

Solaris Packet Protocol

0.1

Generated by Doxygen 1.9.8

1 File Index	1
1.1 File List	1
2 File Documentation	3
2.1 hal/spi/spi.c File Reference	3
2.1.1 Detailed Description	3
2.1.2 Function Documentation	4
2.1.2.1 __attribute__()	4
2.2 spi.c	5
2.3 hal/spi/spi.h File Reference	6
2.3.1 Detailed Description	7
2.3.2 Function Documentation	7
2.3.2.1 SPP_HAL_SPI_BusInit()	7
2.3.2.2 SPP_HAL_SPI_DeviceInit()	7
2.3.2.3 SPP_HAL_SPI_GetHandler()	7
2.3.2.4 SPP_HAL_SPI_Transmit()	8
2.4 spi.h	8
2.5 osal/mutex.c File Reference	9
2.5.1 Detailed Description	9
2.5.2 Function Documentation	10
2.5.2.1 __attribute__()	10
2.6 mutex.c	10
2.7 osal/mutex.h File Reference	11
2.7.1 Detailed Description	12
2.7.2 Typedef Documentation	13
2.7.2.1 osal_mutex_handle_t	13
2.7.3 Enumeration Type Documentation	13
2.7.3.1 osal_mutex_type_t	13
2.7.4 Function Documentation	13
2.7.4.1 OSAL_MutexCreate()	13
2.7.4.2 OSAL_MutexDelete()	14
2.7.4.3 OSAL_MutexGetHolder()	14
2.7.4.4 OSAL_MutexGive()	14
2.7.4.5 OSAL_MutexIsHeldByCurrentTask()	14
2.7.4.6 OSAL_MutexTake()	15
2.7.4.7 OSAL_MutexTryTake()	15
2.8 mutex.h	15
2.9 osal/osal.c File Reference	16
2.9.1 Detailed Description	17
2.9.2 Function Documentation	17
2.9.2.1 __attribute__()	17
2.10 osal.c	18

2.11 osal/osal.h File Reference	20
2.11.1 Detailed Description	22
2.11.2 Typedef Documentation	23
2.11.2.1 osal_mutex_handle_t	23
2.11.2.2 osal_queue_handle_t	23
2.11.2.3 osal_semaphore_handle_t	23
2.11.2.4 osal_task_function_t	23
2.11.2.5 osal_task_handle_t	23
2.11.3 Enumeration Type Documentation	23
2.11.3.1 osal_priority_t	23
2.11.4 Function Documentation	24
2.11.4.1 OSAL_Deinit()	24
2.11.4.2 OSAL_GetTickCount()	24
2.11.4.3 OSAL_Init()	24
2.11.4.4 OSAL_MsToTicks()	24
2.11.4.5 OSAL_MutexCreate()	25
2.11.4.6 OSAL_MutexDelete()	25
2.11.4.7 OSAL_MutexGive()	25
2.11.4.8 OSAL_MutexTake()	26
2.11.4.9 OSAL_QueueCreate()	26
2.11.4.10 OSAL_QueueDelete()	26
2.11.4.11 OSAL_QueueReceive()	27
2.11.4.12 OSAL_QueueSend()	27
2.11.4.13 OSAL_SemaphoreCreate()	28
2.11.4.14 OSAL_SemaphoreDelete()	28
2.11.4.15 OSAL_SemaphoreGive()	28
2.11.4.16 OSAL_SemaphoreTake()	29
2.11.4.17 OSAL_StartScheduler()	29
2.11.4.18 OSAL_TaskCreate()	29
2.11.4.19 OSAL_TaskDelay()	30
2.11.4.20 OSAL_TaskDelete()	30
2.12 osal.h	30
2.13 osal/queue.c File Reference	31
2.13.1 Detailed Description	32
2.13.2 Function Documentation	32
2.13.2.1 __attribute__((...))	32
2.14 queue.c	33
2.15 osal/queue.h File Reference	35
2.15.1 Detailed Description	36
2.15.2 Typedef Documentation	37
2.15.2.1 osal_queue_handle_t	37
2.15.3 Function Documentation	37

2.15.3.1 OSAL_QueueCreate()	37
2.15.3.2 OSAL_QueueDelete()	37
2.15.3.3 OSAL_QueueGetCount()	38
2.15.3.4 OSAL_QueueGetSpace()	38
2.15.3.5 OSAL_QueueIsEmpty()	38
2.15.3.6 OSAL_QueueIsFull()	38
2.15.3.7 OSAL_QueuePeek()	39
2.15.3.8 OSAL_QueueReceive()	39
2.15.3.9 OSAL_QueueReceiveFromISR()	40
2.15.3.10 OSAL_QueueReset()	40
2.15.3.11 OSAL_QueueSend()	40
2.15.3.12 OSAL_QueueSendFromISR()	41
2.16 queue.h	41
2.17 osal/semaphore.c File Reference	42
2.17.1 Detailed Description	42
2.17.2 Function Documentation	43
2.17.2.1 __attribute__((__unused__))	43
2.18 semaphore.c	44
2.19 osal/semaphore.h File Reference	45
2.19.1 Detailed Description	46
2.19.2 Typedef Documentation	46
2.19.2.1 osal_semaphore_handle_t	46
2.19.3 Function Documentation	46
2.19.3.1 OSAL_SemaphoreCreate()	46
2.19.3.2 OSAL_SemaphoreCreateBinary()	47
2.19.3.3 OSAL_SemaphoreDelete()	47
2.19.3.4 OSAL_SemaphoreGetCount()	47
2.19.3.5 OSAL_SemaphoreGive()	48
2.19.3.6 OSAL_SemaphoreGiveFromISR()	48
2.19.3.7 OSAL_SemaphoreTake()	48
2.19.3.8 OSAL_SemaphoreTryTake()	49
2.20 semaphore.h	49
2.21 osal/task.c File Reference	50
2.21.1 Detailed Description	50
2.21.2 Function Documentation	50
2.21.2.1 __attribute__((__unused__))	50
2.22 task.c	51
2.23 osal/task.h File Reference	51
2.23.1 Detailed Description	52
2.23.2 Function Documentation	52
2.23.2.1 SPP_OSAL_GetTaskStorage()	52
2.23.2.2 SPP_OSAL_TaskCreate()	52

2.23.2.3 SPP_OSAL_TaskDelete()	52
2.24 task.h	53
Index	55

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

hal/spi/spi.c	
SPI Hardware Abstraction Layer Implementation	3
hal/spi/spi.h	
SPI Hardware Abstraction Layer	6
osal/mutex.c	
OSAL Mutex Management Implementation	9
osal/mutex.h	
OSAL Mutex Management Interface	11
osal/osal.c	
Operating System Abstraction Layer Implementation	16
osal/osal.h	
Operating System Abstraction Layer	20
osal/queue.c	
OSAL Queue Management Implementation	31
osal/queue.h	
OSAL Queue Management Interface	35
osal/semaphore.c	
OSAL Semaphore Management Implementation	42
osal/semaphore.h	
OSAL Semaphore Management Interface	45
osal/task.c	
OSAL Task Management Implementation	50
osal/task.h	
OSAL Task Management Interface	51

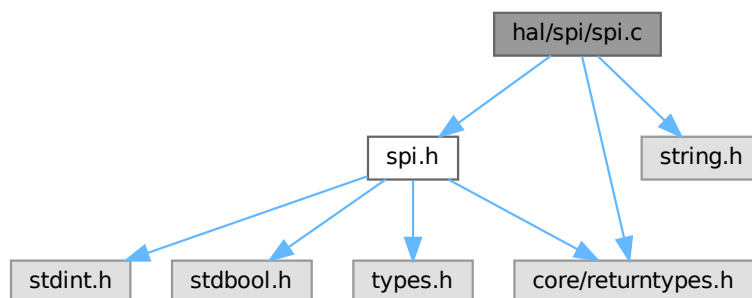
Chapter 2

File Documentation

2.1 hal/spi/spi.c File Reference

SPI Hardware Abstraction Layer Implementation.

```
#include "spi.h"  
#include "core/returntypes.h"  
#include <string.h>  
Include dependency graph for spi.c:
```



Functions

- `__attribute__((weak))`
Default (weak) SPI bus configuration.

2.1.1 Detailed Description

SPI Hardware Abstraction Layer Implementation.

Version

1.0.0

Date

2024

This file provides the default (weak) implementation of SPI HAL functions that can be overridden by platform-specific implementations.

Definition in file [spi.c](#).

2.1.2 Function Documentation

2.1.2.1 `__attribute__()`

```
__attribute__ (
    (weak) )
```

Default (weak) SPI bus configuration.

Default (weak) get semaphore count.

Default (weak) semaphore try take.

Default (weak) semaphore give from ISR.

Default (weak) binary semaphore creation.

Default (weak) reset queue.

Default (weak) check if queue is empty.

Default (weak) check if queue is full.

Default (weak) get queue space.

Default (weak) get queue count.

Default (weak) queue peek.

Default (weak) queue receive from ISR.

Default (weak) queue send from ISR.

Default (weak) start scheduler.

Default (weak) convert milliseconds to ticks.

Default (weak) get tick count.

Default (weak) semaphore give.

Default (weak) semaphore take.

Default (weak) semaphore deletion.

Default (weak) semaphore creation.

Default (weak) queue receive.

Default (weak) queue send.

Default (weak) queue deletion.

Default (weak) queue creation.

Default (weak) mutex creation.

Default (weak) task delay.

Default (weak) task deletion.

Default (weak) task creation.

Default (weak) OSAL deinitialization.

Default (weak) check if mutex is held by current task.

Default (weak) get mutex holder.

Default (weak) mutex try take.

Default (weak) mutex give.

Default (weak) mutex take.

Default (weak) mutex deletion.

Default (weak) SPI transmit function.

Default (weak) SPI device initialization.

Default (weak) SPI handler assigner (void pointers)

Definition at line 19 of file [spi.c](#).

```
00019                                     {
00020     return SPP_ERROR;
00021 }
```

2.2 spi.c

[Go to the documentation of this file.](#)

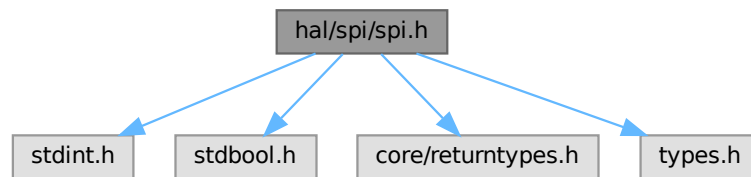
```
00001
00011 #include "spi.h"
00012 #include "core/returntypes.h"
00013 #include <string.h>
00014
00015
00019 __attribute__((weak)) retval_t SPP_HAL_SPI_BusInit() {
00020     return SPP_ERROR;
00021 }
00022
00023
00027 __attribute__((weak)) void* SPP_HAL_SPI_GetHandler() {
00028     return NULL;
00029 }
00030
00034 __attribute__((weak)) retval_t SPP_HAL_SPI_DeviceInit(void* handler) {
00035     return SPP_ERROR;
00036 }
00037
00041 __attribute__((weak)) retval_t SPP_HAL_SPI_Transmit(void* handler, void* data_to_send, void*
data_to_recieve, spp_uint8_t length) {
00042     return SPP_ERROR;
00043 }
00044
```

2.3 hal/spi/spi.h File Reference

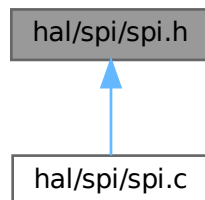
SPI Hardware Abstraction Layer.

```
#include <stdint.h>
#include <stdbool.h>
#include "core/returntypes.h"
#include "types.h"
```

Include dependency graph for spi.h:



This graph shows which files directly or indirectly include this file:



Functions

- `retval_t SPP_HAL_SPI_BusInit (void)`
Initialize SPI bus configuration.
- `void * SPP_HAL_SPI_GetHandler (void)`
Sets a handler for every peripheral ([0] reserved for BMP and [1] reserved for ICM)
- `retval_t SPP_HAL_SPI_DeviceInit (void *p_handler)`
Initialize SPI peripheral and defines transaction details.
- `retval_t SPP_HAL_SPI_Transmit (void *handler, void *data_to_send, void *data_to_recieve, spp_uint8_t length)`
Transmit data over SPI.

2.3.1 Detailed Description

SPI Hardware Abstraction Layer.

Version

1.0.0

Date

2024

This file provides the SPI Hardware Abstraction Layer (HAL) interface for the Solaris Packet Protocol library.

Definition in file [spi.h](#).

2.3.2 Function Documentation

2.3.2.1 SPP_HAL_SPI_BusInit()

```
retval_t SPP_HAL_SPI_BusInit (  
    void )
```

Initialize SPI bus configuration.

Returns

retval_t SPP_OK on success, error code otherwise

2.3.2.2 SPP_HAL_SPI_DeviceInit()

```
retval_t SPP_HAL_SPI_DeviceInit (  
    void * p_handler )
```

Initialize SPI peripheral and defines transaction details.

Parameters

<i>p_handler</i>	Void pointer to peripheral device (device_id[0] on BMP or device_id[1] on ICM)
------------------	--

Returns

retval_t SPP_OK on success, error code otherwise

2.3.2.3 SPP_HAL_SPI_GetHandler()

```
void * SPP_HAL_SPI_GetHandler (  
    void )
```

Sets a handler for every peripheral ([0] reserved for BMP and [1] reserved for ICM)

Returns

a void pointer to an `device_id` array position, NULL otherwise

2.3.2.4 SPP_HAL_SPI_Transmit()

```
retval_t SPP_HAL_SPI_Transmit (
    void * handler,
    void * data_to_send,
    void * data_to_recieve,
    spp_uint8_t length )
```

Transmit data over SPI.

Parameters

<i>handle</i>	Pointer to SPI handle
<i>data_to_send</i>	Pointer to the data to transmit
<i>data_to_recieve</i>	Pointer to where the data recieved will be stored
<i>length</i>	Size of the buffer to transmit data (recieve buffer should be half of the size)

Returns

`retval_t` SPP_OK on success, error code otherwise

2.4 spi.h

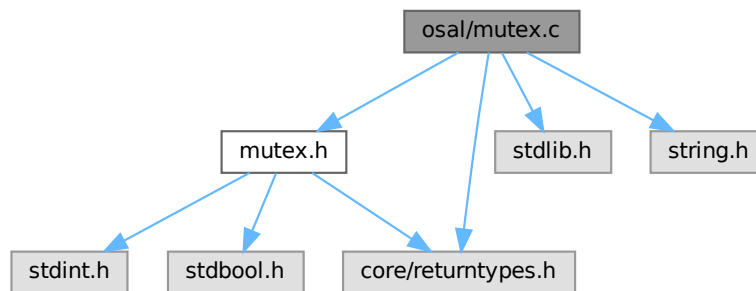
[Go to the documentation of this file.](#)

```
00001
00011 #ifndef SPP_HAL_SPI_H
00012 #define SPP_HAL_SPI_H
00013
00014 #include <stdint.h>
00015 #include <stdbool.h>
00016 #include "core/returntypes.h"
00017 #include "types.h"
00018
00019 #ifdef __cplusplus
00020 extern "C" {
00021 #endif
00022
00028 retval_t SPP_HAL_SPI_BusInit(void);
00029
00030
00036 void* SPP_HAL_SPI_GetHandler(void);
00037
00038
00045 retval_t SPP_HAL_SPI_DeviceInit(void* p_handler);
00046
00056 retval_t SPP_HAL_SPI_Transmit(void* handler, void* data_to_send, void* data_to_recieve, spp_uint8_t
length);
00057
00058
00059 #ifdef __cplusplus
00060 }
00061 #endif
00062
00063 #endif /* SPP_HAL_SPI_H */
```

2.5 osal/mutex.c File Reference

OSAL Mutex Management Implementation.

```
#include "mutex.h"  
#include "core/returntypes.h"  
#include <stdlib.h>  
#include <string.h>  
Include dependency graph for mutex.c:
```



Functions

- [__attribute__\(\(weak\)\)](#)
Default (weak) mutex creation.

2.5.1 Detailed Description

OSAL Mutex Management Implementation.

Version

1.0.0

Date

2024

This file provides the default (weak) implementation of mutex management functions.

Definition in file [mutex.c](#).

2.5.2 Function Documentation

2.5.2.1 `__attribute__()`

```
__attribute__ (
    (weak) )
```

Default (weak) mutex creation.

Default (weak) check if mutex is held by current task.

Default (weak) get mutex holder.

Default (weak) mutex try take.

Default (weak) mutex give.

Default (weak) mutex take.

Default (weak) mutex deletion.

Definition at line 18 of file [mutex.c](#).

```
00019 {
00020     if (mutex_handle == NULL) {
00021         return SPP_ERROR_NULL_POINTER;
00022     }
00023
00024     // Default implementation - simulate mutex creation
00025     *mutex_handle = (void*)0x87654321;
00026     (void)type;
00027     return SPP_OK;
00028 }
```

2.6 mutex.c

[Go to the documentation of this file.](#)

```
00001
00010 #include "mutex.h"
00011 #include "core/returntypes.h"
00012 #include <stdlib.h>
00013 #include <string.h>
00014
00018 __attribute__((weak)) retval_t OSAL_MutexCreate(osal_mutex_handle_t* mutex_handle, osal_mutex_type_t
type)
00019 {
00020     if (mutex_handle == NULL) {
00021         return SPP_ERROR_NULL_POINTER;
00022     }
00023
00024     // Default implementation - simulate mutex creation
00025     *mutex_handle = (void*)0x87654321;
00026     (void)type;
00027     return SPP_OK;
00028 }
00029
00033 __attribute__((weak)) retval_t OSAL_MutexDelete(osal_mutex_handle_t mutex_handle)
00034 {
00035     // Default implementation - just return OK
00036     (void)mutex_handle;
00037     return SPP_OK;
00038 }
00039
00043 __attribute__((weak)) retval_t OSAL_MutexTake(osal_mutex_handle_t mutex_handle, uint32_t timeout_ms)
00044 {
00045     if (mutex_handle == NULL) {
00046         return SPP_ERROR_NULL_POINTER;
00047     }
00048
00049     // Default implementation - always succeeds immediately
00050     (void)timeout_ms;
```



```

00051     return SPP_OK;
00052 }
00053
00057 __attribute__((weak)) retval_t OSAL_MutexGive(osal_mutex_handle_t mutex_handle)
00058 {
00059     if (mutex_handle == NULL) {
00060         return SPP_ERROR_NULL_POINTER;
00061     }
00062
00063     // Default implementation - just return OK
00064     return SPP_OK;
00065 }
00066
00070 __attribute__((weak)) retval_t OSAL_MutexTryTake(osal_mutex_handle_t mutex_handle)
00071 {
00072     if (mutex_handle == NULL) {
00073         return SPP_ERROR_NULL_POINTER;
00074     }
00075
00076     // Default implementation - always succeeds
00077     return SPP_OK;
00078 }
00079
00083 __attribute__((weak)) void* OSAL_MutexGetHolder(osal_mutex_handle_t mutex_handle)
00084 {
00085     // Default implementation - return dummy task handle
00086     (void)mutex_handle;
00087     return (void*)0x11111111;
00088 }
00089
00093 __attribute__((weak)) bool OSAL_MutexIsHeldByCurrentTask(osal_mutex_handle_t mutex_handle)
00094 {
00095     // Default implementation - always return true
00096     (void)mutex_handle;
00097     return true;
00098 }

```

2.7 osal/mutex.h File Reference

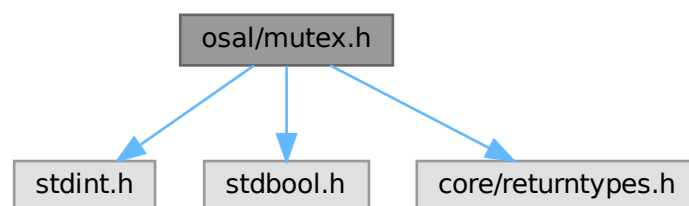
OSAL Mutex Management Interface.

```

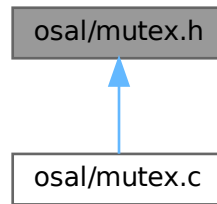
#include <stdint.h>
#include <stdbool.h>
#include "core/returntypes.h"

```

Include dependency graph for mutex.h:



This graph shows which files directly or indirectly include this file:



Typedefs

- typedef void * [osal_mutex_handle_t](#)
OSAL Mutex handle type.

Enumerations

- enum [osal_mutex_type_t](#) { [OSAL_MUTEX_NORMAL](#) = 0 , [OSAL_MUTEX_RECURSIVE](#) = 1 }
Mutex type enumeration.

Functions

- [retval_t OSAL_MutexCreate](#) ([osal_mutex_handle_t](#) *mutex_handle, [osal_mutex_type_t](#) type)
Create a mutex.
- [retval_t OSAL_MutexDelete](#) ([osal_mutex_handle_t](#) mutex_handle)
Delete a mutex.
- [retval_t OSAL_MutexTake](#) ([osal_mutex_handle_t](#) mutex_handle, [uint32_t](#) timeout_ms)
Take (lock) a mutex.
- [retval_t OSAL_MutexGive](#) ([osal_mutex_handle_t](#) mutex_handle)
Give (unlock) a mutex.
- [retval_t OSAL_MutexTryTake](#) ([osal_mutex_handle_t](#) mutex_handle)
Try to take a mutex without blocking.
- void * [OSAL_MutexGetHolder](#) ([osal_mutex_handle_t](#) mutex_handle)
Get mutex holder task.
- bool [OSAL_MutexIsHeldByCurrentTask](#) ([osal_mutex_handle_t](#) mutex_handle)
Check if mutex is held by current task.

2.7.1 Detailed Description

OSAL Mutex Management Interface.

Version

1.0.0

Date

2024

This file provides the mutex management interface for the OSAL layer.

Definition in file [mutex.h](#).

2.7.2 Typedef Documentation

2.7.2.1 osal_mutex_handle_t

```
typedef void* osal_mutex_handle_t
```

OSAL Mutex handle type.

Definition at line 24 of file [mutex.h](#).

2.7.3 Enumeration Type Documentation

2.7.3.1 osal_mutex_type_t

```
enum osal_mutex_type_t
```

Mutex type enumeration.

Enumerator

OSAL_MUTEX_NORMAL	
OSAL_MUTEX_RECURSIVE	

Definition at line 29 of file [mutex.h](#).

```
00029     {  
00030         OSAL_MUTEX_NORMAL = 0,  
00031         OSAL_MUTEX_RECURSIVE = 1  
00032     } osal_mutex_type_t;
```

2.7.4 Function Documentation

2.7.4.1 OSAL_MutexCreate()

```
retval_t OSAL_MutexCreate (  
    osal_mutex_handle_t * mutex_handle,  
    osal_mutex_type_t type )
```

Create a mutex.

Parameters

<i>mutex_handle</i>	Pointer to store mutex handle
<i>type</i>	Mutex type (normal or recursive)

Returns

retval_t SPP_OK on success, error code otherwise

2.7.4.2 OSAL_MutexDelete()

```
retval_t OSAL_MutexDelete (
    osal_mutex_handle_t mutex_handle )
```

Delete a mutex.

Parameters

<i>mutex_handle</i>	Mutex handle
---------------------	--------------

Returns

retval_t SPP_OK on success, error code otherwise

2.7.4.3 OSAL_MutexGetHolder()

```
void * OSAL_MutexGetHolder (
    osal_mutex_handle_t mutex_handle )
```

Get mutex holder task.

Parameters

<i>mutex_handle</i>	Mutex handle
---------------------	--------------

Returns

void* Task handle that currently holds the mutex (NULL if not held)

2.7.4.4 OSAL_MutexGive()

```
retval_t OSAL_MutexGive (
    osal_mutex_handle_t mutex_handle )
```

Give (unlock) a mutex.

Parameters

<i>mutex_handle</i>	Mutex handle
---------------------	--------------

Returns

retval_t SPP_OK on success, error code otherwise

2.7.4.5 OSAL_MutexIsHeldByCurrentTask()

```
bool OSAL_MutexIsHeldByCurrentTask (
    osal_mutex_handle_t mutex_handle )
```

Check if mutex is held by current task.

Parameters

<i>mutex_handle</i>	Mutex handle
---------------------	--------------

Returns

bool true if mutex is held by current task, false otherwise

2.7.4.6 OSAL_MutexTake()

```
retval_t OSAL_MutexTake (
    osal_mutex_handle_t mutex_handle,
    uint32_t timeout_ms )
```

Take (lock) a mutex.

Parameters

<i>mutex_handle</i>	Mutex handle
<i>timeout_ms</i>	Timeout in milliseconds (0 = no wait, UINT32_MAX = wait forever)

Returns

retval_t SPP_OK on success, error code otherwise

2.7.4.7 OSAL_MutexTryTake()

```
retval_t OSAL_MutexTryTake (
    osal_mutex_handle_t mutex_handle )
```

Try to take a mutex without blocking.

Parameters

<i>mutex_handle</i>	Mutex handle
---------------------	--------------

Returns

retval_t SPP_OK on success, SPP_ERROR_TIMEOUT if mutex not available

2.8 mutex.h

[Go to the documentation of this file.](#)

00001

```

00010 #ifndef SPP_OSAL_MUTEX_H
00011 #define SPP_OSAL_MUTEX_H
00012
00013 #include <stdint.h>
00014 #include <stdbool.h>
00015 #include "core/returntypes.h"
00016
00017 #ifdef __cplusplus
00018 extern "C" {
00019 #endif
00020
00024 typedef void* osal_mutex_handle_t;
00025
00029 typedef enum {
00030     OSAL_MUTEX_NORMAL = 0,
00031     OSAL_MUTEX_RECURSIVE = 1
00032 } osal_mutex_type_t;
00033
00041 retval_t OSAL_MutexCreate(osal_mutex_handle_t* mutex_handle, osal_mutex_type_t type);
00042
00049 retval_t OSAL_MutexDelete(osal_mutex_handle_t mutex_handle);
00050
00058 retval_t OSAL_MutexTake(osal_mutex_handle_t mutex_handle, uint32_t timeout_ms);
00059
00066 retval_t OSAL_MutexGive(osal_mutex_handle_t mutex_handle);
00067
00074 retval_t OSAL_MutexTryTake(osal_mutex_handle_t mutex_handle);
00075
00082 void* OSAL_MutexGetHolder(osal_mutex_handle_t mutex_handle);
00083
00090 bool OSAL_MutexIsHeldByCurrentTask(osal_mutex_handle_t mutex_handle);
00091
00092 #ifdef __cplusplus
00093 }
00094 #endif
00095
00096 #endif /* SPP_OSAL_MUTEX_H */

```

2.9 osal/osal.c File Reference

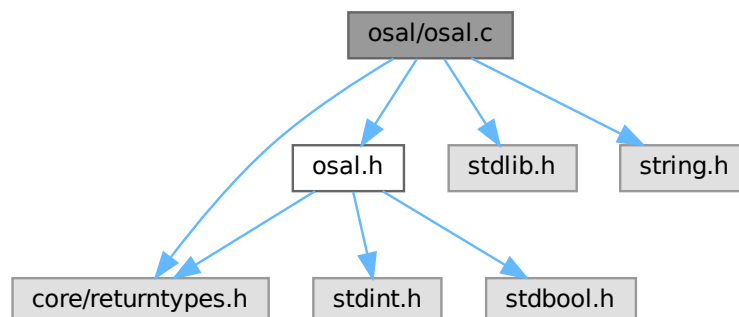
Operating System Abstraction Layer Implementation.

```

#include "core/returntypes.h"
#include "osal.h"
#include <stdlib.h>
#include <string.h>

```

Include dependency graph for osal.c:



Functions

- [__attribute__](#) ((weak))
Default (weak) OSAL initialization.

2.9.1 Detailed Description

Operating System Abstraction Layer Implementation.

Version

1.0.0

Date

2024

This file provides the default (weak) implementation of OSAL functions that can be overridden by OS-specific implementations.

Definition in file [osal.c](#).

2.9.2 Function Documentation

2.9.2.1 [__attribute__](#)()

```
\_\_attribute\_\_ (  
                (weak) )
```

Default (weak) OSAL initialization.

Default (weak) start scheduler.

Default (weak) convert milliseconds to ticks.

Default (weak) get tick count.

Default (weak) semaphore give.

Default (weak) semaphore take.

Default (weak) semaphore deletion.

Default (weak) semaphore creation.

Default (weak) queue receive.

Default (weak) queue send.

Default (weak) queue deletion.

Default (weak) queue creation.

Default (weak) mutex give.

Default (weak) mutex take.

Default (weak) mutex deletion.

Default (weak) mutex creation.

Default (weak) task delay.

Default (weak) task deletion.

Default (weak) task creation.

Default (weak) OSAL deinitialization.

Definition at line 19 of file [osal.c](#).

```
00020 {  
00021     // Default implementation - just return OK  
00022     return SPP_OK;  
00023 }
```

2.10 osal.c

[Go to the documentation of this file.](#)

```

00001
00011 #include "core/returntypes.h"
00012 #include "osal.h"
00013 #include <stdlib.h>
00014 #include <string.h>
00015
00019 __attribute__((weak)) retval_t OSAL_Init(void)
00020 {
00021     // Default implementation - just return OK
00022     return SPP_OK;
00023 }
00024
00028 __attribute__((weak)) retval_t OSAL_Deinit(void)
00029 {
00030     // Default implementation - just return OK
00031     return SPP_OK;
00032 }
00033
00037 __attribute__((weak)) retval_t OSAL_TaskCreate(osal_task_function_t task_function, const char* name,
00038         uint32_t stack_size, void* parameters,osal_priority_t
priority,
osal_task_handle_t* task_handle)
00039 {
00040     if (task_function == NULL || task_handle == NULL) {
00041         return SPP_ERROR_NULL_POINTER;
00042     }
00043
00044     // Default implementation - simulate task creation
00045     *task_handle = (void*)0x12345678;
00046     (void)name;
00047     (void)stack_size;
00048     (void)parameters;
00049     (void)priority;
00050
00051     return SPP_OK;
00052 }
00053
00058 __attribute__((weak)) retval_t OSAL_TaskDelete(osal_task_handle_t task_handle)
00059 {
00060     // Default implementation - just return OK
00061     (void)task_handle;
00062     return SPP_OK;
00063 }
00064
00068 __attribute__((weak)) retval_t OSAL_TaskDelay(uint32_t delay_ms)
00069 {
00070     // Default implementation - busy wait (not recommended for real use)
00071     volatile uint32_t count = delay_ms * 1000;
00072     while (count-- > 0) {
00073         // Busy wait
00074     }
00075     return SPP_OK;
00076 }
00077
00081 __attribute__((weak)) retval_t OSAL_MutexCreate(osal_mutex_handle_t* mutex_handle)
00082 {
00083     if (mutex_handle == NULL) {
00084         return SPP_ERROR_NULL_POINTER;
00085     }
00086
00087     // Default implementation - simulate mutex creation
00088     *mutex_handle = (void*)0x87654321;
00089     return SPP_OK;
00090 }
00091
00095 __attribute__((weak)) retval_t OSAL_MutexDelete(osal_mutex_handle_t mutex_handle)
00096 {
00097     // Default implementation - just return OK
00098     (void)mutex_handle;
00099     return SPP_OK;
00100 }
00101
00105 __attribute__((weak)) retval_t OSAL_MutexTake(osal_mutex_handle_t mutex_handle, uint32_t timeout_ms)
00106 {
00107     if (mutex_handle == NULL) {
00108         return SPP_ERROR_NULL_POINTER;
00109     }
00110
00111     // Default implementation - always succeeds immediately
00112     (void)timeout_ms;
00113     return SPP_OK;
00114 }

```



```

00115
00119 __attribute__((weak)) retval_t OSAL_MutexGive(osal_mutex_handle_t mutex_handle)
00120 {
00121     if (mutex_handle == NULL) {
00122         return SPP_ERROR_NULL_POINTER;
00123     }
00124
00125     // Default implementation - just return OK
00126     return SPP_OK;
00127 }
00128
00132 __attribute__((weak)) retval_t OSAL_QueueCreate(osal_queue_handle_t* queue_handle, uint32_t
queue_length, uint32_t item_size)
00133 {
00134     if (queue_handle == NULL) {
00135         return SPP_ERROR_NULL_POINTER;
00136     }
00137
00138     // Default implementation - simulate queue creation
00139     *queue_handle = (void*)0xABCDEF00;
00140     (void)queue_length;
00141     (void)item_size;
00142     return SPP_OK;
00143 }
00144
00148 __attribute__((weak)) retval_t OSAL_QueueDelete(osal_queue_handle_t queue_handle)
00149 {
00150     // Default implementation - just return OK
00151     (void)queue_handle;
00152     return SPP_OK;
00153 }
00154
00158 __attribute__((weak)) retval_t OSAL_QueueSend(osal_queue_handle_t queue_handle, const void* item,
uint32_t timeout_ms)
00159 {
00160     if (queue_handle == NULL || item == NULL) {
00161         return SPP_ERROR_NULL_POINTER;
00162     }
00163
00164     // Default implementation - always succeeds
00165     (void)timeout_ms;
00166     return SPP_OK;
00167 }
00168
00172 __attribute__((weak)) retval_t OSAL_QueueReceive(osal_queue_handle_t queue_handle, void* item,
uint32_t timeout_ms)
00173 {
00174     if (queue_handle == NULL || item == NULL) {
00175         return SPP_ERROR_NULL_POINTER;
00176     }
00177
00178     // Default implementation - return dummy data
00179     memset(item, 0xAA, sizeof(uint32_t)); // Assume 4-byte items
00180     (void)timeout_ms;
00181     return SPP_OK;
00182 }
00183
00187 __attribute__((weak)) retval_t OSAL_SemaphoreCreate(osal_semaphore_handle_t* semaphore_handle,
uint32_t max_count, uint32_t initial_count)
00188 {
00189     if (semaphore_handle == NULL) {
00190         return SPP_ERROR_NULL_POINTER;
00191     }
00192
00193     // Default implementation - simulate semaphore creation
00194     *semaphore_handle = (void*)0xFEDCBA00;
00195     (void)max_count;
00196     (void)initial_count;
00197     return SPP_OK;
00198 }
00199
00203 __attribute__((weak)) retval_t OSAL_SemaphoreDelete(osal_semaphore_handle_t semaphore_handle)
00204 {
00205     // Default implementation - just return OK
00206     (void)semaphore_handle;
00207     return SPP_OK;
00208 }
00209
00213 __attribute__((weak)) retval_t OSAL_SemaphoreTake(osal_semaphore_handle_t semaphore_handle, uint32_t
timeout_ms)
00214 {
00215     if (semaphore_handle == NULL) {
00216         return SPP_ERROR_NULL_POINTER;
00217     }
00218
00219     // Default implementation - always succeeds
00220     (void)timeout_ms;

```

```

00221     return SPP_OK;
00222 }
00223
00227 __attribute__((weak)) retval_t OSAL_SemaphoreGive(osal_semaphore_handle_t semaphore_handle)
00228 {
00229     if (semaphore_handle == NULL) {
00230         return SPP_ERROR_NULL_POINTER;
00231     }
00232
00233     // Default implementation - just return OK
00234     return SPP_OK;
00235 }
00236
00240 __attribute__((weak)) uint32_t OSAL_GetTickCount(void)
00241 {
00242     // Default implementation - return a dummy value
00243     static uint32_t tick_count = 0;
00244     return ++tick_count;
00245 }
00246
00250 __attribute__((weak)) uint32_t OSAL_MsToTicks(uint32_t ms)
00251 {
00252     // Default implementation - assume 1ms = 1 tick
00253     return ms;
00254 }
00255
00259 __attribute__((weak)) retval_t OSAL_StartScheduler(void)
00260 {
00261     // Default implementation - just return OK (no scheduler to start)
00262     return SPP_OK;
00263 }

```

2.11 osal/osal.h File Reference

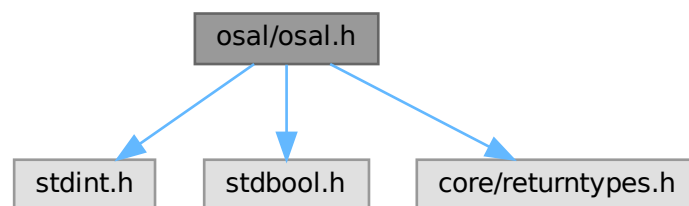
Operating System Abstraction Layer.

```

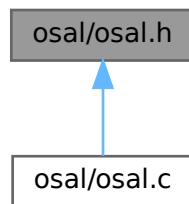
#include <stdint.h>
#include <stdbool.h>
#include "core/returntypes.h"

```

Include dependency graph for osal.h:



This graph shows which files directly or indirectly include this file:



Typedefs

- typedef void * [osal_task_handle_t](#)
OSAL Task handle type.
- typedef void * [osal_mutex_handle_t](#)
OSAL Mutex handle type.
- typedef void * [osal_queue_handle_t](#)
OSAL Queue handle type.
- typedef void * [osal_semaphore_handle_t](#)
OSAL Semaphore handle type.
- typedef void(* [osal_task_function_t](#)) (void *parameters)
Task function pointer type.

Enumerations

- enum [osal_priority_t](#) {
[OSAL_PRIORITY_IDLE](#) = 0 , [OSAL_PRIORITY_LOW](#) = 1 , [OSAL_PRIORITY_NORMAL](#) = 2 ,
[OSAL_PRIORITY_HIGH](#) = 3 ,
[OSAL_PRIORITY_CRITICAL](#) = 4 }
Task priority levels.

Functions

- [retval_t OSAL_Init](#) (void)
Initialize OSAL.
- [retval_t OSAL_Deinit](#) (void)
Deinitialize OSAL.
- [retval_t OSAL_TaskCreate](#) ([osal_task_function_t](#) task_function, const char *name, [uint32_t](#) stack_size, void *parameters, [osal_priority_t](#) priority, [osal_task_handle_t](#) *task_handle)
Create a task.
- [retval_t OSAL_TaskDelete](#) ([osal_task_handle_t](#) task_handle)
Delete a task.
- [retval_t OSAL_TaskDelay](#) ([uint32_t](#) delay_ms)
Delay task execution.
- [retval_t OSAL_MutexCreate](#) ([osal_mutex_handle_t](#) *mutex_handle)

Create a mutex.

- `retval_t OSAL_MutexDelete (osal_mutex_handle_t mutex_handle)`

Delete a mutex.

- `retval_t OSAL_MutexTake (osal_mutex_handle_t mutex_handle, uint32_t timeout_ms)`

Take (lock) a mutex.

- `retval_t OSAL_MutexGive (osal_mutex_handle_t mutex_handle)`

Give (unlock) a mutex.

- `retval_t OSAL_QueueCreate (osal_queue_handle_t *queue_handle, uint32_t queue_length, uint32_t item_size)`

Create a queue.

- `retval_t OSAL_QueueDelete (osal_queue_handle_t queue_handle)`

Delete a queue.

- `retval_t OSAL_QueueSend (osal_queue_handle_t queue_handle, const void *item, uint32_t timeout_ms)`

Send item to queue.

- `retval_t OSAL_QueueReceive (osal_queue_handle_t queue_handle, void *item, uint32_t timeout_ms)`

Receive item from queue.

- `retval_t OSAL_SemaphoreCreate (osal_semaphore_handle_t *semaphore_handle, uint32_t max_count, uint32_t initial_count)`

Create a semaphore.

- `retval_t OSAL_SemaphoreDelete (osal_semaphore_handle_t semaphore_handle)`

Delete a semaphore.

- `retval_t OSAL_SemaphoreTake (osal_semaphore_handle_t semaphore_handle, uint32_t timeout_ms)`

Take a semaphore.

- `retval_t OSAL_SemaphoreGive (osal_semaphore_handle_t semaphore_handle)`

Give a semaphore.

- `uint32_t OSAL_GetTickCount (void)`

Get system tick count.

- `uint32_t OSAL_MsToTicks (uint32_t ms)`

Convert milliseconds to ticks.

- `retval_t OSAL_StartScheduler (void)`

Start the RTOS scheduler.

2.11.1 Detailed Description

Operating System Abstraction Layer.

Version

1.0.0

Date

2024

This file provides the main Operating System Abstraction Layer (OSAL) interface for the Solaris Packet Protocol library.

Definition in file [osal.h](#).

2.11.2 Typedef Documentation

2.11.2.1 osal_mutex_handle_t

```
typedef void* osal_mutex_handle_t
```

OSAL Mutex handle type.

Definition at line 30 of file [osal.h](#).

2.11.2.2 osal_queue_handle_t

```
typedef void* osal_queue_handle_t
```

OSAL Queue handle type.

Definition at line 35 of file [osal.h](#).

2.11.2.3 osal_semaphore_handle_t

```
typedef void* osal_semaphore_handle_t
```

OSAL Semaphore handle type.

Definition at line 40 of file [osal.h](#).

2.11.2.4 osal_task_function_t

```
typedef void(* osal_task_function_t) (void *parameters)
```

Task function pointer type.

Definition at line 45 of file [osal.h](#).

2.11.2.5 osal_task_handle_t

```
typedef void* osal_task_handle_t
```

OSAL Task handle type.

Definition at line 25 of file [osal.h](#).

2.11.3 Enumeration Type Documentation

2.11.3.1 osal_priority_t

```
enum osal_priority_t
```

Task priority levels.

Enumerator

OSAL_PRIORITY_IDLE	
OSAL_PRIORITY_LOW	
OSAL_PRIORITY_NORMAL	
OSAL_PRIORITY_HIGH	
OSAL_PRIORITY_CRITICAL	

Definition at line 50 of file [osal.h](#).

```
00050     {
00051         OSAL_PRIORITY_IDLE = 0,
00052         OSAL_PRIORITY_LOW = 1,
00053         OSAL_PRIORITY_NORMAL = 2,
00054         OSAL_PRIORITY_HIGH = 3,
00055         OSAL_PRIORITY_CRITICAL = 4
00056     } osal_priority_t;
```

2.11.4 Function Documentation

2.11.4.1 OSAL_Deinit()

```
retval_t OSAL_Deinit (
    void )
```

Deinitialize OSAL.

Returns

retval_t SPP_OK on success, error code otherwise

2.11.4.2 OSAL_GetTickCount()

```
uint32_t OSAL_GetTickCount (
    void )
```

Get system tick count.

Returns

uint32_t Current tick count

2.11.4.3 OSAL_Init()

```
retval_t OSAL_Init (
    void )
```

Initialize OSAL.

Returns

retval_t SPP_OK on success, error code otherwise

2.11.4.4 OSAL_MsToTicks()

```
uint32_t OSAL_MsToTicks (
    uint32_t ms )
```

Convert milliseconds to ticks.

Parameters

<i>ms</i>	Milliseconds
-----------	--------------

Returns

uint32_t Equivalent ticks

2.11.4.5 OSAL_MutexCreate()

```
retval_t OSAL_MutexCreate (
    osal_mutex_handle_t * mutex_handle )
```

Create a mutex.

Parameters

<i>mutex_handle</i>	Pointer to store mutex handle
---------------------	-------------------------------

Returns

retval_t SPP_OK on success, error code otherwise

2.11.4.6 OSAL_MutexDelete()

```
retval_t OSAL_MutexDelete (
    osal_mutex_handle_t mutex_handle )
```

Delete a mutex.

Parameters

<i>mutex_handle</i>	Mutex handle
---------------------	--------------

Returns

retval_t SPP_OK on success, error code otherwise

2.11.4.7 OSAL_MutexGive()

```
retval_t OSAL_MutexGive (
    osal_mutex_handle_t mutex_handle )
```

Give (unlock) a mutex.

Parameters

<i>mutex_handle</i>	Mutex handle
---------------------	--------------

Returns

retval_t SPP_OK on success, error code otherwise

2.11.4.8 OSAL_MutexTake()

```
retval_t OSAL_MutexTake (
    osal_mutex_handle_t mutex_handle,
    uint32_t timeout_ms )
```

Take (lock) a mutex.

Parameters

<i>mutex_handle</i>	Mutex handle
<i>timeout_ms</i>	Timeout in milliseconds (0 = no wait, UINT32_MAX = wait forever)

Returns

retval_t SPP_OK on success, error code otherwise

2.11.4.9 OSAL_QueueCreate()

```
retval_t OSAL_QueueCreate (
    osal_queue_handle_t * queue_handle,
    uint32_t queue_length,
    uint32_t item_size )
```

Create a queue.

Parameters

<i>queue_handle</i>	Pointer to store queue handle
<i>queue_length</i>	Maximum number of items in queue
<i>item_size</i>	Size of each item in bytes

Returns

retval_t SPP_OK on success, error code otherwise

2.11.4.10 OSAL_QueueDelete()

```
retval_t OSAL_QueueDelete (
    osal_queue_handle_t queue_handle )
```


Delete a queue.

Parameters

<i>queue_handle</i>	Queue handle
---------------------	--------------

Returns

retval_t SPP_OK on success, error code otherwise

2.11.4.11 OSAL_QueueReceive()

```
retval_t OSAL_QueueReceive (
    osal_queue_handle_t queue_handle,
    void * item,
    uint32_t timeout_ms )
```

Receive item from queue.

Parameters

<i>queue_handle</i>	Queue handle
<i>item</i>	Pointer to buffer for received item
<i>timeout_ms</i>	Timeout in milliseconds

Returns

retval_t SPP_OK on success, error code otherwise

2.11.4.12 OSAL_QueueSend()

```
retval_t OSAL_QueueSend (
    osal_queue_handle_t queue_handle,
    const void * item,
    uint32_t timeout_ms )
```

Send item to queue.

Parameters

<i>queue_handle</i>	Queue handle
<i>item</i>	Pointer to item to send
<i>timeout_ms</i>	Timeout in milliseconds

Returns

retval_t SPP_OK on success, error code otherwise

2.11.4.13 OSAL_SemaphoreCreate()

```
retval_t OSAL_SemaphoreCreate (
    osal_semaphore_handle_t * semaphore_handle,
    uint32_t max_count,
    uint32_t initial_count )
```

Create a semaphore.

Parameters

<i>semaphore_handle</i>	Pointer to store semaphore handle
<i>max_count</i>	Maximum count value
<i>initial_count</i>	Initial count value

Returns

retval_t SPP_OK on success, error code otherwise

2.11.4.14 OSAL_SemaphoreDelete()

```
retval_t OSAL_SemaphoreDelete (
    osal_semaphore_handle_t semaphore_handle )
```

Delete a semaphore.

Parameters

<i>semaphore_handle</i>	Semaphore handle
-------------------------	------------------

Returns

retval_t SPP_OK on success, error code otherwise

2.11.4.15 OSAL_SemaphoreGive()

```
retval_t OSAL_SemaphoreGive (
    osal_semaphore_handle_t semaphore_handle )
```

Give a semaphore.

Parameters

<i>semaphore_handle</i>	Semaphore handle
-------------------------	------------------

Returns

retval_t SPP_OK on success, error code otherwise

2.11.4.16 OSAL_SemaphoreTake()

```
retval_t OSAL_SemaphoreTake (
    osal_semaphore_handle_t semaphore_handle,
    uint32_t timeout_ms )
```

Take a semaphore.

Parameters

<i>semaphore_handle</i>	Semaphore handle
<i>timeout_ms</i>	Timeout in milliseconds

Returns

retval_t SPP_OK on success, error code otherwise

2.11.4.17 OSAL_StartScheduler()

```
retval_t OSAL_StartScheduler (
    void )
```

Start the RTOS scheduler.

Returns

retval_t SPP_OK on success, error code otherwise

2.11.4.18 OSAL_TaskCreate()

```
retval_t OSAL_TaskCreate (
    osal_task_function_t task_function,
    const char * name,
    uint32_t stack_size,
    void * parameters,
    osal_priority_t priority,
    osal_task_handle_t * task_handle )
```

Create a task.

Parameters

<i>task_function</i>	Task function pointer
<i>name</i>	Task name
<i>stack_size</i>	Stack size in bytes
<i>parameters</i>	Task parameters
<i>priority</i>	Task priority
<i>task_handle</i>	Pointer to store task handle

Returns

retval_t SPP_OK on success, error code otherwise

2.11.4.19 OSAL_TaskDelay()

```
retval_t OSAL_TaskDelay (
    uint32_t delay_ms )
```

Delay task execution.

Parameters

<i>delay_ms</i>	Delay in milliseconds
-----------------	-----------------------

Returns

retval_t SPP_OK on success, error code otherwise

2.11.4.20 OSAL_TaskDelete()

```
retval_t OSAL_TaskDelete (
    osal_task_handle_t task_handle )
```

Delete a task.

Parameters

<i>task_handle</i>	Task handle (NULL for current task)
--------------------	-------------------------------------

Returns

retval_t SPP_OK on success, error code otherwise

2.12 osal.h

[Go to the documentation of this file.](#)

```
00001
00011 #ifndef SPP_OSAL_H
00012 #define SPP_OSAL_H
00013
00014 #include <stdint.h>
00015 #include <stdbool.h>
00016 #include "core/returntypes.h"
00017
00018 #ifdef __cplusplus
00019 extern "C" {
00020 #endif
00021
00025 typedef void* osal_task_handle_t;
00026
00030 typedef void* osal_mutex_handle_t;
00031
00035 typedef void* osal_queue_handle_t;
```

```

00036
00040 typedef void* osal_semaphore_handle_t;
00041
00045 typedef void (*osal_task_function_t)(void* parameters);
00046
00050 typedef enum {
00051     OSAL_PRIORITY_IDLE = 0,
00052     OSAL_PRIORITY_LOW = 1,
00053     OSAL_PRIORITY_NORMAL = 2,
00054     OSAL_PRIORITY_HIGH = 3,
00055     OSAL_PRIORITY_CRITICAL = 4
00056 } osal_priority_t;
00057
00063 retval_t OSAL_Init(void);
00064
00070 retval_t OSAL_Deinit(void);
00071
00083 retval_t OSAL_TaskCreate(osal_task_function_t task_function, const char* name,
00084                          uint32_t stack_size, void* parameters, osal_priority_t priority,
00085                          osal_task_handle_t* task_handle);
00086
00093 retval_t OSAL_TaskDelete(osal_task_handle_t task_handle);
00094
00101 retval_t OSAL_TaskDelay(uint32_t delay_ms);
00102
00109 retval_t OSAL_MutexCreate(osal_mutex_handle_t* mutex_handle);
00110
00117 retval_t OSAL_MutexDelete(osal_mutex_handle_t mutex_handle);
00118
00126 retval_t OSAL_MutexTake(osal_mutex_handle_t mutex_handle, uint32_t timeout_ms);
00127
00134 retval_t OSAL_MutexGive(osal_mutex_handle_t mutex_handle);
00135
00144 retval_t OSAL_QueueCreate(osal_queue_handle_t* queue_handle, uint32_t queue_length, uint32_t
    item_size);
00145
00152 retval_t OSAL_QueueDelete(osal_queue_handle_t queue_handle);
00153
00162 retval_t OSAL_QueueSend(osal_queue_handle_t queue_handle, const void* item, uint32_t timeout_ms);
00163
00172 retval_t OSAL_QueueReceive(osal_queue_handle_t queue_handle, void* item, uint32_t timeout_ms);
00173
00182 retval_t OSAL_SemaphoreCreate(osal_semaphore_handle_t* semaphore_handle, uint32_t max_count, uint32_t
    initial_count);
00183
00190 retval_t OSAL_SemaphoreDelete(osal_semaphore_handle_t semaphore_handle);
00191
00199 retval_t OSAL_SemaphoreTake(osal_semaphore_handle_t semaphore_handle, uint32_t timeout_ms);
00200
00207 retval_t OSAL_SemaphoreGive(osal_semaphore_handle_t semaphore_handle);
00208
00214 uint32_t OSAL_GetTickCount(void);
00215
00222 uint32_t OSAL_MsToTicks(uint32_t ms);
00223
00229 retval_t OSAL_StartScheduler(void);
00230
00231 #ifdef __cplusplus
00232 }
00233 #endif
00234
00235 #endif /* SPP_OSAL_H */

```

2.13 osal/queue.c File Reference

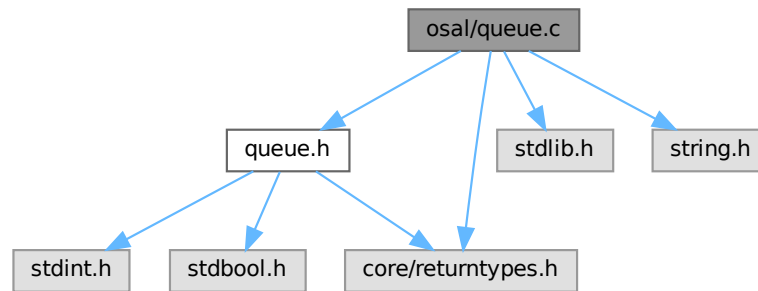
OSAL Queue Management Implementation.

```

#include "queue.h"
#include "core/returntypes.h"
#include <stdlib.h>
#include <string.h>

```

Include dependency graph for queue.c:



Functions

- [__attribute__](#) ((weak))
Default (weak) queue creation.

2.13.1 Detailed Description

OSAL Queue Management Implementation.

Version

1.0.0

Date

2024

This file provides the default (weak) implementation of queue management functions.

Definition in file [queue.c](#).

2.13.2 Function Documentation

2.13.2.1 [__attribute__](#) ()

```
__attribute__ (
    (weak) )
```

Default (weak) queue creation.

Default (weak) reset queue.

Default (weak) check if queue is empty.

Default (weak) check if queue is full.

Default (weak) get queue space.

Default (weak) get queue count.

Default (weak) queue peek.

Default (weak) queue receive from ISR.

Default (weak) queue receive.

Default (weak) queue send from ISR.

Default (weak) queue send.

Default (weak) queue deletion.

Definition at line 18 of file [queue.c](#).

```
00019 {
00020     if (queue_handle == NULL) {
00021         return SPP_ERROR_NULL_POINTER;
00022     }
00023
00024     // Default implementation - simulate queue creation
00025     *queue_handle = (void*)0xABCDEF00;
00026     (void)queue_length;
00027     (void)item_size;
00028     return SPP_OK;
00029 }
```

2.14 queue.c

[Go to the documentation of this file.](#)

```
00001
00010 #include "queue.h"
00011 #include "core/returntypes.h"
00012 #include <stdlib.h>
00013 #include <string.h>
00014
00018 __attribute__((weak)) retval_t OSAL_QueueCreate(osal_queue_handle_t* queue_handle, uint32_t
queue_length, uint32_t item_size)
00019 {
00020     if (queue_handle == NULL) {
00021         return SPP_ERROR_NULL_POINTER;
00022     }
00023
00024     // Default implementation - simulate queue creation
00025     *queue_handle = (void*)0xABCDEF00;
00026     (void)queue_length;
00027     (void)item_size;
00028     return SPP_OK;
00029 }
00030
00034 __attribute__((weak)) retval_t OSAL_QueueDelete(osal_queue_handle_t queue_handle)
00035 {
00036     // Default implementation - just return OK
00037     (void)queue_handle;
00038     return SPP_OK;
00039 }
00040
00044 __attribute__((weak)) retval_t OSAL_QueueSend(osal_queue_handle_t queue_handle, const void* item,
uint32_t timeout_ms)
00045 {
00046     if (queue_handle == NULL || item == NULL) {
00047         return SPP_ERROR_NULL_POINTER;
00048     }
00049
00050     // Default implementation - always succeeds
00051     (void)timeout_ms;
00052     return SPP_OK;
00053 }
00054
```

```

00058 __attribute__((weak)) retval_t OSAL_QueueSendFromISR(osal_queue_handle_t queue_handle, const void*
item, bool* higher_priority_task_woken)
00059 {
00060     if (queue_handle == NULL || item == NULL) {
00061         return SPP_ERROR_NULL_POINTER;
00062     }
00063     // Default implementation - always succeeds
00064     if (higher_priority_task_woken != NULL) {
00065         *higher_priority_task_woken = false;
00066     }
00067     return SPP_OK;
00068 }
00069 }
00070
00074 __attribute__((weak)) retval_t OSAL_QueueReceive(osal_queue_handle_t queue_handle, void* item,
uint32_t timeout_ms)
00075 {
00076     if (queue_handle == NULL || item == NULL) {
00077         return SPP_ERROR_NULL_POINTER;
00078     }
00079     // Default implementation - return dummy data
00080     memset(item, 0xAA, sizeof(uint32_t)); // Assume 4-byte items
00081     (void)timeout_ms;
00082     return SPP_OK;
00083 }
00084 }
00085
00089 __attribute__((weak)) retval_t OSAL_QueueReceiveFromISR(osal_queue_handle_t queue_handle, void* item,
bool* higher_priority_task_woken)
00090 {
00091     if (queue_handle == NULL || item == NULL) {
00092         return SPP_ERROR_NULL_POINTER;
00093     }
00094     // Default implementation - return dummy data
00095     memset(item, 0xBB, sizeof(uint32_t)); // Assume 4-byte items
00096     if (higher_priority_task_woken != NULL) {
00097         *higher_priority_task_woken = false;
00098     }
00099     return SPP_OK;
00100 }
00101 }
00102
00106 __attribute__((weak)) retval_t OSAL_QueuePeek(osal_queue_handle_t queue_handle, void* item, uint32_t
timeout_ms)
00107 {
00108     if (queue_handle == NULL || item == NULL) {
00109         return SPP_ERROR_NULL_POINTER;
00110     }
00111     // Default implementation - return dummy data
00112     memset(item, 0xCC, sizeof(uint32_t)); // Assume 4-byte items
00113     (void)timeout_ms;
00114     return SPP_OK;
00115 }
00116 }
00117
00121 __attribute__((weak)) uint32_t OSAL_QueueGetCount(osal_queue_handle_t queue_handle)
00122 {
00123     // Default implementation - return dummy count
00124     (void)queue_handle;
00125     return 5; // Simulate 5 items in queue
00126 }
00127
00131 __attribute__((weak)) uint32_t OSAL_QueueGetSpace(osal_queue_handle_t queue_handle)
00132 {
00133     // Default implementation - return dummy space
00134     (void)queue_handle;
00135     return 10; // Simulate 10 free spaces
00136 }
00137
00141 __attribute__((weak)) bool OSAL_QueueIsFull(osal_queue_handle_t queue_handle)
00142 {
00143     // Default implementation - never full
00144     (void)queue_handle;
00145     return false;
00146 }
00147
00151 __attribute__((weak)) bool OSAL_QueueIsEmpty(osal_queue_handle_t queue_handle)
00152 {
00153     // Default implementation - never empty
00154     (void)queue_handle;
00155     return false;
00156 }
00157
00161 __attribute__((weak)) retval_t OSAL_QueueReset(osal_queue_handle_t queue_handle)
00162 {
00163     // Default implementation - just return OK
00164     (void)queue_handle;

```

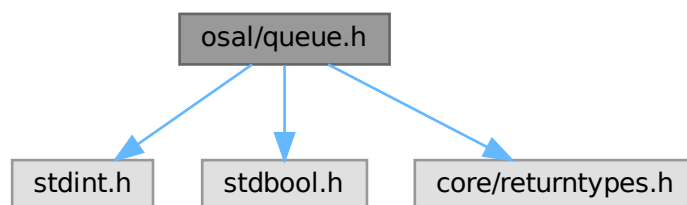


```
00165     return SPP_OK;  
00166 }
```

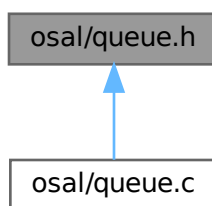
2.15 osal/queue.h File Reference

OSAL Queue Management Interface.

```
#include <stdint.h>  
#include <stdbool.h>  
#include "core/returntypes.h"  
Include dependency graph for queue.h:
```



This graph shows which files directly or indirectly include this file:



Typedefs

- typedef void * [osal_queue_handle_t](#)
OSAL Queue handle type.

Functions

- `retval_t OSAL_QueueCreate (osal_queue_handle_t *queue_handle, uint32_t queue_length, uint32_t item_size)`
Create a queue.
- `retval_t OSAL_QueueDelete (osal_queue_handle_t queue_handle)`
Delete a queue.
- `retval_t OSAL_QueueSend (osal_queue_handle_t queue_handle, const void *item, uint32_t timeout_ms)`
Send item to queue.
- `retval_t OSAL_QueueSendFromISR (osal_queue_handle_t queue_handle, const void *item, bool *higher_priority_task_woken)`
Send item to queue from ISR.
- `retval_t OSAL_QueueReceive (osal_queue_handle_t queue_handle, void *item, uint32_t timeout_ms)`
Receive item from queue.
- `retval_t OSAL_QueueReceiveFromISR (osal_queue_handle_t queue_handle, void *item, bool *higher_priority_task_woken)`
Receive item from queue from ISR.
- `retval_t OSAL_QueuePeek (osal_queue_handle_t queue_handle, void *item, uint32_t timeout_ms)`
Peek at item in queue without removing it.
- `uint32_t OSAL_QueueGetCount (osal_queue_handle_t queue_handle)`
Get number of items in queue.
- `uint32_t OSAL_QueueGetSpace (osal_queue_handle_t queue_handle)`
Get available space in queue.
- `bool OSAL_QueueIsFull (osal_queue_handle_t queue_handle)`
Check if queue is full.
- `bool OSAL_QueueIsEmpty (osal_queue_handle_t queue_handle)`
Check if queue is empty.
- `retval_t OSAL_QueueReset (osal_queue_handle_t queue_handle)`
Reset queue (remove all items)

2.15.1 Detailed Description

OSAL Queue Management Interface.

Version

1.0.0

Date

2024

This file provides the queue management interface for the OSAL layer.

Definition in file [queue.h](#).

2.15.2 Typedef Documentation

2.15.2.1 osal_queue_handle_t

```
typedef void* osal_queue_handle_t
```

OSAL Queue handle type.

Definition at line 24 of file [queue.h](#).

2.15.3 Function Documentation

2.15.3.1 OSAL_QueueCreate()

```
retval_t OSAL_QueueCreate (
    osal_queue_handle_t * queue_handle,
    uint32_t queue_length,
    uint32_t item_size )
```

Create a queue.

Parameters

<i>queue_handle</i>	Pointer to store queue handle
<i>queue_length</i>	Maximum number of items in queue
<i>item_size</i>	Size of each item in bytes

Returns

retval_t SPP_OK on success, error code otherwise

2.15.3.2 OSAL_QueueDelete()

```
retval_t OSAL_QueueDelete (
    osal_queue_handle_t queue_handle )
```

Delete a queue.

Parameters

<i>queue_handle</i>	Queue handle
---------------------	--------------

Returns

retval_t SPP_OK on success, error code otherwise

2.15.3.3 OSAL_QueueGetCount()

```
uint32_t OSAL_QueueGetCount (
    osal_queue_handle_t queue_handle )
```

Get number of items in queue.

Parameters

<i>queue_handle</i>	Queue handle
---------------------	--------------

Returns

uint32_t Number of items in queue

2.15.3.4 OSAL_QueueGetSpace()

```
uint32_t OSAL_QueueGetSpace (
    osal_queue_handle_t queue_handle )
```

Get available space in queue.

Parameters

<i>queue_handle</i>	Queue handle
---------------------	--------------

Returns

uint32_t Available space in queue

2.15.3.5 OSAL_QueueIsEmpty()

```
bool OSAL_QueueIsEmpty (
    osal_queue_handle_t queue_handle )
```

Check if queue is empty.

Parameters

<i>queue_handle</i>	Queue handle
---------------------	--------------

Returns

bool true if queue is empty, false otherwise

2.15.3.6 OSAL_QueueIsFull()

```
bool OSAL_QueueIsFull (
```

```
osal_queue_handle_t queue_handle )
```

Check if queue is full.

Parameters

<i>queue_handle</i>	Queue handle
---------------------	--------------

Returns

bool true if queue is full, false otherwise

2.15.3.7 OSAL_QueuePeek()

```
retval_t OSAL_QueuePeek (
    osal_queue_handle_t queue_handle,
    void * item,
    uint32_t timeout_ms )
```

Peek at item in queue without removing it.

Parameters

<i>queue_handle</i>	Queue handle
<i>item</i>	Pointer to buffer for peeked item
<i>timeout_ms</i>	Timeout in milliseconds

Returns

retval_t SPP_OK on success, error code otherwise

2.15.3.8 OSAL_QueueReceive()

```
retval_t OSAL_QueueReceive (
    osal_queue_handle_t queue_handle,
    void * item,
    uint32_t timeout_ms )
```

Receive item from queue.

Parameters

<i>queue_handle</i>	Queue handle
<i>item</i>	Pointer to buffer for received item
<i>timeout_ms</i>	Timeout in milliseconds

Returns

retval_t SPP_OK on success, error code otherwise

2.15.3.9 OSAL_QueueReceiveFromISR()

```
retval_t OSAL_QueueReceiveFromISR (
    osal_queue_handle_t queue_handle,
    void * item,
    bool * higher_priority_task_woken )
```

Receive item from queue from ISR.

Parameters

<i>queue_handle</i>	Queue handle
<i>item</i>	Pointer to buffer for received item
<i>higher_priority_task_woken</i>	Pointer to flag indicating if higher priority task was woken

Returns

retval_t SPP_OK on success, error code otherwise

2.15.3.10 OSAL_QueueReset()

```
retval_t OSAL_QueueReset (
    osal_queue_handle_t queue_handle )
```

Reset queue (remove all items)

Parameters

<i>queue_handle</i>	Queue handle
---------------------	--------------

Returns

retval_t SPP_OK on success, error code otherwise

2.15.3.11 OSAL_QueueSend()

```
retval_t OSAL_QueueSend (
    osal_queue_handle_t queue_handle,
    const void * item,
    uint32_t timeout_ms )
```

Send item to queue.

Parameters

<i>queue_handle</i>	Queue handle
<i>item</i>	Pointer to item to send
<i>timeout_ms</i>	Timeout in milliseconds

Returns

retval_t SPP_OK on success, error code otherwise

2.15.3.12 OSAL_QueueSendFromISR()

```
retval_t OSAL_QueueSendFromISR (
    osal_queue_handle_t queue_handle,
    const void * item,
    bool * higher_priority_task_woken )
```

Send item to queue from ISR.

Parameters

<i>queue_handle</i>	Queue handle
<i>item</i>	Pointer to item to send
<i>higher_priority_task_woken</i>	Pointer to flag indicating if higher priority task was woken

Returns

retval_t SPP_OK on success, error code otherwise

2.16 queue.h

[Go to the documentation of this file.](#)

```
00001
00010 #ifndef SPP_OSAL_QUEUE_H
00011 #define SPP_OSAL_QUEUE_H
00012
00013 #include <stdint.h>
00014 #include <stdbool.h>
00015 #include "core/returntypes.h"
00016
00017 #ifdef __cplusplus
00018 extern "C" {
00019 #endif
00020
00024 typedef void* osal_queue_handle_t;
00025
00034 retval_t OSAL_QueueCreate(osal_queue_handle_t* queue_handle, uint32_t queue_length, uint32_t
    item_size);
00035
00042 retval_t OSAL_QueueDelete(osal_queue_handle_t queue_handle);
00043
00052 retval_t OSAL_QueueSend(osal_queue_handle_t queue_handle, const void* item, uint32_t timeout_ms);
00053
00062 retval_t OSAL_QueueSendFromISR(osal_queue_handle_t queue_handle, const void* item, bool*
    higher_priority_task_woken);
00063
00072 retval_t OSAL_QueueReceive(osal_queue_handle_t queue_handle, void* item, uint32_t timeout_ms);
00073
```

```

00082 retval_t OSAL_QueueReceiveFromISR(osal_queue_handle_t queue_handle, void* item, bool*
      higher_priority_task_woken);
00083
00092 retval_t OSAL_QueuePeek(osal_queue_handle_t queue_handle, void* item, uint32_t timeout_ms);
00093
00100 uint32_t OSAL_QueueGetCount(osal_queue_handle_t queue_handle);
00101
00108 uint32_t OSAL_QueueGetSpace(osal_queue_handle_t queue_handle);
00109
00116 bool OSAL_QueueIsFull(osal_queue_handle_t queue_handle);
00117
00124 bool OSAL_QueueIsEmpty(osal_queue_handle_t queue_handle);
00125
00132 retval_t OSAL_QueueReset(osal_queue_handle_t queue_handle);
00133
00134 #ifdef __cplusplus
00135 }
00136 #endif
00137
00138 #endif /* SPP_OSAL_QUEUE_H */

```

2.17 osal/semaphore.c File Reference

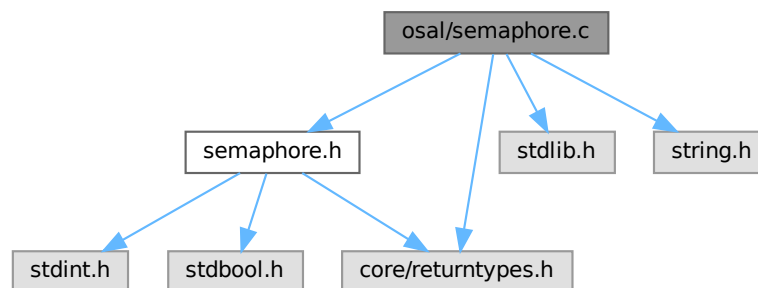
OSAL Semaphore Management Implementation.

```

#include "semaphore.h"
#include "core/returntypes.h"
#include <stdlib.h>
#include <string.h>

```

Include dependency graph for semaphore.c:



Functions

- `__attribute__((weak))`
Default (weak) counting semaphore creation.

2.17.1 Detailed Description

OSAL Semaphore Management Implementation.

Version

1.0.0

Date

2024

This file provides the default (weak) implementation of semaphore management functions.

Definition in file [semaphore.c](#).

2.17.2 Function Documentation

2.17.2.1 __attribute__((weak))

```
__attribute__((weak))
```

Default (weak) counting semaphore creation.

Default (weak) get semaphore count.

Default (weak) semaphore try take.

Default (weak) semaphore give from ISR.

Default (weak) semaphore give.

Default (weak) semaphore take.

Default (weak) semaphore deletion.

Default (weak) binary semaphore creation.

Definition at line 18 of file [semaphore.c](#).

```
00019 {
00020     if (semaphore_handle == NULL) {
00021         return SPP_ERROR_NULL_POINTER;
00022     }
00023
00024     if (initial_count > max_count) {
00025         return SPP_ERROR_INVALID_PARAMETER;
00026     }
00027
00028     // Default implementation - simulate semaphore creation
00029     *semaphore_handle = (void*)0xDEADBEEF;
00030     (void)max_count;
00031     (void)initial_count;
00032     return SPP_OK;
00033 }
```

2.18 semaphore.c

[Go to the documentation of this file.](#)

```

00001
00010 #include "semaphore.h"
00011 #include "core/returntypes.h"
00012 #include <stdlib.h>
00013 #include <string.h>
00014
00018 __attribute__((weak)) retval_t OSAL_SemaphoreCreate(osal_semaphore_handle_t* semaphore_handle,
uint32_t max_count, uint32_t initial_count)
00019 {
00020     if (semaphore_handle == NULL) {
00021         return SPP_ERROR_NULL_POINTER;
00022     }
00023
00024     if (initial_count > max_count) {
00025         return SPP_ERROR_INVALID_PARAMETER;
00026     }
00027
00028     // Default implementation - simulate semaphore creation
00029     *semaphore_handle = (void*)0xDEADBEEF;
00030     (void)max_count;
00031     (void)initial_count;
00032     return SPP_OK;
00033 }
00034
00038 __attribute__((weak)) retval_t OSAL_SemaphoreCreateBinary(osal_semaphore_handle_t* semaphore_handle)
00039 {
00040     if (semaphore_handle == NULL) {
00041         return SPP_ERROR_NULL_POINTER;
00042     }
00043
00044     // Default implementation - simulate binary semaphore creation
00045     *semaphore_handle = (void*)0xBEEFDEAD;
00046     return SPP_OK;
00047 }
00048
00052 __attribute__((weak)) retval_t OSAL_SemaphoreDelete(osal_semaphore_handle_t semaphore_handle)
00053 {
00054     // Default implementation - just return OK
00055     (void)semaphore_handle;
00056     return SPP_OK;
00057 }
00058
00062 __attribute__((weak)) retval_t OSAL_SemaphoreTake(osal_semaphore_handle_t semaphore_handle, uint32_t
timeout_ms)
00063 {
00064     if (semaphore_handle == NULL) {
00065         return SPP_ERROR_NULL_POINTER;
00066     }
00067
00068     // Default implementation - always succeeds immediately
00069     (void)timeout_ms;
00070     return SPP_OK;
00071 }
00072
00076 __attribute__((weak)) retval_t OSAL_SemaphoreGive(osal_semaphore_handle_t semaphore_handle)
00077 {
00078     if (semaphore_handle == NULL) {
00079         return SPP_ERROR_NULL_POINTER;
00080     }
00081
00082     // Default implementation - just return OK
00083     return SPP_OK;
00084 }
00085
00089 __attribute__((weak)) retval_t OSAL_SemaphoreGiveFromISR(osal_semaphore_handle_t semaphore_handle,
bool* higher_priority_task_woken)
00090 {
00091     if (semaphore_handle == NULL) {
00092         return SPP_ERROR_NULL_POINTER;
00093     }
00094
00095     // Default implementation - always succeeds
00096     if (higher_priority_task_woken != NULL) {
00097         *higher_priority_task_woken = false;
00098     }
00099     return SPP_OK;
00100 }
00101
00105 __attribute__((weak)) retval_t OSAL_SemaphoreTryTake(osal_semaphore_handle_t semaphore_handle)
00106 {
00107     if (semaphore_handle == NULL) {
00108         return SPP_ERROR_NULL_POINTER;

```

```

00109     }
00110
00111     // Default implementation - always succeeds
00112     return SPP_OK;
00113 }
00114
00118 __attribute__((weak)) uint32_t OSAL_SemaphoreGetCount(osal_semaphore_handle_t semaphore_handle)
00119 {
00120     // Default implementation - return dummy count
00121     (void)semaphore_handle;
00122     return 3; // Simulate 3 available permits
00123 }

```

2.19 osal/semaphore.h File Reference

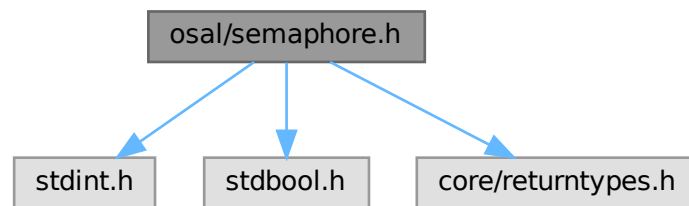
OSAL Semaphore Management Interface.

```

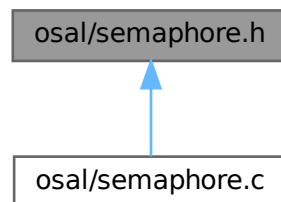
#include <stdint.h>
#include <stdbool.h>
#include "core/returntypes.h"

```

Include dependency graph for semaphore.h:



This graph shows which files directly or indirectly include this file:



Typedefs

- typedef void * [osal_semaphore_handle_t](#)
OSAL Semaphore handle type.

Functions

- `retval_t OSAL_SemaphoreCreate (osal_semaphore_handle_t *semaphore_handle, uint32_t max_count, uint32_t initial_count)`
Create a counting semaphore.
- `retval_t OSAL_SemaphoreCreateBinary (osal_semaphore_handle_t *semaphore_handle)`
Create a binary semaphore.
- `retval_t OSAL_SemaphoreDelete (osal_semaphore_handle_t semaphore_handle)`
Delete a semaphore.
- `retval_t OSAL_SemaphoreTake (osal_semaphore_handle_t semaphore_handle, uint32_t timeout_ms)`
Take a semaphore.
- `retval_t OSAL_SemaphoreGive (osal_semaphore_handle_t semaphore_handle)`
Give a semaphore.
- `retval_t OSAL_SemaphoreGiveFromISR (osal_semaphore_handle_t semaphore_handle, bool *higher_priority_task_woken)`
Give a semaphore from ISR.
- `retval_t OSAL_SemaphoreTryTake (osal_semaphore_handle_t semaphore_handle)`
Try to take a semaphore without blocking.
- `uint32_t OSAL_SemaphoreGetCount (osal_semaphore_handle_t semaphore_handle)`
Get current semaphore count.

2.19.1 Detailed Description

OSAL Semaphore Management Interface.

Version

1.0.0

Date

2024

This file provides the semaphore management interface for the OSAL layer.

Definition in file [semaphore.h](#).

2.19.2 Typedef Documentation

2.19.2.1 `osal_semaphore_handle_t`

```
typedef void* osal_semaphore_handle_t
```

OSAL Semaphore handle type.

Definition at line 24 of file [semaphore.h](#).

2.19.3 Function Documentation

2.19.3.1 `OSAL_SemaphoreCreate()`

```
retval_t OSAL_SemaphoreCreate (
    osal_semaphore_handle_t * semaphore_handle,
    uint32_t max_count,
    uint32_t initial_count )
```

Create a counting semaphore.

Parameters

<i>semaphore_handle</i>	Pointer to store semaphore handle
<i>max_count</i>	Maximum count value
<i>initial_count</i>	Initial count value

Returns

retval_t SPP_OK on success, error code otherwise

2.19.3.2 OSAL_SemaphoreCreateBinary()

```
retval_t OSAL_SemaphoreCreateBinary (
    osal_semaphore_handle_t * semaphore_handle )
```

Create a binary semaphore.

Parameters

<i>semaphore_handle</i>	Pointer to store semaphore handle
-------------------------	-----------------------------------

Returns

retval_t SPP_OK on success, error code otherwise

2.19.3.3 OSAL_SemaphoreDelete()

```
retval_t OSAL_SemaphoreDelete (
    osal_semaphore_handle_t semaphore_handle )
```

Delete a semaphore.

Parameters

<i>semaphore_handle</i>	Semaphore handle
-------------------------	------------------

Returns

retval_t SPP_OK on success, error code otherwise

2.19.3.4 OSAL_SemaphoreGetCount()

```
uint32_t OSAL_SemaphoreGetCount (
    osal_semaphore_handle_t semaphore_handle )
```

Get current semaphore count.

Parameters

<i>semaphore_handle</i>	Semaphore handle
-------------------------	------------------

Returns

uint32_t Current semaphore count

2.19.3.5 OSAL_SemaphoreGive()

```
retval_t OSAL_SemaphoreGive (
    osal_semaphore_handle_t semaphore_handle )
```

Give a semaphore.

Parameters

<i>semaphore_handle</i>	Semaphore handle
-------------------------	------------------

Returns

retval_t SPP_OK on success, error code otherwise

2.19.3.6 OSAL_SemaphoreGiveFromISR()

```
retval_t OSAL_SemaphoreGiveFromISR (
    osal_semaphore_handle_t semaphore_handle,
    bool * higher_priority_task_woken )
```

Give a semaphore from ISR.

Parameters

<i>semaphore_handle</i>	Semaphore handle
<i>higher_priority_task_woken</i>	Pointer to flag indicating if higher priority task was woken

Returns

retval_t SPP_OK on success, error code otherwise

2.19.3.7 OSAL_SemaphoreTake()

```
retval_t OSAL_SemaphoreTake (
    osal_semaphore_handle_t semaphore_handle,
    uint32_t timeout_ms )
```

Take a semaphore.

Parameters

<i>semaphore_handle</i>	Semaphore handle
<i>timeout_ms</i>	Timeout in milliseconds

Returns

retval_t SPP_OK on success, error code otherwise

2.19.3.8 OSAL_SemaphoreTryTake()

```
retval_t OSAL_SemaphoreTryTake (
    osal_semaphore_handle_t semaphore_handle )
```

Try to take a semaphore without blocking.

Parameters

<i>semaphore_handle</i>	Semaphore handle
-------------------------	------------------

Returns

retval_t SPP_OK on success, SPP_ERROR_TIMEOUT if semaphore not available

2.20 semaphore.h

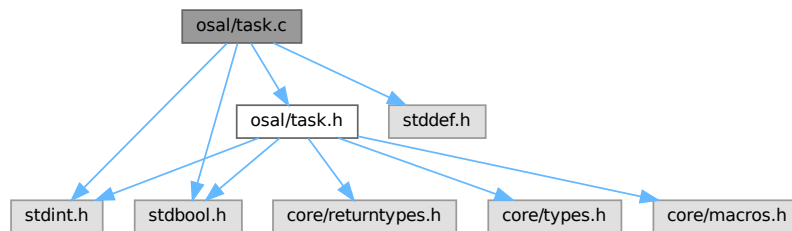
[Go to the documentation of this file.](#)

```
00001
00010 #ifndef SPP_OSAL_SEMAPHORE_H
00011 #define SPP_OSAL_SEMAPHORE_H
00012
00013 #include <stdint.h>
00014 #include <stdbool.h>
00015 #include "core/returntypes.h"
00016
00017 #ifdef __cplusplus
00018 extern "C" {
00019 #endif
00020
00024 typedef void* osal_semaphore_handle_t;
00025
00034 retval_t OSAL_SemaphoreCreate(osal_semaphore_handle_t* semaphore_handle, uint32_t max_count, uint32_t
initial_count);
00035
00042 retval_t OSAL_SemaphoreCreateBinary(osal_semaphore_handle_t* semaphore_handle);
00043
00050 retval_t OSAL_SemaphoreDelete(osal_semaphore_handle_t semaphore_handle);
00051
00059 retval_t OSAL_SemaphoreTake(osal_semaphore_handle_t semaphore_handle, uint32_t timeout_ms);
00060
00067 retval_t OSAL_SemaphoreGive(osal_semaphore_handle_t semaphore_handle);
00068
00076 retval_t OSAL_SemaphoreGiveFromISR(osal_semaphore_handle_t semaphore_handle, bool*
higher_priority_task_woken);
00077
00084 retval_t OSAL_SemaphoreTryTake(osal_semaphore_handle_t semaphore_handle);
00085
00092 uint32_t OSAL_SemaphoreGetCount(osal_semaphore_handle_t semaphore_handle);
00093
00094 #ifdef __cplusplus
00095 }
00096 #endif
00097
00098 #endif /* SPP_OSAL_SEMAPHORE_H */
```

2.21 osal/task.c File Reference

OSAL Task Management Implementation.

```
#include "osal/task.h"
#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
Include dependency graph for task.c:
```



Functions

- [__attribute__](#) ((weak))

2.21.1 Detailed Description

OSAL Task Management Implementation.

Version

1.0.0

Date

2024

This file provides the default (weak) implementation of task management functions.

Definition in file [task.c](#).

2.21.2 Function Documentation

2.21.2.1 [__attribute__](#) ()

```
\_\_attribute\_\_ (
    (weak) )
```

Definition at line 15 of file [task.c](#).

```
00015     {
00016         return NULL;
00017     }
```


2.22 task.c

[Go to the documentation of this file.](#)

```

00001
00010 #include "osal/task.h"
00011 #include <stdint.h>
00012 #include <stdbool.h>
00013 #include <stddef.h>
00014
00015 __attribute__((weak)) void * SPP_OSAL_GetTaskStorage() {
00016     return NULL;
00017 }
00018
00019
00020 __attribute__((weak)) void* SPP_OSAL_TaskCreate(void *p_function, const char *const task_name,
00021     const uint32_t stack_depth, void *const p_custom_data,
00022     spp_uint32_t priority, void * p_storage)
00023 {
00024     {
00025         // If no implementation is defined, then error is returned
00026         return NULL;
00027     }
00028 }
00029 __attribute__((weak)) retval_t SPP_OSAL_TaskDelete(void *p_task){
00030     return SPP_ERROR;
00031 }

```

2.23 osal/task.h File Reference

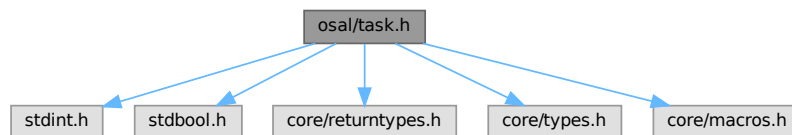
OSAL Task Management Interface.

```

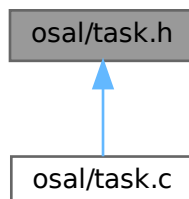
#include <stdint.h>
#include <stdbool.h>
#include "core/returntypes.h"
#include "core/types.h"
#include "core/macros.h"

```

Include dependency graph for task.h:



This graph shows which files directly or indirectly include this file:



Functions

- void * [SPP_OSAL_GetTaskStorage](#) ()
- void * [SPP_OSAL_TaskCreate](#) (void *p_function, const char *const task_name, const uint32_t stack_depth, void *const p_custom_data, spp_uint32_t priority, void *p_storage)
- retval_t [SPP_OSAL_TaskDelete](#) (void *p_task)

2.23.1 Detailed Description

OSAL Task Management Interface.

Version

1.0.0

Date

2024

This file provides the task management interface for the OSAL layer.

Definition in file [task.h](#).

2.23.2 Function Documentation

2.23.2.1 SPP_OSAL_GetTaskStorage()

```
void * SPP_OSAL_GetTaskStorage ( )
```

2.23.2.2 SPP_OSAL_TaskCreate()

```
void * SPP_OSAL_TaskCreate (
    void * p_function,
    const char *const task_name,
    const uint32_t stack_depth,
    void *const p_custom_data,
    spp_uint32_t priority,
    void * p_storage )
```

2.23.2.3 SPP_OSAL_TaskDelete()

```
retval_t SPP_OSAL_TaskDelete (
    void * p_task )
```

2.24 task.h

[Go to the documentation of this file.](#)

```
00001
00010 #ifndef SPP_OSAL_TASK_H
00011 #define SPP_OSAL_TASK_H
00012
00013 #include <stdint.h>
00014 #include <stdbool.h>
00015 #include "core/returntypes.h"
00016 #include "core/types.h"
00017 #include "core/macros.h"
00018
00019 void * SPP_OSAL_GetTaskStorage();
00020
00021
00022 void* SPP_OSAL_TaskCreate(void *p_function, const char *const task_name,
00023                          const uint32_t stack_depth, void *const p_custom_data,
00024                          spp_uint32_t priority, void * p_storage);
00025
00026 retval_t SPP_OSAL_TaskDelete(void *p_task);
00027
00028
00029
00030 #endif /* SPP_OSAL_TASK_H */
```


Index

- [__attribute__](#)
 - [mutex.c, 10](#)
 - [osal.c, 17](#)
 - [queue.c, 32](#)
 - [semaphore.c, 43](#)
 - [spi.c, 4](#)
 - [task.c, 50](#)
- [hal/spi/spi.c, 3, 5](#)
- [hal/spi/spi.h, 6, 8](#)
- [mutex.c](#)
 - [__attribute__, 10](#)
- [mutex.h](#)
 - [osal_mutex_handle_t, 13](#)
 - [OSAL_MUTEX_NORMAL, 13](#)
 - [OSAL_MUTEX_RECURSIVE, 13](#)
 - [osal_mutex_type_t, 13](#)
 - [OSAL_MutexCreate, 13](#)
 - [OSAL_MutexDelete, 13](#)
 - [OSAL_MutexGetHolder, 14](#)
 - [OSAL_MutexGive, 14](#)
 - [OSAL_MutexIsHeldByCurrentTask, 14](#)
 - [OSAL_MutexTake, 15](#)
 - [OSAL_MutexTryTake, 15](#)
- [osal.c](#)
 - [__attribute__, 17](#)
- [osal.h](#)
 - [OSAL_Deinit, 24](#)
 - [OSAL_GetTickCount, 24](#)
 - [OSAL_Init, 24](#)
 - [OSAL_MsToTicks, 24](#)
 - [osal_mutex_handle_t, 23](#)
 - [OSAL_MutexCreate, 25](#)
 - [OSAL_MutexDelete, 25](#)
 - [OSAL_MutexGive, 25](#)
 - [OSAL_MutexTake, 26](#)
 - [OSAL_PRIORITY_CRITICAL, 24](#)
 - [OSAL_PRIORITY_HIGH, 24](#)
 - [OSAL_PRIORITY_IDLE, 24](#)
 - [OSAL_PRIORITY_LOW, 24](#)
 - [OSAL_PRIORITY_NORMAL, 24](#)
 - [osal_priority_t, 23](#)
 - [osal_queue_handle_t, 23](#)
 - [OSAL_QueueCreate, 26](#)
 - [OSAL_QueueDelete, 26](#)
 - [OSAL_QueueReceive, 27](#)
 - [OSAL_QueueSend, 27](#)
 - [osal_semaphore_handle_t, 23](#)
 - [OSAL_SemaphoreCreate, 27](#)
 - [OSAL_SemaphoreDelete, 28](#)
 - [OSAL_SemaphoreGive, 28](#)
 - [OSAL_SemaphoreTake, 28](#)
 - [OSAL_StartScheduler, 29](#)
 - [osal_task_function_t, 23](#)
 - [osal_task_handle_t, 23](#)
 - [OSAL_TaskCreate, 29](#)
 - [OSAL_TaskDelay, 30](#)
 - [OSAL_TaskDelete, 30](#)
- [osal/mutex.c, 9, 10](#)
- [osal/mutex.h, 11, 15](#)
- [osal/osal.c, 16, 18](#)
- [osal/osal.h, 20, 30](#)
- [osal/queue.c, 31, 33](#)
- [osal/queue.h, 35, 41](#)
- [osal/semaphore.c, 42, 44](#)
- [osal/semaphore.h, 45, 49](#)
- [osal/task.c, 50, 51](#)
- [osal/task.h, 51, 53](#)
- [OSAL_Deinit](#)
 - [osal.h, 24](#)
- [OSAL_GetTickCount](#)
 - [osal.h, 24](#)
- [OSAL_Init](#)
 - [osal.h, 24](#)
- [OSAL_MsToTicks](#)
 - [osal.h, 24](#)
- [osal_mutex_handle_t](#)
 - [mutex.h, 13](#)
 - [osal.h, 23](#)
- [OSAL_MUTEX_NORMAL](#)
 - [mutex.h, 13](#)
- [OSAL_MUTEX_RECURSIVE](#)
 - [mutex.h, 13](#)
- [osal_mutex_type_t](#)
 - [mutex.h, 13](#)
- [OSAL_MutexCreate](#)
 - [mutex.h, 13](#)
 - [osal.h, 25](#)
- [OSAL_MutexDelete](#)
 - [mutex.h, 13](#)
 - [osal.h, 25](#)
- [OSAL_MutexGetHolder](#)
 - [mutex.h, 14](#)
- [OSAL_MutexGive](#)
 - [mutex.h, 14](#)
 - [osal.h, 25](#)
- [OSAL_MutexIsHeldByCurrentTask](#)

- mutex.h, [14](#)
- OSAL_MutexTake
 - mutex.h, [15](#)
 - osal.h, [26](#)
- OSAL_MutexTryTake
 - mutex.h, [15](#)
- OSAL_PRIORITY_CRITICAL
 - osal.h, [24](#)
- OSAL_PRIORITY_HIGH
 - osal.h, [24](#)
- OSAL_PRIORITY_IDLE
 - osal.h, [24](#)
- OSAL_PRIORITY_LOW
 - osal.h, [24](#)
- OSAL_PRIORITY_NORMAL
 - osal.h, [24](#)
- osal_priority_t
 - osal.h, [23](#)
- osal_queue_handle_t
 - osal.h, [23](#)
 - queue.h, [37](#)
- OSAL_QueueCreate
 - osal.h, [26](#)
 - queue.h, [37](#)
- OSAL_QueueDelete
 - osal.h, [26](#)
 - queue.h, [37](#)
- OSAL_QueueGetCount
 - queue.h, [37](#)
- OSAL_QueueGetSpace
 - queue.h, [38](#)
- OSAL_QueueIsEmpty
 - queue.h, [38](#)
- OSAL_QueueIsFull
 - queue.h, [38](#)
- OSAL_QueuePeek
 - queue.h, [39](#)
- OSAL_QueueReceive
 - osal.h, [27](#)
 - queue.h, [39](#)
- OSAL_QueueReceiveFromISR
 - queue.h, [40](#)
- OSAL_QueueReset
 - queue.h, [40](#)
- OSAL_QueueSend
 - osal.h, [27](#)
 - queue.h, [40](#)
- OSAL_QueueSendFromISR
 - queue.h, [41](#)
- osal_semaphore_handle_t
 - osal.h, [23](#)
 - semaphore.h, [46](#)
- OSAL_SemaphoreCreate
 - osal.h, [27](#)
 - semaphore.h, [46](#)
- OSAL_SemaphoreCreateBinary
 - semaphore.h, [47](#)
- OSAL_SemaphoreDelete

- osal.h, [28](#)
- semaphore.h, [47](#)
- OSAL_SemaphoreGetCount
 - semaphore.h, [47](#)
- OSAL_SemaphoreGive
 - osal.h, [28](#)
 - semaphore.h, [48](#)
- OSAL_SemaphoreGiveFromISR
 - semaphore.h, [48](#)
- OSAL_SemaphoreTake
 - osal.h, [28](#)
 - semaphore.h, [48](#)
- OSAL_SemaphoreTryTake
 - semaphore.h, [49](#)
- OSAL_StartScheduler
 - osal.h, [29](#)
- osal_task_function_t
 - osal.h, [23](#)
- osal_task_handle_t
 - osal.h, [23](#)
- OSAL_TaskCreate
 - osal.h, [29](#)
- OSAL_TaskDelay
 - osal.h, [30](#)
- OSAL_TaskDelete
 - osal.h, [30](#)
- queue.c
 - __attribute__, [32](#)
- queue.h
 - osal_queue_handle_t, [37](#)
 - OSAL_QueueCreate, [37](#)
 - OSAL_QueueDelete, [37](#)
 - OSAL_QueueGetCount, [37](#)
 - OSAL_QueueGetSpace, [38](#)
 - OSAL_QueueIsEmpty, [38](#)
 - OSAL_QueueIsFull, [38](#)
 - OSAL_QueuePeek, [39](#)
 - OSAL_QueueReceive, [39](#)
 - OSAL_QueueReceiveFromISR, [40](#)
 - OSAL_QueueReset, [40](#)
 - OSAL_QueueSend, [40](#)
 - OSAL_QueueSendFromISR, [41](#)
- semaphore.c
 - __attribute__, [43](#)
- semaphore.h
 - osal_semaphore_handle_t, [46](#)
 - OSAL_SemaphoreCreate, [46](#)
 - OSAL_SemaphoreCreateBinary, [47](#)
 - OSAL_SemaphoreDelete, [47](#)
 - OSAL_SemaphoreGetCount, [47](#)
 - OSAL_SemaphoreGive, [48](#)
 - OSAL_SemaphoreGiveFromISR, [48](#)
 - OSAL_SemaphoreTake, [48](#)
 - OSAL_SemaphoreTryTake, [49](#)
- spi.c
 - __attribute__, [4](#)
- spi.h

- SPP_HAL_SPI_BusInit, [7](#)
- SPP_HAL_SPI_DeviceInit, [7](#)
- SPP_HAL_SPI_GetHandler, [7](#)
- SPP_HAL_SPI_Transmit, [8](#)
- SPP_HAL_SPI_BusInit
 - spi.h, [7](#)
- SPP_HAL_SPI_DeviceInit
 - spi.h, [7](#)
- SPP_HAL_SPI_GetHandler
 - spi.h, [7](#)
- SPP_HAL_SPI_Transmit
 - spi.h, [8](#)
- SPP_OSAL_GetTaskStorage
 - task.h, [52](#)
- SPP_OSAL_TaskCreate
 - task.h, [52](#)
- SPP_OSAL_TaskDelete
 - task.h, [52](#)
- task.c
 - __attribute__, [50](#)
- task.h
 - SPP_OSAL_GetTaskStorage, [52](#)
 - SPP_OSAL_TaskCreate, [52](#)
 - SPP_OSAL_TaskDelete, [52](#)