Solaris Packet Protocol

0.1

Generated by Doxygen 1.9.8

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

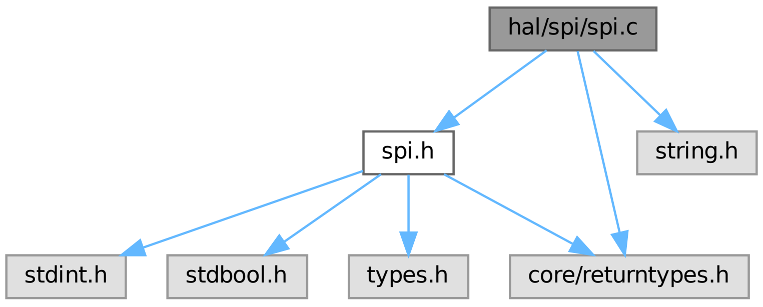
Chapter 2

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 asssigner (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.

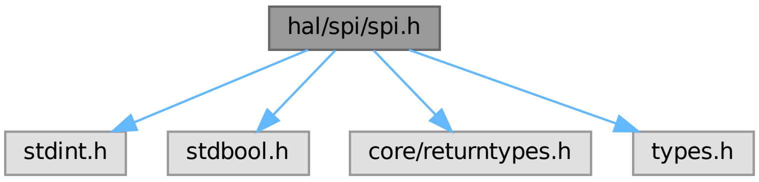
```
00001
00011 #include "spi.h"
00012 #include "core/returntypes.h"
00013 #include <string.h>
00014
00015
00019 __attribute__((weak)) retval_t SPP_HAL_SPI_BusInit() {
00020     return SPP_ERROR;
00021 }
00022
00023
00027 __attribute__((weak)) void* SPP_HAL_SPI_GetHandler() {
00028     return NULL;
00029 }
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

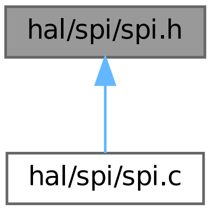
SPI Hardware Abstraction Layer.

```
#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 pheripheral ([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

p_handler	Void pointer to pheripheral 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 pheripheral ([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

handle	Pointer to SPI handle
data_to_send	Pointer to the data to transmit
data_to_recieve	Pointer to where the data recieved will be stored
length	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.

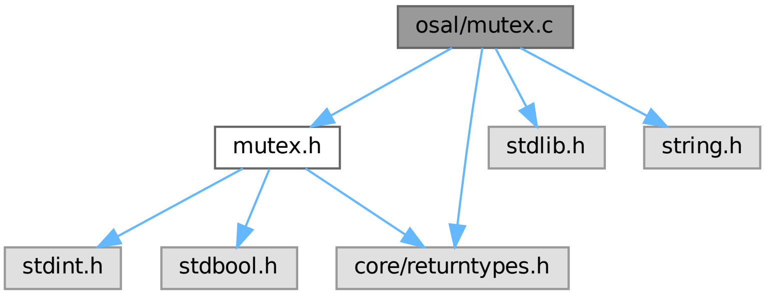
```

#ifndef SPP_HAL_SPI_H
#define SPP_HAL_SPI_H

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

#ifdef __cplusplus
extern "C" {
#endif

retval_t SPP_HAL_SPI_BusInit(void);


void* SPP_HAL_SPI_GetHandler(void);


retval_t SPP_HAL_SPI_DeviceInit(void* p_handler);

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


#ifdef __cplusplus
}
#endif

#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;
```

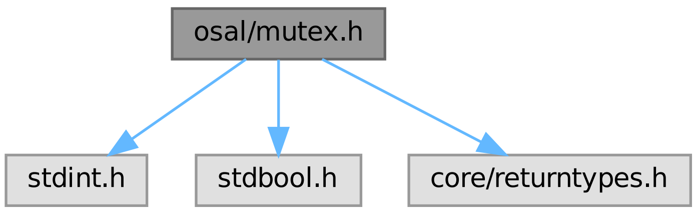
```
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_MutexGive(osal_mutex_handle_t mutex_handle)
{
    if (mutex_handle == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - just return OK
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_MutexTryTake(osal_mutex_handle_t mutex_handle)
{
    if (mutex_handle == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - always succeeds
    return SPP_OK;
}

__attribute__((weak)) void* OSAL_MutexGetHolder(osal_mutex_handle_t mutex_handle)
{
    // Default implementation - return dummy task handle
    (void)mutex_handle;
    return (void*)0x11111111;
}

__attribute__((weak)) bool OSAL_MutexIsHeldByCurrentTask(osal_mutex_handle_t mutex_handle)
{
    // Default implementation - always return true
    (void)mutex_handle;
    return true;
}
```

2.7 osal/mutex.h File Reference

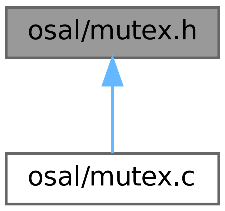
OSAL Mutex Management Interface.

```
#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

mutex_handle	Pointer to store mutex handle
type	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

mutex_handle	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

mutex_handle	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

mutex_handle	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

mutex_handle	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

mutex_handle	Mutex handle
timeout_ms	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

mutex_handle	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
```

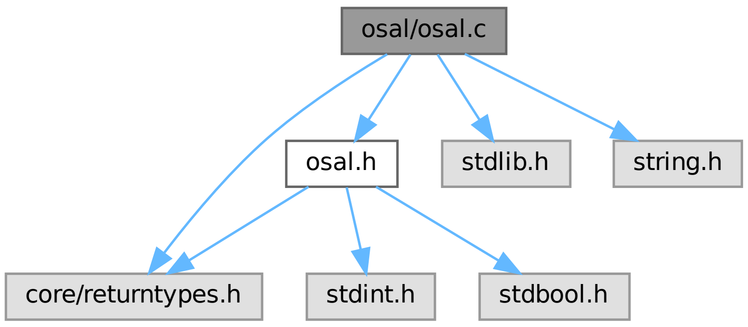
```
#ifndef SPP_OSAL_MUTEX_H
#define SPP_OSAL_MUTEX_H

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

#ifdef __cplusplus
extern "C" {
#endif

typedef void* osal_mutex_handle_t;

typedef enum {
    OSAL_MUTEX_NORMAL = 0,
    OSAL_MUTEX_RECURSIVE = 1
} osal_mutex_type_t;

retval_t OSAL_MutexCreate(osal_mutex_handle_t* mutex_handle, osal_mutex_type_t type);

retval_t OSAL_MutexDelete(osal_mutex_handle_t mutex_handle);

retval_t OSAL_MutexTake(osal_mutex_handle_t mutex_handle, uint32_t timeout_ms);

retval_t OSAL_MutexGive(osal_mutex_handle_t mutex_handle);

retval_t OSAL_MutexTryTake(osal_mutex_handle_t mutex_handle);

void* OSAL_MutexGetHolder(osal_mutex_handle_t mutex_handle);

bool OSAL_MutexIsHeldByCurrentTask(osal_mutex_handle_t mutex_handle);

#ifdef __cplusplus
}
#endif

#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.

```c

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

__attribute__((weak)) retval_t OSAL_Init(void)
{
    // Default implementation - just return OK
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_Deinit(void)
{
    // Default implementation - just return OK
    return SPP_OK;
}

__attribute__((weak)) 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)
{
    if (task_function == NULL || task_handle == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - simulate task creation
    *task_handle = (void*)0x12345678;
    (void)name;
    (void)stack_size;
    (void)parameters;
    (void)priority;

    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_TaskDelete(osal_task_handle_t task_handle)
{
    // Default implementation - just return OK
    (void)task_handle;
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_TaskDelay(uint32_t delay_ms)
{
    // Default implementation - busy wait (not recommended for real use)
    volatile uint32_t count = delay_ms * 1000;
    while (count--) {
        // Busy wait
    }
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_MutexCreate(osal_mutex_handle_t* mutex_handle)
{
    if (mutex_handle == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - simulate mutex creation
    *mutex_handle = (void*)0x87654321;
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_MutexDelete(osal_mutex_handle_t mutex_handle)
{
    // Default implementation - just return OK
    (void)mutex_handle;
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_MutexTake(osal_mutex_handle_t mutex_handle, uint32_t timeout_ms)
{
    if (mutex_handle == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - always succeeds immediately
    (void)timeout_ms;
    return SPP_OK;
}
```

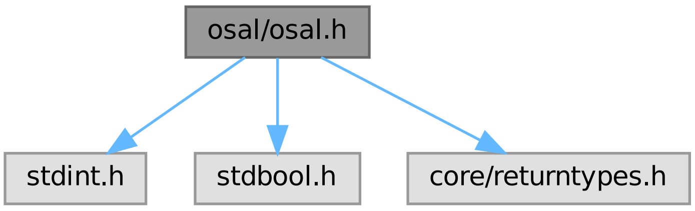
```
__attribute__((weak)) retval_t OSAL_MutexGive(osal_mutex_handle_t mutex_handle)
{
    if (mutex_handle == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - just return OK
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_QueueCreate(osal_queue_handle_t* queue_handle, uint32_t
queue_length, uint32_t item_size)
{
    if (queue_handle == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - simulate queue creation
    *queue_handle = (void*)0xABCDEF00;
    (void)queue_length;
    (void)item_size;
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_QueueDelete(osal_queue_handle_t queue_handle)
{
    // Default implementation - just return OK
    (void)queue_handle;
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_QueueSend(osal_queue_handle_t queue_handle, const void* item,
uint32_t timeout_ms)
{
    if (queue_handle == NULL || item == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - always succeeds
    (void)timeout_ms;
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_QueueReceive(osal_queue_handle_t queue_handle, void* item,
uint32_t timeout_ms)
{
    if (queue_handle == NULL || item == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - return dummy data
    memset(item, 0xAA, sizeof(uint32_t)); // Assume 4-byte items
    (void)timeout_ms;
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_SemaphoreCreate(osal_semaphore_handle_t* semaphore_handle,
uint32_t max_count, uint32_t initial_count)
{
    if (semaphore_handle == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - simulate semaphore creation
    *semaphore_handle = (void*)0xFEDCBA00;
    (void)max_count;
    (void)initial_count;
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_SemaphoreDelete(osal_semaphore_handle_t semaphore_handle)
{
    // Default implementation - just return OK
    (void)semaphore_handle;
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_SemaphoreTake(osal_semaphore_handle_t semaphore_handle, uint32_t
timeout_ms)
{
    if (semaphore_handle == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - always succeeds
    (void)timeout_ms;
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_SemaphoreGive(osal_semaphore_handle_t semaphore_handle)
{
    if (semaphore_handle == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - just return OK
    return SPP_OK;
}

__attribute__((weak)) uint32_t OSAL_GetTickCount(void)
{
    // Default implementation - return a dummy value
    static uint32_t tick_count = 0;
    return ++tick_count;
}

__attribute__((weak)) uint32_t OSAL_MsToTicks(uint32_t ms)
{
    // Default implementation - assume 1ms = 1 tick
    return ms;
}

__attribute__((weak)) retval_t OSAL_StartScheduler(void)
{
    // Default implementation - just return OK (no scheduler to start)
    return SPP_OK;
}
```

2.11 osal/osal.h File Reference

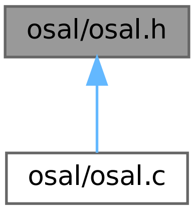
Operating System Abstraction Layer.

```
#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

ms	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

mutex_handle	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

mutex_handle	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

mutex_handle	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

mutex_handle	Mutex handle
timeout_ms	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

queue_handle	Pointer to store queue handle
queue_length	Maximum number of items in queue
item_size	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

queue_handle	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

queue_handle	Queue handle
item	Pointer to buffer for received item
timeout_ms	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

queue_handle	Queue handle
item	Pointer to item to send
timeout_ms	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

semaphore_handle	Pointer to store semaphore handle
max_count	Maximum count value
initial_count	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

semaphore_handle	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

semaphore_handle	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

semaphore_handle	Semaphore handle
timeout_ms	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

task_function	Task function pointer
name	Task name
stack_size	Stack size in bytes
parameters	Task parameters
priority	Task priority
task_handle	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

delay_ms	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

task_handle	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.

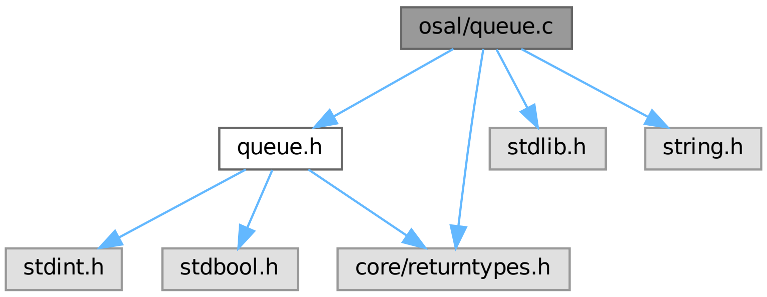
```

#ifndef SPP_OSAL_H
#define SPP_OSAL_H

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

#ifdef __cplusplus
extern "C" {
#endif

typedef void* osal_task_handle_t;

typedef void* osal_mutex_handle_t;

typedef void* osal_queue_handle_t;

typedef void* osal_semaphore_handle_t;

typedef void (*osal_task_function_t)(void* parameters);

typedef enum {
    OSAL_PRIORITY_IDLE = 0,
    OSAL_PRIORITY_LOW = 1,
    OSAL_PRIORITY_NORMAL = 2,
    OSAL_PRIORITY_HIGH = 3,
    OSAL_PRIORITY_CRITICAL = 4
} osal_priority_t;

retval_t OSAL_Init(void);

retval_t OSAL_Deinit(void);

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);

retval_t OSAL_TaskDelete(osal_task_handle_t task_handle);

retval_t OSAL_TaskDelay(uint32_t delay_ms);

retval_t OSAL_MutexCreate(osal_mutex_handle_t* mutex_handle);

retval_t OSAL_MutexDelete(osal_mutex_handle_t mutex_handle);

retval_t OSAL_MutexTake(osal_mutex_handle_t mutex_handle, uint32_t timeout_ms);

retval_t OSAL_MutexGive(osal_mutex_handle_t mutex_handle);

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

retval_t OSAL_QueueDelete(osal_queue_handle_t queue_handle);

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

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

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

retval_t OSAL_SemaphoreDelete(osal_semaphore_handle_t semaphore_handle);

retval_t OSAL_SemaphoreTake(osal_semaphore_handle_t semaphore_handle, uint32_t timeout_ms);

retval_t OSAL_SemaphoreGive(osal_semaphore_handle_t semaphore_handle);

uint32_t OSAL_GetTickCount(void);

uint32_t OSAL_MsToTicks(uint32_t ms);

retval_t OSAL_StartScheduler(void);

#ifdef __cplusplus
}
#endif

#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.

```
{
    if (queue_handle == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - simulate queue creation
    *queue_handle = (void*)0xABCDEF00;
    (void)queue_length;
    (void)item_size;
    return SPP_OK;
}
```

2.14 queue.c

Go to the documentation of this file.

```

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

__attribute__((weak)) retval_t OSAL_QueueCreate(osal_queue_handle_t* queue_handle, uint32_t
      queue_length, uint32_t item_size)
{
    if (queue_handle == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - simulate queue creation
    *queue_handle = (void*)0xABCDEF00;
    (void)queue_length;
    (void)item_size;
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_QueueDelete(osal_queue_handle_t queue_handle)
{
    // Default implementation - just return OK
    (void)queue_handle;
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_QueueSend(osal_queue_handle_t queue_handle, const void* item,
      uint32_t timeout_ms)
{
    if (queue_handle == NULL || item == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - always succeeds
    (void)timeout_ms;
    return SPP_OK;
}

```

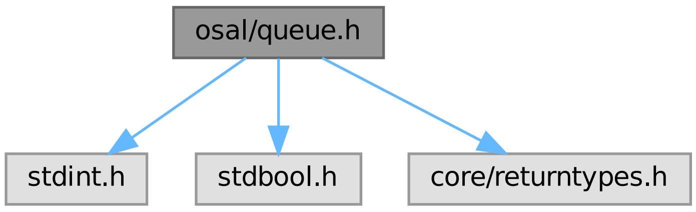
```
__attribute__((weak)) retval_t OSAL_QueueSendFromISR(osal_queue_handle_t queue_handle, const void* item, bool* higher_priority_task_woken)
{
    if (queue_handle == NULL || item == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - always succeeds
    if (higher_priority_task_woken != NULL) {
        *higher_priority_task_woken = false;
    }
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_QueueReceive(osal_queue_handle_t queue_handle, void* item, uint32_t timeout_ms)
{
    if (queue_handle == NULL || item == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - return dummy data
    memset(item, 0xAA, sizeof(uint32_t)); // Assume 4-byte items
    (void)timeout_ms;
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_QueueReceiveFromISR(osal_queue_handle_t queue_handle, void* item, bool* higher_priority_task_woken)
{
    if (queue_handle == NULL || item == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - return dummy data
    memset(item, 0xBB, sizeof(uint32_t)); // Assume 4-byte items
    if (higher_priority_task_woken != NULL) {
        *higher_priority_task_woken = false;
    }
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_QueuePeek(osal_queue_handle_t queue_handle, void* item, uint32_t timeout_ms)
{
    if (queue_handle == NULL || item == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - return dummy data
    memset(item, 0xCC, sizeof(uint32_t)); // Assume 4-byte items
    (void)timeout_ms;
    return SPP_OK;
}

__attribute__((weak)) uint32_t OSAL_QueueGetCount(osal_queue_handle_t queue_handle)
{
    // Default implementation - return dummy count
    (void)queue_handle;
    return 5; // Simulate 5 items in queue
}

__attribute__((weak)) uint32_t OSAL_QueueGetSpace(osal_queue_handle_t queue_handle)
{
    // Default implementation - return dummy space
    (void)queue_handle;
    return 10; // Simulate 10 free spaces
}

__attribute__((weak)) bool OSAL_QueueIsFull(osal_queue_handle_t queue_handle)
{
    // Default implementation - never full
    (void)queue_handle;
    return false;
}

__attribute__((weak)) bool OSAL_QueueIsEmpty(osal_queue_handle_t queue_handle)
{
    // Default implementation - never empty
    (void)queue_handle;
    return false;
}

__attribute__((weak)) retval_t OSAL_QueueReset(osal_queue_handle_t queue_handle)
{
    // Default implementation - just return OK
    (void)queue_handle;
    return SPP_OK;
}
```

2.15 osal/queue.h File Reference

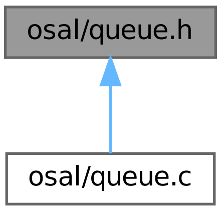
OSAL Queue Management Interface.

```
#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

queue_handle	Pointer to store queue handle
queue_length	Maximum number of items in queue
item_size	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

queue_handle	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

queue_handle	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

queue_handle	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

queue_handle	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

queue_handle	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

queue_handle	Queue handle
item	Pointer to buffer for peeked item
timeout_ms	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

queue_handle	Queue handle
item	Pointer to buffer for received item
timeout_ms	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

queue_handle	Queue handle
item	Pointer to buffer for received item
higher_priority_task_woken	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

queue_handle	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

queue_handle	Queue handle
item	Pointer to item to send
timeout_ms	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

queue_handle	Queue handle
item	Pointer to item to send
higher_priority_task_woken	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.

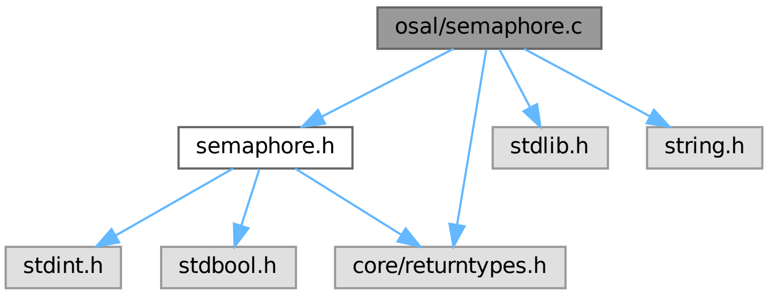
```

#ifndef SPP_OSAL_QUEUE_H
#define SPP_OSAL_QUEUE_H

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

#ifdef __cplusplus
extern "C" {
#endif

typedef void* osal_queue_handle_t;

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

retval_t OSAL_QueueDelete(osal_queue_handle_t queue_handle);

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

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

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

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

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

uint32_t OSAL_QueueGetCount(osal_queue_handle_t queue_handle);

uint32_t OSAL_QueueGetSpace(osal_queue_handle_t queue_handle);

bool OSAL_QueueIsFull(osal_queue_handle_t queue_handle);

bool OSAL_QueueIsEmpty(osal_queue_handle_t queue_handle);

retval_t OSAL_QueueReset(osal_queue_handle_t queue_handle);

#ifdef __cplusplus
}
#endif

#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__()

```
__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.

```
{
    if (semaphore_handle == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    if (initial_count > max_count) {
        return SPP_ERROR_INVALID_PARAMETER;
    }

    // Default implementation - simulate semaphore creation
    *semaphore_handle = (void*)0xDEADBEEF;
    (void)max_count;
    (void)initial_count;
    return SPP_OK;
}
```

2.18 semaphore.c

Go to the documentation of this file.

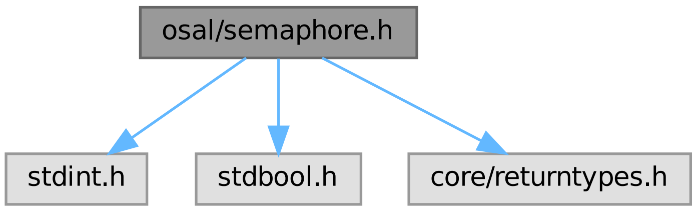
```

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

__attribute__((weak)) retval_t OSAL_SemaphoreCreate(osal_semaphore_handle_t* semaphore_handle,
uint32_t max_count, uint32_t initial_count)
{
    if (semaphore_handle == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    if (initial_count > max_count) {
        return SPP_ERROR_INVALID_PARAMETER;
    }

    // Default implementation - simulate semaphore creation
    *semaphore_handle = (void*)0xDEADBEEF;
    (void)max_count;
    (void)initial_count;
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_SemaphoreCreateBinary(osal_semaphore_handle_t* semaphore_handle)
{
    if (semaphore_handle == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - simulate binary semaphore creation
    *semaphore_handle = (void*)0xBEEFDEAD;
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_SemaphoreDelete(osal_semaphore_handle_t semaphore_handle)
{
    // Default implementation - just return OK
    (void)semaphore_handle;
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_SemaphoreTake(osal_semaphore_handle_t semaphore_handle, uint32_t
timeout_ms)
{
    if (semaphore_handle == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - always succeeds immediately
    (void)timeout_ms;
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_SemaphoreGive(osal_semaphore_handle_t semaphore_handle)
{
    if (semaphore_handle == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - just return OK
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_SemaphoreGiveFromISR(osal_semaphore_handle_t semaphore_handle,
bool* higher_priority_task_woken)
{
    if (semaphore_handle == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - always succeeds
    if (higher_priority_task_woken != NULL) {
        *higher_priority_task_woken = false;
    }
    return SPP_OK;
}

__attribute__((weak)) retval_t OSAL_SemaphoreTryTake(osal_semaphore_handle_t semaphore_handle)
{
    if (semaphore_handle == NULL) {
        return SPP_ERROR_NULL_POINTER;
    }

    // Default implementation - always succeeds
    return SPP_OK;
}

__attribute__((weak)) uint32_t OSAL_SemaphoreGetCount(osal_semaphore_handle_t semaphore_handle)
{
    // Default implementation - return dummy count
    (void)semaphore_handle;
    return 3; // Simulate 3 available permits
}
```

2.19 osal/semaphore.h File Reference

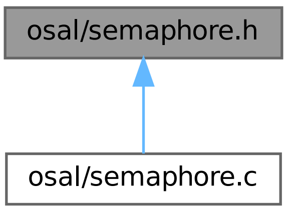
OSAL Semaphore Management Interface.

```
#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

semaphore_handle	Pointer to store semaphore handle
max_count	Maximum count value
initial_count	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

semaphore_handle	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

semaphore_handle	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

semaphore_handle	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

semaphore_handle	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

semaphore_handle	Semaphore handle
higher_priority_task_woken	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

semaphore_handle	Semaphore handle
timeout_ms	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

semaphore_handle	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.

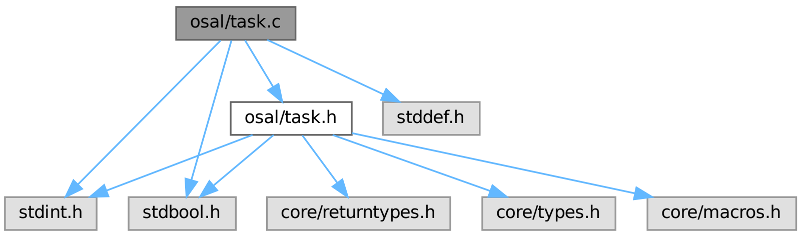
```

#ifndef SPP_OSAL_SEMAPHORE_H
#define SPP_OSAL_SEMAPHORE_H

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

#ifdef __cplusplus
extern "C" {
#endif

typedef void* osal_semaphore_handle_t;

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

retval_t OSAL_SemaphoreCreateBinary(osal_semaphore_handle_t* semaphore_handle);

retval_t OSAL_SemaphoreDelete(osal_semaphore_handle_t semaphore_handle);

retval_t OSAL_SemaphoreTake(osal_semaphore_handle_t semaphore_handle, uint32_t timeout_ms);

retval_t OSAL_SemaphoreGive(osal_semaphore_handle_t semaphore_handle);

retval_t OSAL_SemaphoreGiveFromISR(osal_semaphore_handle_t semaphore_handle, bool* higher_priority_task_woken);

retval_t OSAL_SemaphoreTryTake(osal_semaphore_handle_t semaphore_handle);

uint32_t OSAL_SemaphoreGetCount(osal_semaphore_handle_t semaphore_handle);

#ifdef __cplusplus
}
#endif

#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.

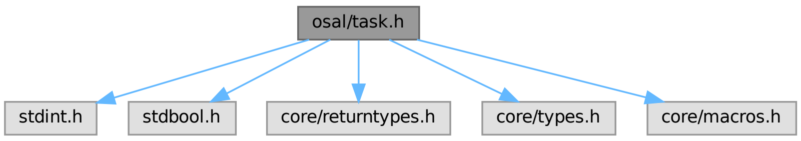
```
#include "osal/task.h"
#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>

__attribute__((weak))  void * SPP_OSAL_GetTaskStorage(){
    return NULL;
}


__attribute__((weak)) 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)

{
    // If no implementation is defined, then error is returned
    return NULL;
}

__attribute__((weak)) retval_t SPP_OSAL_TaskDelete(void *p_task){
    return SPP_ERROR;
}
```

2.23 osal/task.h File Reference

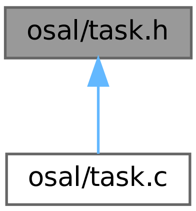
OSAL Task Management Interface.

```
#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.

```
#ifndef SPP_OSAL_TASK_H
#define SPP_OSAL_TASK_H

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

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);



#endif /* SPP_OSAL_TASK_H */
```

Index