byte_*	The byte to send to the stream buffer.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.35 xSystemAssert()

```
xReturn xSystemAssert (
        const char * file_,
        const int line_ )
```

The xSystemAssert() syscall is used to raise a system assert. In order fot xSystemAssert() to have an effect the configuration setting CONFIG_SYSTEM_ASSERT_BEHAVIOR must be defined. That said, it is recommended that the ASSERT C macro be used in place of xSystemAssert(). In order for the ASSERT C macro to have any effect, the configuration setting CONFIG_ENABLE_SYSTEM_ASSERT must be defined.

See also

xReturn

CONFIG_SYSTEM_ASSERT_BEHAVIOR

CONFIG_ENABLE_SYSTEM_ASSERT

ASSERT

Parameters

*file_*	The C file where the assert occurred. This will be set by the ASSERT C macro.
*line_*	The C file line where the assert occurred. This will be set by the ASSERT C macro.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.36 xSystemGetSystemInfo()

```
xReturn xSystemGetSystemInfo (
        xSystemInfo * info_ )
```

The xSystemGetSystemInfo() syscall is used to inquire about the system. The information bout the system that may be obtained is the product (i.e., OS) name, version and number of tasks.

See also

xReturn

xSystemInfo

xMemFree()

Parameters

*info_*	The system information. The system information must be freed by xMemFree().

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.37 xSystemHalt() `xReturn xSystemHalt ( void )`

The xSystemHalt() syscall is used to halt HeliOS. Once called, xSystemHalt() will disable all interrupts and stops the execution of further statements. The system will have to be reset to recover.

See also

xReturn

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.38 xSystemInit() `xReturn xSystemInit ( void )`

The xSystemInit() syscall is used to bootstrap HeliOS and must be the first syscall made in the user's application. The xSystemInit() syscall initializes memory and calls initialization functions through the port layer.

See also

xReturn

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.39 xTaskChangePeriod()

```
xReturn xTaskChangePeriod (
        xTask task_,
        const xTicks period_ )
```

The xTaskChangePeriod() is used to change the interval period of a task timer. The period is measured in ticks. While architecture and/or platform dependent, a tick is often one millisecond. In order for the task timer to have an effect, the task must be in the "waiting" state which can be set using xTaskWait().

See also

xReturn

xTask

xTicks

xTaskWait()

Parameters

*task_*	The task to be operated on.
*period_*	The interval period in ticks.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.40 xTaskChangeWDPeriod()

```
xReturn xTaskChangeWDPeriod (
        xTask task_,
        const xTicks period_ )
```

The xTaskChangeWDPeriod() syscall is used to change the task watchdog timer period. This has no effect unless CONFIG_TASK_WD_TIMER_ENABLE is defined and the watchdog timer period is greater than zero. The task watchdog timer will place a task in a suspended state if a task's runtime exceeds the watchdog timer period. The task watchdog timer period is set on a per task basis.

See also

xReturn

xTask

xTicks

CONFIG_TASK_WD_TIMER_ENABLE

Parameters

*task_*	The task to be operated on.
*period_*	The task watchdog timer period measured in ticks. Ticks is platform and/or architecture dependent. However, most platforms and/or architectures have a one millisecond tick duration.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.41 xTaskCreate()

```
xReturn xTaskCreate (
    xTask * task_,
    const xByte * name_,
    void(*)(xTask task_, xTaskParm parm_) callback_,
    xTaskParm taskParameter_ )
```

The xTaskCreate() syscall is used to create a new task. Neither the xTaskCreate() or xTaskDelete() syscalls can be called from within a task (i.e., while the scheduler is running).

See also

xReturn

xTaskDelete()

xTask

xTaskParm

CONFIG_TASK_NAME_BYTES

Parameters

*task_*	The task to be operated on.
*name_*	The name of the task which must be exactly CONFIG_TASK_NAME_BYTES (default is 8) bytes in length. Shorter task names must be padded.
*callback_*	The task's main (i.e., entry point) function.
*taskParameter_*	A parameter which is accessible from the task's main function. If a task parameter is not needed, this parameter may be set to null.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn

(a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.42 xTaskDelete()

```
xReturn xTaskDelete (
    const xTask task_ )
```

The xTaskDelete() syscall is used to delete an existing task. Neither the xTaskCreate() or xTaskDelete() syscalls can be called from within a task (i.e., while the scheduler is running).

See also

xReturn

xTask

Parameters

*task_*	The task to be operated on.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.43 xTaskGetAllRunTimeStats()

```
xReturn xTaskGetAllRunTimeStats (
    xTaskRunTimeStats * stats_,
    xBase * tasks_ )
```

The xTaskGetAllRunTimeStats() syscall is used to obtain the runtime statistics of all tasks.

See also

xReturn

xTask

xTaskRunTimeStats

xMemFree()

Parameters

*stats_*	The runtime statistics. The runtime statics must be freed by xMemFree().
*tasks_*	The number of tasks in the runtime statistics.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.44 xTaskGetAllTaskInfo()

```
xReturn xTaskGetAllTaskInfo (
    xTaskInfo * info_,
    xBase * tasks_ )
```

The xTaskGetAllTaskInfo() syscall is used to get info about all tasks. xTaskGetAllTaskInfo() is similar to xTaskGetAllRunTimeStats() with one difference, xTaskGetAllTaskInfo() provides the state and name of the task along with the task's runtime statistics.

See also

xReturn

xTaskInfo

xMemFree()

Parameters

*info_*	Information about the tasks. The task information must be freed by xMemFree().
*tasks_*	The number of tasks.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.45 xTaskGetHandleById()

```
xReturn xTaskGetHandleById (
        xTask * task_,
        const xBase id_ )
```

The xTaskGetHandleById() syscall will get the task handle using the task id.

See also

xReturn

xTask

Parameters

*task_*	The task to be operated on.
*id_*	The task id.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.46 xTaskGetHandleByName()

```
xReturn xTaskGetHandleByName (
        xTask * task_,
        const xByte * name_ )
```

The xTaskGetHandleByName() syscall will get the task handle using the task name.

See also

xReturn

xTask

CONFIG_TASK_NAME_BYTES

Parameters

*task_*	The task to be operated on.
*name_*	The name of the task which must be exactly CONFIG_TASK_NAME_BYTES (default is 8) bytes in length. Shorter task names must be padded.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.47 xTaskGetId()

```
xReturn xTaskGetId (
    const xTask task_,
    xBase * id_ )
```

The xTaskGetId() syscall is used to obtain the id of a task.

See also

xReturn

xTask

Parameters

*task_*	The task to be operated on.
*id_*	The id of the task.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.48 xTaskGetName()

```
xReturn xTaskGetName (
    const xTask task_,
    xByte ** name_ )
```

The xTaskGetName() syscall is used to get the ASCII name of a task. The size of the task name is CONFIG_TASK_NAME_BYTES (default is 8) bytes in length.

See also

xReturn

xTask

xMemFree()

Parameters

*task_*	The task to be operated on.
*name_*	The task name which must be precisely CONFIG_TASK_NAME_BYTES (default is 8) bytes in length. The task name must be freed by xMemFree().

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.49 xTaskGetNumberOfTasks()

```
xReturn xTaskGetNumberOfTasks (
        xBase * tasks_ )
```

The xTaskGetNumberOfTasks() syscall is used to obtain the number of tasks regardless of their state (i.e., suspended, running or waiting).

See also

xReturn

Parameters

*tasks_*	The number of tasks.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.50 xTaskGetPeriod()

```
xReturn xTaskGetPeriod (
        const xTask task_,
        xTicks * period_ )
```

The xTaskGetPeriod() syscall is used to obtain the current task timer period.

See also

xReturn

xTask

xTicks

Parameters

*task_*	The task to be operated on.
*period_*	The task timer period in ticks. Ticks is platform and/or architecture dependent. However, most platforms and/or architect

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.51 xTaskGetSchedulerState() `xReturn xTaskGetSchedulerState ( xSchedulerState * state_ )`

The xTaskGetSchedulerState() is used to get the state of the scheduler.

See also

xReturn

xSchedulerState

Parameters

*state_*	The state of the scheduler.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.52 xTaskGetTaskInfo()

```
xReturn xTaskGetTaskInfo (
        const xTask task_,
        xTaskInfo * info_ )
```

The xTaskGetTaskInfo() syscall is used to get info about a single task. xTaskGetTaskInfo() is similar to xTaskGetTaskRunTimeStats() with one difference, xTaskGetTaskInfo() provides the state and name of the task along with the task's runtime statistics.

See also

xReturn

xMemFree()

xTask

xTaskInfo

Parameters

*task_*	The task to be operated on.
*info_*	Information about the task. The task information must be freed by xMemFree().

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.53 xTaskGetTaskRunTimeStats()

```
xReturn xTaskGetTaskRunTimeStats (
        const xTask task_,
        xTaskRunTimeStats * stats_ )
```

The xTaskGetTaskRunTimeStats() syscall is used to get the runtime statistics for a single task.

See also

xReturn

xTask

xTaskRunTimeStats

xMemFree()

Parameters

*task_*	The task to be operated on.
*stats_*	The runtime statistics. The runtime statistics must be freed by xMemFree().

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.54 xTaskGetTaskState()

```
xReturn xTaskGetTaskState (
    const xTask task_,
    xTaskState * state_ )
```

The xTaskGetTaskState() syscall is used to obtain the state of a task (i.e., suspended, running or waiting).

See also

xReturn

xTask

xTaskState

Parameters

*task_*	The task to be operated on.
*state_*	The state of the task.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.55 xTaskGetWDPeriod()

```
xReturn xTaskGetWDPeriod (
    const xTask task_,
    xTicks * period_ )
```

The xTaskGetWDPeriod() syscall is used to obtain the task watchdog timer period.

See also

xReturn

xTask

xTicks

CONFIG_TASK_WD_TIMER_ENABLE

Parameters

*task_*	The task to be operated on.
*period_*	The task watchdog timer period, measured in ticks. Ticks are platform and/or architecture dependent. However, on must platforms and/or architectures the tick represents one millisecond.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.56 xTaskNotificationIsWaiting()

```
xReturn xTaskNotificationIsWaiting (
    const xTask task_,
    xBase * res_ )
```

The xTaskNotificationIsWaiting() syscall is used to inquire as to whether a direct-to-task notification is waiting for the given task.

See also

xReturn

xTask

Parameters

*task_*	Task to be operated on.
*res_*	The result of the inquiry; taken here to mean "true" if there is a waiting direct-to-task notification. Otherwise "false", if there is not a waiting direct-to-notification.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.57 xTaskNotifyGive()

```
xReturn xTaskNotifyGive (
        xTask task_,
        const xBase bytes_,
        const xByte * value_ )
```

The xTaskNotifyGive() syscall is used to give (i.e., send) a direct-to-task notification to the given task.

See also

xReturn

xTask

CONFIG_NOTIFICATION_VALUE_BYTES

Parameters

*task_*	The task to be operated on.
*bytes_*	The number of bytes contained in the notification value. The number of bytes in the notification value cannot exceed CONFIG_NOTIFICATION_VALUE_BYTES (default is 8) bytes.
*value_*	The notification value which is a byte array whose length is defined by "bytes_".

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.58 xTaskNotifyStateClear()

```
xReturn xTaskNotifyStateClear (
        xTask task_ )
```

The xTaskNotifyStateClear() syscall is used to clear a waiting direct-to-task notification for the given task.

See also

xReturn

xTask

Parameters

*task_*	The task to be operated on.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.59 xTaskNotifyTake()

```
xReturn xTaskNotifyTake (
    xTask task_,
    xTaskNotification * notification_ )
```

The xTaskNotifyTake() syscall is used to take (i.e., receive) a waiting direct-to-task notification.

See also

xReturn

xTask

CONFIG_NOTIFICATION_VALUE_BYTES

xTaskNotification

Parameters

*task_*	The task to be operated on.
*notification_*	The direct-to-task notification.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.60 xTaskResetTimer()

```
xReturn xTaskResetTimer (
    xTask task_ )
```

The xTaskResetTimer() syscall is used to reset the task timer. In effect, this sets the elapsed time, measured in ticks, back to zero.

See also

xReturn

xTask

xTicks

Parameters

*task_*	The task to be operated on.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.61 xTaskResume()

```
xReturn xTaskResume (
        xTask task_ )
```

The xTaskResume() syscall will place a task in the "running" state. A task in this state will run continuously until suspended and is scheduled to run cooperatively by the HeliOS scheduler.

See also

xReturn

xTask

xTaskResume()

xTaskSuspend()

xTaskWait()

Parameters

*task_*	The task to be operated on.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.62 xTaskResumeAll()

```
xReturn xTaskResumeAll (
        void )
```

The xTaskResumeAll() syscall is used to set the scheduler state to running. xTaskStartScheduler() must still be called to pass control to the scheduler. If the scheduler state is not running, then xTaskStartScheduler() will simply return to the caller when called.

See also

xReturn

xTaskStartScheduler()

xTaskResumeAll()

xTaskSuspendAll()

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.63 xTaskStartScheduler() `xReturn xTaskStartScheduler ( void )`

The xTaskStartScheduler() syscall is used to start the HeliOS task scheduler. On this syscall is made, control is handed over to HeliOS. In order to suspend the scheduler and return to the caller, the xTaskSuspendAll() syscall will need to be made. Once a call to xTaskSuspendAll() is made, xTaskResumeAll() must be called before calling xTaskStartScheduler() again. If xTaskStartScheduler() is called while the scheduler is in a suspended state, xTaskStartScheduler() will immediately return.

See also

xReturn

xTaskResumeAll()

xTaskSuspendAll()

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.64 xTaskSuspend() `xReturn xTaskSuspend ( xTask task_ )`

The xTaskSuspend() syscall will place a task in the "suspended" state. A task in this state is not scheduled to run by the HeliOS scheduler and will not run.

See also

xReturn

xTask

xTaskResume()

xTaskSuspend()

xTaskWait()

Parameters

*task_*	The task to be operated on.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.65 xTaskSuspendAll() `xReturn xTaskSuspendAll ( void )`

The xTaskSuspendAll() syscall is used to set the scheduler state to suspended. If called from a running task, the HeliOS scheduler will quit and return control back to the caller. To set the scheduler state to running, xTaskResumeAll() must be called followed by a call to xTaskStartScheduler().

See also

xReturn

xTaskStartScheduler()

xTaskResumeAll()

xTaskSuspendAll()

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.66 xTaskWait()

```
xReturn xTaskWait (
        xTask task_ )
```

The xTaskWait() syscall will place a task in the "waiting" state. A task in this state is not scheduled to run by the HeliOS scheduler *UNTIL* an event occurs. When an event occurs, the HeliOS will schedule the task to run until the even has passed (e.g., the task either "takes" or "clears a direct-to-task notification"). Tasks in the "waiting" state are tasks that are using event-driven multitasking. HeliOS supports two types of events: task timers and direct-to-task notifications.

See also

xReturn

xTask

xTaskResume()

xTaskSuspend()

xTaskWait()

Parameters

*task_*	The task to be operated on.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.67 xTimerChangePeriod()

```
xReturn xTimerChangePeriod (
        xTimer timer_,
        const xTicks period_ )
```

The xTimerChangePeriod() syscall is used to change the time period on an application timer. Once the period has elapsed, the application timer is considered expired.

See also

xReturn

xTimer

xTicks

Parameters

*timer_*	The application timer to be operated on.
*period_*	The application timer period, measured in ticks.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.68 xTimerCreate()

```
xReturn xTimerCreate (
    xTimer * timer_,
    const xTicks period_ )
```

The xTimerCreate() syscall is used to create a new application timer. Application timers are not the same as task timers. Application timers are not part of HeliOS's event-driven multitasking. Application timers are just that, timers for use by the user's application for general purpose timekeeping. Application timers can be started, stopped, reset and have time period, measured in ticks, that elapses.

See also

xReturn

xTimer

xTicks

xTimerDelete()

Parameters

*timer_*	The application timer to be operated on.
*period_*	The application timer period, measured in ticks.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.69 xTimerDelete()

```
xReturn xTimerDelete (
    const xTimer timer_ )
```

The xTimerDelete() syscall is used to delete an application timer created with xTimerCreate().

See also

xReturn

xTimer

xTicks

xTimerCreate()

Parameters

*timer_*	The application timer to be operated on.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.70 xTimerGetPeriod()

```
xReturn xTimerGetPeriod (
    const xTimer timer_,
    xTicks * period_ )
```

The xTimerGetPeriod() syscall is used to obtain the current period for an application timer.

See also

xReturn

xTimer

xTicks

Parameters

*timer_*	The application timer to be operate don.
*period_*	The application timer period, measured in ticks.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.71 xTimerHasTimerExpired()

```
xReturn xTimerHasTimerExpired (
    const xTimer timer_,
    xBase * res_ )
```

The xTimerHasTimerExpired() syscall is used to inquire as to whether an application timer has expired. If the application timer has expired, it must be reset with xTimerReset(). If a timer is not active (i.e., started), it cannot expire even if the timer period has elapsed.

See also

xReturn

xTimer

xTimerReset()

Parameters

*timer_*	The application timer to be operated on.
*res_*	The result of the inquiry; taken here to mean "true" if the application timer has elapsed (i.e., expired). "False" if the application timer has not expired

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.72 xTimerIsTimerActive()

```
xReturn xTimerIsTimerActive (
    const xTimer timer_,
    xBase * res_ )
```

The xTimerIsTimerActive() syscall is used to inquire as to whether an application timer is active. An application timer is considered to be active if the application timer has been started by xTimerStare().

See also

xReturn

xTimer

xTimerStart()

xTimerStop()

Parameters

*timer_*	The application timer to be operated on.
*res_*	The result of the inquiry; taken here to mean "true" if the application timer is running. "False" if the application timer is not running.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.73 xTimerReset() `xReturn xTimerReset ( xTimer timer_ )`

The xTimerReset() syscall is used to reset an application timer. Resetting has the effect of setting the application timer's elapsed time to zero.

See also

xReturn

xTimer

xTimerReset()

xTimerStart()

xTimerStop()

Parameters

*timer_*	The application timer to be operated on.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.74 xTimerStart()

```
xReturn xTimerStart (
        xTimer timer_ )
```

The xTimerStart() syscall is used to place an application timer in the running state.

See also

xReturn

xTimer

xTimerReset()

xTimerStart()

xTimerStop()

Parameters

*timer_*	The application timer to be operated on.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

4.2.4.75 xTimerStop()

```
xReturn xTimerStop (
        xTimer timer_ )
```

The xTimerStop() syscall is used to place an application timer in the suspended state.

See also

xReturn

xTimer

xTimerReset()

xTimerStart()

xTimerStop()

Parameters

*timer_*	The application timer to be operated on.

Returns

On success, the syscall returns ReturnOK. On failure, the syscall returns ReturnError. A failure is any condition in which the syscall was unable to achieve its intended objective. For example, if xTaskGetId() was unable to locate the task by the task object (i.e., xTask) passed to the syscall, because either the object was null or invalid (e.g., a deleted task), xTaskGetId() would return ReturnError. All HeliOS syscalls return the xReturn (a.k.a., Return_t) type which can either be ReturnOK or ReturnError. The C macros OK() and ERROR() can be used as a more concise way of checking the return value of a syscall (e.g., if(OK(xMemGetUsed(&size))) {} or if(ERROR(xMemGetUsed(&size))) {}).

Index