Contents

1. Services
   Introduction ................................................................. 6
   MUST ............................................................................. 7
   Recompilation .................................................................. 7
   Return Value in init ......................................................... 7
   New Version Field in wsio_drv_info_t ................................. 7
   dma_sync_IO renamed ....................................................... 8
   I/O Nodes Within the I/O Tree ........................................... 8
   Highly Recommended ....................................................... 9
   Enhanced WSIO Functions ............................................... 9
   Removed Macros ............................................................ 11
   WSIO Memory Allocator ................................................... 12
   Event Handling ............................................................. 14

2. Manpages
   New Services ............................................................... 16
1 Services
Introduction

The WSIO interfaces have changed, and new interfaces have been added to WSIO in HP-UX 11i. These changes are described in this guide.

This document is organized into three sections:

- MUST – describes what a driver must change to function properly
- Highly Recommended – items available to take advantage of the new WSIO interfaces
- Event Handling – describes a major new WSIO function.
MUST

This section describes mandatory changes to structures, headers, and other areas to guarantee HP-UX 11i driver performance. Function and macro deletions and renames are also listed.

Recompilation

There are some changes in the header files. One of them is in the struct `isc_table_type` in `io.h`. These changes affect the offsets in some data structures, so drivers must be recompiled. If an 11.0 driver is installed without recompilation, the system panics because the wrong offsets are accessed.

Return Value in init

A driver's init routine returns either `WSIO_OK(0)` or `WSIO_ERROR(-1)`. WSIO is now checking for the return value. If `WSIO_OK` is not returned, the driver is not allowed to claim a card.

New Version Field in `wsio_drv_info_t`

WSIO has added the field “unsigned int driver_version” to the `wsio_drv_info_t` structure in `wsio.h`:

```c
typedef struct wsio_drv_info{
    drv_info_t *drv_info;
    drv_ops_t *drv_ops;
    wsio_drv_data_t *drv_data;
    unsigned int *driver_version; /* New field */
} wsio_drv_info_t;
```

This field is to version stamp the structure for future use. Set it with a define `WSIO_DRV_CURRENT_VERSION` (in `wsio.h`):

```c
wsio_drv_info_t my_drv_info = {
    &my_info,
    &my_ops,
    &my_data,
    WSIO_DRV_CURRENT_VERSION,
};
```
MUST

**dma_sync_IO renamed**

The macro `dma_sync_IO` has been renamed `DMA_SYNC_IO` to be consistent with the common practice of capitalizing macro names.

**I/O Nodes Within the I/O Tree**

The I/O tree, nodes, and its internal components are private to GIO and WSIO. A driver should not access them. If a driver has been entering the node directly to retrieve information, it is a bug and won't work any more because some definitions have changed.
Highly Recommended

New I/O, Memory Allocation, DMA, and 32-bit DMA services are described in this chapter. The advantages of these new services and the limitations of older services are detailed. Macros that will be deprecated soon are listed.

Enhanced WSIO Functions

WSIO has made available many new functions to provide a driver more flexibility with services in areas such as DMA and interrupt. These new functions were added in HP-UX 11i; driver writers are encouraged to utilize them. Each is described in the Manpage Section of this document.

DMA Services

New interfaces:

```c
wsio_allocate_dma_handle()
wsio_allocate_shared_mem()
wsio_dma_pass_thru()
wsio_dma_set_device_attributes()
wsio_fast_map_dma_buffer()
wsio_free_dma_handle()
wsio_free_shared_mem()
wsio_flush_shared_mem()
wsio_init_map_context()
wsio_io_sync()
wsio_iova_to_phys()
wsio_map_dma_buffer()
wsio_remap_dma_buffer()
wsio_set_dma_attributes()
wsio_set_dma_callback()
wsio_unmap_dma_buffer()
```
Advantages of these new services:

- Multiple DMA handles can be allocated per card/instance
- Each handle can be configured for different types of DMA
- Many attributes are supported
- Drivers can set DMA attributes per card/instance or per handle
- Drivers can specify a callback function, which will be executed when resources become available.
- Drivers can allocate mapped buffers of varying sizes

Old Interfaces:

- `wsio_dma_alloc()`
- `wsio_dma_free()`
- `wsio_fastmap()`
- `wsio_get_pva()`
- `wsio_map()`
- `wsio_remap()`
- `wsio_set_attributes()`
- `wsio_unmap()`

Limitations of these services:

- Only one DMA object can be allocated per card/instance
- Only a limited number of DMA attributes can be set
- `wsio_dma_alloc()` only allocates buffers 64 bytes in size
- Services behave like the old 10.x services

Interrupt Services

New Interfaces:

- `wsio_intr_activate()`
- `wsio_intr_alloc()`
- `wsio_intr_deactivate()`
- `wsio_intr_deactivate_nowait()`
- `wsio_intr_free()`
- `wsio_intr_get_assigned_cpu()`
- `wsio_intr_get_irq_line()`
- `wsio_intr_get_txn_info()`
- `wsio_intr_set_cpu_spec()`
- `wsio_intr_set_irq_line()`
- `wsio_ordered_interrupts()`
Advantages of these new services:

- Multiple interrupt objects can be allocated per card/instance
- Each object can be configured independently

Old Interfaces

```c
isrlink()
isrunlink()
wsio_allocate_pa_interrupt()
wsio_get_interruptions()
wsio_get_eim_from_iobj()
wsio_get_new_eim()
wsio_reassign_eim_mask()
wsio_set_pa_interrupt()
```

Limitations:

- `isrlink()` can allocate a maximum of two objects
- Limited flexibility in managing the interrupt objects

Removed Macros

These macros will be deprecated in a later release. Do not use them.

```c
io.h/get_id
io.h/get_isc
sysmacros.h/m_slot
sysmacros.h/m_unit
sysmacros.h/m_vsc
wsio.h/m_funcnum
wsio.h/m_funcnum
wsio.h/m_slot
wsio.h/m_vsc
wsio.h/WSIO_SLEEP
wsio.h/WSIO_WAKEUP
wsio.h/wsio_sync_buses
wsio.h/wsio_shared_mem_lock
wsio.h/shared_mem_unlock
```
WSIO Memory Allocator

The WSIO Memory Allocator is a general purpose memory allocator that can be used by drivers for their memory requirements and allocation of DMA buffers. Drivers are strongly advised to move to this interface instead of the traditional MALLOC/FREE interface.

Memory Allocation Services

The following new set of WSIO memory services can be called by drivers to allocate and free memory for DMA buffers and other memory (except shared memory for card-driver communication).

- wsio_alloc_mem_handle()
- wsio_mem_alloc_attrib()
- wsio_free_mem_handle()
- wsio_alloc_mem()
- wsio_free_mem

These functions are available in the Manpage Section of this document.

32-Bit DMA Services

This set of services is meant for DMA transactions on platforms that support more than 4GB of memory and does not support IOPDIRs. In this case, the driver does not know whether the buffer is above or below 4 GB. On IA64 platforms with >4 GB, DMA transactions involving buffers above 4 GB will not be possible. The WSIO services: wsio_setup_dma32(), and wsio_cleanup_dma32(), allow the drivers to allocate duplicate buffers below 4 GB for DMA purposes. These WSIO services allocate the buffers and copy data from the original buffer to the new buffer and vice versa.

The services should typically be used when the WSIO mapping service returns a MAP_E_HIGH_ADDR error for an existing buffer. For example:
wsio_dma_map_t io_range;
void * tmp_buf;

retval = wsio_map_dma_buffer(isc, dma_handle, context,
hints, range_type,
user_level_buffer,
io_range);

if(retval == MAP_E_HIGH_ADDR)
{
    wsio_setup_dma32(isc, user_level_buffer, &tmp_buff,
size, dma_type);

    /* Use tmp_buf for the DMA transaction */

    wsio_cleanup_dma32(isc, user_level_buffer, tmp_buf,
size, dm_type);
}

On the IA64 platforms, wsio_map_dma_buffer() simply checks if the host buffer is below 4 GB. If it is not, it returns MAP_E_HIGH_ADDR. If the DMA transaction is outbound, wsio_setup_dma32() copies data from the original buffer to the temporary buffer. If it is an inbound transaction, wsio_cleanup_dma32() copies data from the temporary buffer to the original buffer.

See the details of these functions in the Manpage Section of this document.
Event Handling

The HP-UX 11i version of WSIO comes with a rich set of interfaces to allow a driver to register a driver’s event handler and identify what event it is capable of dealing. When the event occurs, WSIO will call the driver’s handler. Online replacement and addition of PCI cards is also supported with this version (see the On-Line Addition/Replacement chapter in the HP-UX 11i DDG). A driver can take advantage of these capabilities through these new entry points. The new interfaces dealing with event handling are:

```c
wsio_install_drv_event_handler()
wsio_reg_drv_capability_mask(0)
wsio_query_supported_function()
```

See their descriptions in the Manpages Section of this document.
2 Manpages
New Services

This chapter contains the manpages for the new services provided by HP-UX version 11i.
NAME

`wsio_alloc_mem` (WSIO3) - Service to allocate memory for DMA buffers or control structures.

SYNOPSIS

```c
#include<sys/wsio.h>

wsio_alloc_status_t
wsio_alloc_mem (wsio_mem_handle_t mem_handle,  
    size_t size,  
    wsio_vaddr_t * vaddr,  
    wsio_alloc_flags_t flags)
```

PARAMETERS

- **`mem_handle`** A handle allocated by a call to `wsio_alloc_mem_handle`
- **`size`** The size of the buffer.
- **`vaddr`** A pointer to the allocated buffer if successful else NULL
- **`flags`** Flags which describe restrictions
  - `WSIO_WAITOK` Flag to indicate service can sleep if waiting for resources

DESCRIPTION

This WSIO service can be called by drivers to allocate memory for DMA buffers or control structures. The first parameter to the service must be a `mem_handle` that was allocated by the driver using `wsio_alloc_mem_handle`. 
**RETURN VALUES**

*WSIO_ALLOC_OK*  
Indicates that the buffer was allocated

*WSIO_ALLOC_OUT_OF_RESOURCES*  
Unable to allocate the specified resources

**CONSTRAINTS**

**EXAMPLES**

**SEE ALSO**

*wsio_free_mem*
NAME

wsio_alloc_mem_handle(WSIO3) - Service to specify the type of memory to allocate.

SYNOPSIS

#include<sys/wsio.h>

wsio_alloc_status_t
wsio_alloc_mem_handle (struct isc_table_type *isc,
                      wsio_mem_handle_t *mem_handle,
                      wsio_mem_alloc_attrib_t attribs)

PARAMETERS

isc A pointer to the device's isc structure

mem_handle A pointer to a variable where the handle will be stored upon completion.

attribs Attributes describing the criteria for the type of memory to allocate.

WSIO_OPTIMIZE_FOR_DEVICE
Allocate memory close to the device.
On Superdome systems, memory will be allocated on the same cell as the device.

WSIO_OPTIMIZE_FOR_CPU
Allocate memory close to the current CPU. On Superdome systems, memory will be allocated on the same cell as the CPU. This is the default behavior.

WSIO_32BIT_MEMORY
The buffer must be allocated below 4GB.
WSIO_ALLOC_MEM_HANDLE(WSIO)

DESCRIPTION

Drivers will call this service to specify the type of memory they want to allocate. The service will return a mem_handle which can be passed into the WSIO memory alloc and free routines. Drivers can allocate more than one mem_handle to specify different criteria for memory allocation.

While it is not a requirement, it is recommended that drivers call this routine early in their initialization sequence. This is due to the high overhead of the routine. Buffer alignment is as follows:

Allocation Size Buffer Aligned On
Less Than Cacheline Size       32 Byte Boundary
Greater Than or Equal to Cacheline Size Cacheline Boundary
Greater Than or Equal to I/O Page Size (4K) 4K Boundary

RETURN VALUES

WSIO_ALLOC_OK
    Indicates that the buffer was allocated
WSIO_ALLOC_OUT_OF_RESOURCES
    Unable to allocate the specified resources
WSIOINVAL_PARAM
    A parameter was not valid

CONSTRAINTS

None
EXAMPLES

SEE ALSO

wsio_free_mem_handle
NAME

wsio_allocate_dma_handle(WSIO3) – Obtain a handle used to setup DMA

SYNOPSIS

#include <sys/wsio.h>

void *wsio_allocate_dma_handle (struct isc_table_type *isc);

PARAMETERS

isc Pointer to the driver’s isc_table entry.

DESCRIPTION

The wsio_allocate_dma_handle() WSIO function is called by device drivers to obtain a DMA handle. This handle, which is passed to all DMA services, can be associated with various DMA hints, and is used to control DMA. Multiple handles can be allocated, allowing a device driver to associate different hints with each handle.

RETURN VALUES

A void pointer to the handle. If NULL is returned, a handle count could not be allocated.

CONSTRAINTS

EXAMPLE

void *dma_handle;

dma_handle = wsio_allocate_dma_handle(isc_entry);
if (dma_handle == NULL {
    /* No handle allocated. */
    return (ERROR);
}
/* The DMA handle is now in the dma_handle variable */
SEE ALSO

wsioAllocate dma handle(WSIO3),
wsioAllocate shared mem(WSIO3),
wsioAllocate dma pass thru(WSIO3),
wsioFastmap dma buffer(WSIO3),
wsioFree dma handle(WSIO3),
wsioFree shared mem(WSIO3),
wsioFlush shared mem(WSIO3),
wsioInit map context(WSIO3),
wsioIova to phys(WSIO3),
wsioMap dma buffer(WSIO3),
wsioRemap dma buffer(WSIO3),
wsioSet device attributes(WSIO3),
wsioSet dma attributes(WSIO3),
wsioUnmap dma buffer(WSIO3)
NAME

wsio_allocate_shared_mem(WSIO3) - Set up an I/O virtually contiguous DMA buffer.

SYNOPSIS

#include <sys/wsio.h>

wsio_map_status_t wsio_allocate_shared_mem (
    struct isc_table_type *isc, void *dma_handle, 
    size_t size, wsio_iova_t *iova, wsio_vaddr_t *vaddr, 
    wsio_shared_mem_attr_t shared_mem_attr);

PARAMETERS

isc Pointer to the driver’s isc_table entry.
dma_handle DMA handle allocated using wsio_allocate_dma_handle().
size Size of buffer to allocate.
iova Pointer that contains the I/O virtual address upon completion. A wsio_iova_t must be allocated by the driver, and the pointer to this is what should be passed into the macro.
vaddr Pointer that contains the host virtual address upon completion. A wsio_vaddr_t must be allocated by the driver, and the pointer to this is what should be passed into the macro.
shared_mem_attr Bitmask that indicates how to allocate the memory. The acceptable values are described in the following list. If a type of 0 is used, the default behavior of WSIO_IO_SHMEM_OPTIMIZE_DEVICE_LATENCY is used. The behavior of the allocation is also affected by attributes set using wsio_dma_set_device_attributes(), and wsio_set_dma_attributes().
The following are the `wsio_shared_mem_attr_t` allowable bitmask values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>WSIO_IO_SHMEM_OPTIMIZE_DEVICE_LATENCY</code></td>
<td>Allocation should optimize for device access latency. If possible, allocate object in memory local to a bus bridge.</td>
</tr>
<tr>
<td><code>WSIO_IO_SHMEM_OPTIMIZE_HOST_LATENCY</code></td>
<td>Allocation should optimize for host access latency. If possible, allocate in host memory.</td>
</tr>
<tr>
<td><code>WSIO_IO_SHMEM_DMA_ALLOC_COMPATIBLE</code></td>
<td>Behave exactly as the 10.X <code>dma_alloc</code> service. This type is for compatibility with 10.X only.</td>
</tr>
<tr>
<td><code>WSIO_IO_SHMEM_INBOUND</code></td>
<td>This attribute can be OR'ed with the other attributes to indicate the buffer is used exclusively for inbound DMA.</td>
</tr>
<tr>
<td><code>WSIO_IO_SHMEM_OUTBOUND</code></td>
<td>This attribute can be OR'ed with the other attributes to indicate the buffer is used exclusively for outbound DMA.</td>
</tr>
<tr>
<td><code>WSIO_IO_SHMEM_DEV_WEAK_OK</code></td>
<td>This attribute can be OR'ed with the other attributes to indicate the accesses to the buffer can be weakly ordered. The default ordering is the strongest that can be provided for the given I/O bus.</td>
</tr>
<tr>
<td><code>WSIO_IO_SHMEM_ALIGN_ON_SIZE</code></td>
<td>This attribute can be OR'ed with the other attributes to specify <code>size</code> also indicates the alignment boundary for the allocation.</td>
</tr>
</tbody>
</table>
DESCRIPTION

The `wsio_allocate_shared_mem()` WSIO function is called by a device driver to allocate an I/O virtually contiguous DMA buffer that is to be used for continuous DMA. Continuous DMA means that the memory appears contiguous to the I/O device and can be read or written by the I/O device on a continuous basis. For packet DMA or DMA that is used for temporary mappings, `wsio_map_dma_buffer()` should be used.

If a callback function is set up (see `wsio_set_dma_callback()`), and no resources are available when the call is made, `WSIO_MAP_W_CALLBACK` will be returned, and the callback will be triggered when resources become available.

This function may be called in a non-blocking context.

RETURN VALUES

- **WSIO_MAP_OK**  
  Success.

- **WSIO_MAP_W_CALLBACK**  
  Returned if no resources are available and a callback is registered.

- **WSIO_MAP_E_NO_RESOURCES**  
  Returned if no resources are available and no callback is registered.

- **WSIO_MAP_E_RESOURCE_ERROR**  
  Returned if cannot allocate resources. If this is returned, the allocation will never succeed.

- **WSIO_MAP_E_PARAMETER_ERROR**  
  Returned on bad parameter (Software bug).

- **WSIO_MAP_E_UNKNOWN_ERROR**  
  Returned if there is an unknown error.

CONSTRAINTS
Example

```c
void *dma_handle = NULL;
wsio_iova_t io_virtual_addr;
wsio_vaddr_t host_virtual_addr;

dma_handle = wsio_allocate_dma_handle(isc_entry);
if (dma_handle == NULL) {
    /* No handle allocated. */
    return ERROR;
}

/* The DMA handle is now in the dma_handle variable */

if (wsio_allocate_shared_mem(isc_entry,dma_handle,buf_size,
    &io_virtual_addr,&host_virtual_addr,0) != WSIO_MAP_OK) {
    /* Unable to allocate shared memory, so return an error */
    return ERROR;
}

/* A buffer of size 'buf_size' is now allocated and mapped
   * in both host virtual memory and I/O virtual memory space. */
```

See Also

wsio_allocate_dma_handle(WSIO3), wsio_dma_pass_thru(WSIO3),
wsio_free_dma_handle(WSIO3), wsio_free_shared_mem(WSIO3),
wsio_flush_shared_mem(WSIO3), wsio_iova_to_phys(WSIO3),
wsio_set_device_attributes(WSIO3),
wsio_set_dma_attributes(WSIO3)
NAME

wsio_cleanup_dma32(WSIO3) – Cleans up after a buffer allocated by wsio_setup_dma32

SYNOPSIS

#include<sys/wsio.h>

void
wsio_cleanup_dma32 (struct isc_table_type *isc,
    (    
        void *drv_buf,
        void *tmp_buf,
        size_t size,
        wsio_dma_buffer_hints_t, trans);

PARAMETERS

isc Pointer to the device isc structure.
drv_buf Pointer to the original buffer provided by the driver if DMA was inbound, else NULL.
tmp_buf Pointer to the buffer allocated by the service wsio_setup_dma32().
size The size of the buffer
trans Indicates the direction of the DMA action:
    WSIO_DMA_INBOUND Writing to memory
    WSIO_DMA_OUTBOUND Reading from memory

If (trans == WSIO_DMA_INBOUND) the data will be copied from the temporary buffer to the driver buffer.

DESCRIPTION

This service frees the temporary buffer. If DMA was inbound, it copies the data from the temporary buffer to the original before doing so. If DMA was outbound, no copying is required, and the driver can pass a NULL value for drv_buf.
RETURN VALUES

None

CONSTRAINTS

EXAMPLE

/*
 * tmp_buf points to a buffer allocated by a previous call
 * to wsio_setup_dma32()
 */
if(tmp_buffer != NULL)
    wsio_cleanup_dma32(my_isc, drv_buf, tmp_buf, buf_size,
                       WSIO_DMA_OUTBOUND);

SEE ALSO

wsio_setup_dma32()
NAME

wsio_dma_pass_thru(WSIO3) – Call a DMA pass-through function that might not otherwise be accessible.

SYNOPSIS

```c
#include <sys/wsio.h>

intptr_t wsio_dma_pass_thru (struct isc_table_type *isc,
                              void *dma_handle,
                              wsio_pt_type_t pass_thru_type,
                              intptr_t pass_thru_param);
```

PARAMETERS

- **isc**: Pointer to the driver’s isc_table entry.
- **dma_handle**: DMA handle allocated using `wsio_allocate_dma_handle()`.
- **pass_thru_type**: Indicates which pass-thru function to call. Two types are defined as shown in the following items:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSIO_MAP_PT_DEVICE_LOCK</td>
<td>Prevents a device’s access to shared memory. This can be used so that processors can access host memory atomically, and can be used for synchronization. The parameter’s pass_thru_param indicates whether shared memory should be locked (1) or unlocked (0). Implementation of this function is not required, so the return code is zero (0) if the function is implemented or non-zero if not implemented.</td>
</tr>
</tbody>
</table>
DESCRIPTION

The `wsio_dma_pass_thru()` WSIO function is present to allow new DMA interfaces to be added without breaking binary compatibility. It allows such interfaces to be accessed if they are present.

RETURN VALUES

Dependent on the specific pass-thru function being called.

CONSTRAINTS

EXAMPLE

```c
if (wsio_dma_pass_thru(isc_entry, dma_handle, 
                     WSIO_MAP_PT_SYNC_BUSSES, 0) != 0) { 
    /* The function isn't implemented */ 
    return(ERROR); 
} else { 
    /* The function is implemented, and completed correctly */ 
    return(0); 
} 
/* This code will attempt to sync memory associated with 
* dma_handle 
*/
```
SEE ALSO

wsio_allocate_dma_handle(WSIO3),
wsio_allocate_shared_mem(WSIO3),
wsio_fastmap_dma_buffer(WSIO3), wsio_flush_shared_mem(WSIO3),
wsio_free_dma_handle(WSIO3), wsio_free_shared_mem(WSIO3),
wsio_init_map_context(WSIO3), wsio_iova_to_phys(WSIO3),
wsio_map_dma_buffer(WSIO3), wsio_remap_dma_buffer(WSIO3),
wsio_set_device_attributes(WSIO3),
wsio_set_dma_attributes(WSIO3) wsio_unmap_dma_buffer(WSIO3)
NAME

wsio_dma_set_device_attributes(WSIO3) – Associate DMA hints with a device

SYNOPSIS

#include <sys/wsio.h>

wsio_map_status_t wsio_dma_set_device_attributes(
    struct isc_table_type *isc,
    void *dma_handle,
    wsio_dma_attribute_t attribute
    wsio_dma_attr_param_t param);

PARAMETERS

isc Pointer to the driver’s isc_table entry.
dma_handle DMA handle allocated using
    wsio_allocate_dma_handle().
attribute Indicates which hint to set for the device associated
    with dma_handle. The possible attributes follow:
    WSIO_DMA_ATTR_ADDR_WIDTH
        Bits of addressing supported by the device. This is used to determine
        whether a device can DMA directly to memory buffers.
        Default value = 32
    WSIO_DMA_ATTR_ALIGNMENT
        Byte alignment of DMA buffer required for device.
        Default value = HW Dep.
WSIO_DMA_ATTR_ATM
ATM hint. Used by hardware in some
implementations.

0 = not ATM
1 = ATM 48 (optimize for 48-byte
transfers)
2 = ATM 192 (optimize for
192-byte transfers)

Default value = 0

WSIO_DMA_ATTR_CALLBACK
Specifies a function to call when
resources become available.

Default value = NULL

WSIO_DMA_ATTR_CALLBACK_ARG
Specifies an argument to the callback
function.

Default value = 0

WSIO_DMA_ATTR_FLUSH_ON_USE
Specifies the cacheline should be
flushed from any intermediate
buffers as soon as it is referenced.
This inhibits any coalescing of data
by bus bridges.

Default value = 0

WSIO_DMA_ATTR_IGN_ALIGN
Specifies the mapping service should
not handle cacheline fragments in a
special way.

Default value = 0
WSIO_DMA_ATTR_INTERLEAVE

IOVA allocation model

0 = DMA streams are normally interleaved (mass-storage).
1 = DMA streams are normally not interleaved (networking)
2 = DMA buffers are static and accessed randomly (low fat).

Default value = 0

WSIO_DMA_ATTR_PREFETCH

Specifies how aggressively hardware should prefetch for outbound DMA.

0 = no prefetch
1 = moderate prefetch
2 = aggressive prefetch

Default value = 1

WSIO_DMA_ATTR_SAFE

Specifies the most conservative coherency model should be used for inbound DMA. Inhibits semicoherent transactions such as WRITE_PURGE unless it is guaranteed that no data in processor caches will be lost.

1 = ON
2 = OFF

Default value = 0

WSIO_DMA_ATTR_TXN_SIZE

Specifies the default transaction size used by the device. This is used by hardware to optimize conversion of transactions between buses.

Default value = HW Dep.

WSIO_DMA_ATTR_INBOUND

DMA buffers will be used exclusively for inbound DMA.

Default value = 0
wsio_dma_set_device_attributes(WSIO3)

**DESCRIPTION**

The `wsio_dma_set_device_attributes()` WSIO function is used to associate DMA transaction hints and attributes with a specific device. These hints are overridden by any hints set for a specific DMA handle via `wsio_set_dma_attributes()`, or some hints passed in as parameters to `wsio_map_dma_buffer()`.

`wsio_dma_set_device_attributes()` can be called in a non-blocking context. If `WSIO_DMA_ATTR_INTERLEAVE` is set to 1, a subsequent `wsio_allocate_shared_mem()` or `wsio_map_dma_buffer()` can only successfully request a buffer with a maximum of one page(4K) in length and this buffer cannot cross a page boundary. This is the limitation placed by the underlying platform. If a larger buffer is desirable, use the default value of 0. This larger buffer can be used for control structures rather than packet DMAs.

Do not call `wsio_dma_set_device_attributes()` to set the `WSIO_DMA_ATTR_INTERLEAVE` to the default value of 0. The call will fail.

**RETURN VALUES**

- **WSIO_MAP_OK**  Success.
- **WSIO_MAP_E_PARAMETER_ERROR**  Returned if an invalid parameter has caused failure of the call.
CONSTRAINTS

EXAMPLE

if (wsio_dma_set_device_attributes(isc_entry, dma_handle,  
    WSIO_DMA_ATTR_INTERLEAVE, 1) != WSIO_MAP_OK) {  
    /* There was a parameter error */  
    return(ERROR);  
} else {  
    /* DMA streams are now not normally interleaved for all DMA  
      * associated with this device  
      */  
    return(0);  
}

SEE ALSO

wsio_allocate_dma_handle(WSIO3),  
wsio_allocate_shared_mem(WSIO3), wsio_dma_pass_thru(WSIO3),  
wsio_fastmap_dma_buffer(WSIO3), wsio_free_dma_handle(WSIO3),  
wsio_free_shared_mem(WSIO3), wsio_flush_shared_mem(WSIO3),  
wsio_init_map_context(WSIO3), wsio_iova_to_phys(WSIO3),  
wsio_map_dma_buffer(WSIO3), wsio_remap_dma_buffer(WSIO3),  
wsio_set_dma_attributes(WSIO3), wsio_unmap_dma_buffer(WSIO3),
NAME

wsio_fastmap_dma_buffer (WSIO3) - Function to map an existing memory object for packet DMA.

SYNOPSIS

```
#include <sys/wsio.h>

wsio_map_status_t wsio_fastmap_dma_buffer ( 
    struct isc_table_type *isc, void *dma_handle, 
    wsio_range_type_t range_type, 
    wsio_dma_map_t *host_range, 
    wsio_dma_map_t *io_range);
```

PARAMETERS

isc Pointer to the driver's isc_table entry.
dma_handle DMA handle allocated using wsio_allocate_dma_handle().
range_type Indicates the type of host memory being mapped. It can be:
    KERNELSPACE Indicates host_range is a kernel virtual buffer.
    PHYSICAL Indicates host_range is a physical buffer.
    >0 Indicates host_range is in user space, and this will be the space ID of the virtual address.
host_range Pointer to an address/length structure that contains information about the host space to map. If the mapping was only partially completed, this will contain information about the remaining space to be mapped when the call completes.
io_range Pointer to an address/length structure that will contain information about the I/O space that was mapped.
DESCRIPTION

The `wsio_fastmap_dma_buffer()` function is called by a device driver to map an existing memory object for packet DMA. It operates in the same way as `wsio_map_dma_buffer()` except the entire host address range must reside on a single physical page. If the range cannot be mapped in a single call, an error will be returned. Cacheline fragments are ignored (the same behavior as `WSIO_DMA_IGN_ALIGNMENT`).

The size of a buffer can have a maximum length of one page (4K) and this buffer can not cross a page boundary.

All mappings will remain in effect until `wsio_unmap_dma_buffer()` or `wsio_remap_dma_buffer()` are called to remove or change them.

Device drivers can set up a callback routine that will come into play if resources are not available at the time a mapping is attempted. If this callback is set up, `WSIO_MAP_W_CALLBACK` will be returned instead of a no resource error. When resources become available, the callback routine will be called to indicate this to the device driver. For more information on how to setup and use a callback, see the `wsio_set_dma_callback()` manpage. `wsio_fastmap_dma_buffer()` can be called in a non-blocking context.

RETURN VALUES

- **WSIO_MAP_OK**  Returned if the entire buffer has been mapped.
- **WSIO_MAP_W_CALLBACK**  Returned if no resources are available and a callback function exists.
- **WSIO_MAP_E_NO_RESOURCES**  Returned if no resources are available and no callback function exists.
- **WSIO_MAP_ERESOURCE_ERROR**  Returned if the request cannot and will never succeed.
- **WSIO_MAP_EHIGH_ADDR**  Returned if the call failed because the device cannot reach the destination address.
- **WSIO_MAP_EPARAMETER_ERROR**  Returned if an invalid parameter has caused failure of the call.
WSIO_MAP_E_UNKNOWN_ERROR
Returned for hardware or other errors.

CONSTRAINTS

EXAMPLE

```c
void *dma_handle;
wsio_dma_map_t host_range, io_range;

dma_handle = wsio_allocate_dma_handle(isc_entry);
host_range.iov_base = host_virtual_address;
host_range.iov_len = dma_buffer_length;

if (wsio_fastmap_dma_buffer(isc_entry, dma_handle,
  KERNELSPACE, &host_range, &io_range) != WSIO_MAP_OK) {
  /* Unable to map the range, so return an error */
  return(ERROR);
}

/* The host virtual buffer represented in the above code by
 * 'host_virtual_address' is now mapped.
 */
```

SEE ALSO

wsio_allocate_dma_handle(WSIO3),
wsio_allocate_shared_mem(WSIO3), wsio_dma_pass_thru(WSIO3),
wsio_free_dma_handle(WSIO3), wsio_free_shared_mem(WSIO3),
wsio_flush_shared_mem(WSIO3), wsio_init_map_context(WSIO3),
wsio_iova_to_phys(WSIO3), wsio_map_dma_buffer(WSIO3),
wsio_remap_dma_buffer(WSIO3),
wsio_set_device_attributes(WSIO3),
wsio_set_dma_attributes(WSIO3), wsio_unmap_dma_buffer(WSIO3),
NAME

wsio_flush_shared_mem(WSIO3) - Flush an I/O virtually contiguous DMA buffer.

SYNOPSIS

```c
#include <sys/wsio.h>

wsio_map_status_t wsio_flush_shared_mem (
    struct isc_table_type *isc, void *dma_handle, 
    size_t size, wsio_iova_t iova, wsio_vaddr_t vaddr, 
    wsio_shared_mem_attr_t shared_mem_attr);
```

PARAMETERS

- **isc**  
  Pointer to the driver's isc_table entry.

- **dma_handle**  
  DMA handle allocated using  
  wsio_allocate_dma_handle.

- **size**  
  Size of buffer to be flushed.

- **iova**  
  I/O virtual address of the shared memory.

- **vaddr**  
  Host virtual address of the shared memory.

- **shared_mem_attr**  
  Bitmask that was used to allocate the shared memory.

DESCRIPTION

The `wsio_flush_shared_mem()` WSIO function is called by a device driver to guarantee the consistency of the memory object allocated via `wsio_allocate_shared_mem()`. Any non-coherent buffers associated with the memory object are flushed. All parameters to the function should be the same as those passed to the call that allocated the memory. `wsio_flush_shared_mem()` can be called in a non-blocking context.

RETURN VALUES

- **WSIO_MAP_OK**  
  Success.

- **WSIO_W_NOP**  
  The call has no effect. The caller need not call it again.
CONSTRAINTS

EXAMPLE

```c
void *dma_handle = NULL;
wsio_iova_t io_virtual_addr;
wsio_vaddr_t host_virtual_addr;

dma_handle = wsio_allocate_dma_handle(isc_entry);
if (dma_handle == NULL) {
    /* No handle allocated. */
    return(ERROR);
}
/* The DMA handle is now in the dma_handle variable */

if (wsio_allocate_shared_mem(
    isc_entry, dma_handle, buf_size, &io_virtual_addr,
    &host_virtual_addr, 0) != WSIO_MAP_OK) {
/* Unable to allocate the shared memory,
    * so return an error */
    return(ERROR);
}
/* A buffer of size 'buf_size' is now allocated and mapped in
    * both host virtual memory and I/O virtual memory space.
    */
wsio_flush_shared_mem(isc_entry, dma_handle, buf_size,
    io_virtual_addr, host_virtual_addr, 0);
```

SEE ALSO

wsio_allocate_dma_handle(WSIO3),
wsio_allocate_shared_mem(WSIO3), wsio_dma_pass_thru(WSIO3),
wsio_free_dma_handle(WSIO3), wsio_free_shared_mem(WSIO3),
wsio_iova_to_phys(WSIO3), wsio_set_device_attributes(WSIO3),
wsio_set_dma_attributes(WSIO3)
NAME

wsio_free_dma_handle (WSIO3) - Release a DMA handle.

SYNOPSIS

#include <sys/wsio.h>

void wsio_free_dma_handle (struct isc_table_type *isc, void *dma_handle);

PARAMETERS

isc Pointer to the driver’s isc_table entry.

dma_handle Pointer to the DMA handle to free.

DESCRIPTION

The wsio_free_dma_handle () WSIO function is called by device drivers to release a handle that has been allocated by wsio_allocate_dma_handle (). It should be called anytime a handle is no longer needed.

RETURN VALUES

None.

CONSTRAINTS
EXAMPLE

```c
void *dma_handle;

dma_handle = wsio_allocate_dma_handle(isc_entry);
if (dma_handle == NULL) {
    /* No handle allocated. */
    return(ERROR);
}
/* The DMA handle is now in the dma_handle variable */

wsio_free_dma_handle(isc_entry,dma_handle);
/* The DMA handle has now been released */
```

SEE ALSO

wsio_alloc DMA handle (WSIO3),
wsio_allocate_shared_mem (WSIO3), wsio_dma_pass_thru (WSIO3),
wsio_fastmap_dma_buffer (WSIO3), wsio_free_shared_mem (WSIO3),
wsio_flush_shared_mem (WSIO3), wsio_init_map_context (WSIO3),
wsio_i ova_to_phys (WSIO3), wsio_map_dma_buffer (WSIO3),
wsio_remap_dma_buffer (WSIO3),
wsio_set_device_attributes (WSIO3),
wsio_set_dma_attributes (WSIO3), wsio_unmap_dma_buffer (WSIO3),
NAME

wsio_free_mem(WSIO3) - Frees memory allocated by wsio_alloc_mem.

SYNOPSIS

```c
#include<sys/wsio.h>

void wsio_free_mem (wsio_mem_handle_t mem_handle,
                    wsio_vaddr_t vaddr)
```

PARAMETERS

- `mem_handle`: A handle allocated by a call to `wsio_alloc_mem_handle`
- `vaddr`: A pointer to the allocated buffer

DESCRIPTION

This WSIO service is called to free memory allocated by the service `wsio_alloc_mem()`.

RETURN VALUES

None.

CONSTRAINTS

EXAMPLE

SEE ALSO

wsio_alloc_mem
NAME

wsio_free_mem_handle (WSIO3) – Destroy handle previously allocated by
wsio_alloc_mem_handle.

SYNOPSIS

#include<sys/wsio.h>

void wsio_free_mem_handle (wsio_mem_handle_t mem_handle)

PARAMETERS

mem_handle A handle allocated by a call to
wsio_alloc_mem_handle.

DESCRIPTION

Drivers call this service to destroy a mem_handle that was allocated by a
previous call to wsio_alloc_mem_handle().

RETURN VALUES

None.

CONSTRAINTS

EXAMPLE

SEE ALSO

wsio_alloc_mem_handle
NAME

wsio_free_shared_mem(WSIO3) - Release an I/O virtually contiguous DMA buffer.

SYNOPSIS

```c
#include <sys/wsio.h>

void wsio_free_shared_mem (struct isc_table_type *isc,
                           void *dma_handle, size_t size,
                           wsio_iova_t iova, wsio_vaddr_t vaddr,
                           wsio_shared_mem_attr_t shared_mem_attr);
```

PARAMETERS

- `isc` Pointer to the driver’s isc_table entry.
- `dma_handle` DMA handle allocated using `wsio_allocate_dma_handle()`.
- `size` Size of buffer to be released.
- `iova` I/O virtual address of the shared memory.
- `vaddr` Host virtual address of the shared memory.
- `shared_mem_attr` Bit mask that was used to allocate the shared memory.

DESCRIPTION

The `wsio_free_shared_mem()` WSIO function is called by a device driver to release an I/O virtually contiguous DMA buffer that was allocated by `wsio_allocate_shared_mem()`. All parameters to the macro should be the same as those passed to the call that allocated the memory.

`wsio_free_shared_mem()` can be called in a non-blocking context.

RETURN VALUES

None
Manpages
wsio_free_shared_mem(WSIO3)

CONSTRAINTS

EXAMPLE

```c
void *dma_handle = NULL;
wsio_iova_t io_virtual_addr;
wsio_vaddr_t host_virtual_addr;

dma_handle = wsio_allocate_dma_handle(isc_entry);
if (dma_handle == NULL) {
    /* No handle allocated. */
    return(ERROR);
}
/* The DMA handle is now in the dma_handle variable */

if (wsio_allocate_shared_mem(isc_entry,dma_handle,buf_size,
    &io_virtual_addr,
    &host_virtual_addr,0) != WSIO_MAP_OK) {
    /* Unable to allocate shared memory,
    * so return an error */
    return(ERROR);
}
/* A buffer of size 'buf_size' is now allocated and mapped
* in both host virtual memory and I/O virtual memory space. */
wsio_free_shared_mem(isc_entry,dma_handle,buf_size,
    io_virtual_addr,host_virtual_addr,0);
```

SEE ALSO

wsio_allocate_dma_handle(WSIO3),
wsio_allocate_shared_mem(WSIO3), wsio_dma_pass_thru(WSIO3),
wsio_free_dma_handle(WSIO3), wsio_flush_shared_mem(WSIO3),
wsio_iova_to_phys(WSIO3), wsio_set_device_attributes(WSIO3),
wsio_set_dma_attributes(WSIO3)
NAME

wsio_init_map_context (WSIO3) - Initialize the context used for DMA mapping.

SYNOPSIS

```
#include <sys/wsio.h>

void wsio_init_map_context (wsio_map_context_t *context);
```

PARAMETERS

context    Pointer to the context to be initialized.

DESCRIPTION

The `wsio_init_map_context()` WSIO function is called by device drivers to initialize a context that is needed by `wsio_map_dma_buffer()`, and other map related functions. A context is used internally by mapping services so that system resources can be used efficiently. The same context should be used for a set of DMA mappings that are all going to be unmapped at the same time. This context should be initialized before its first use. If the mapping that is being performed is independent of all other mappings, then a context is not necessary and `NULL` should be passed to the mapping function in the context field.

RETURN VALUES

None

CONSTRAINTS
EXAMPLE

wsio_map_context_t dma_context;

wsio_init_map_context(&dma_context);
/* dma_context is now ready to be used for mapping. It can
 * be used for any number of mappings as long as they are
 * all unmapped at the same time
 */

SEE ALSO

wsio_allocate_dma_handle(WSIO3), wsio_dma_pass_thru(WSIO3),
wsio_fastmap_dma_buffer(WSIO3), wsio_free_dma_handle(WSIO3),
wsio_iova_to_phys(WSIO3), wsio_map_dma_buffer(WSIO3),
wsio_remap_dma_buffer(WSIO3),
wsio_set_device_attributes(WSIO3),
wsio_set_dma_attributes(WSIO3), wsio_unmap_dma_buffer(WSIO3).
NAME

wsio_install_drv_event_handler(WSIO3) - Install a driver’s event handler

SYNOPSIS

```
#include <sys/wsio.h>

int wsio_install_drv_event_handler (wsio_drv_info_t * I_drv_hdr,
                                  wsio_drv_event_handler_t I_drv_handler)
```

PARAMETERS

- `I_drv_hdr` Pointer to the driver’s `wsio_drv_info_t` structure.
- `I_drv_handler` Function pointer to the driver’s handler. The type `wsio_drv_event_handler_t` is defined as follows in the header file `wsio.h`:

```c
typedef void (*wsio_drv_event_handler_t) __attribute__((__null_check__)) ((wsio_generic_event_t *));
```

DESCRIPTION

The `wsio_install_drv_event_handler()` WSIO function is called by a device driver to register an event handler to deal with events. When an event occurs, such as a suspend request, WSIO will invoke this handler. This function should be called in a driver’s installation routine after `wsio_install_driver()` is executed.

RETURN VALUES

- `WSIO_OK` Returned on success.
- `WSIO_DRV_NOT_FOUND` An earlier call to `wsio_install_driver()` was not successful.
- `WSIO_HANDLER_NULL` Drv_handler is a NULL pointer.
WSIO_INFO_NULL

Drv_info is a NULL pointer.

CONSTRAINTS

EXAMPLE

static wsio_drv_info_t my_drv_info {
    &my_info,
    &my_ops,
    &my_data,
    WSIO_DRV_CURRENT_VERSION,
    /* This is a macro defined in wsio.h, bearing a version
     * stamp */
}

my_driver_install() {
    if (wsio_install_driver(&my_drv_info) != 1) { /* 0 means error */
        return (0);
    }
    /* Register the event handler with WSIO */
    if (wsio_install_drv_event_handler(
        &my_drv_info, my_handler) != WSIO_OK) {
        /* The example given shows that wsio_uninstall_driver()
         * is called. In this case, the driver
         * will not be brought up. If a driver chooses to
         * ignore the error and proceed, it will
         * run without being able to handle events */
        wsio_uninstall_drv_event_handler(&my_drv_info);
        return (0); /* 0 means error */
    }
}

SEE ALSO

wsio_query_supported_function(WSIO_DRV),
wsio_reg_drv_capability_mask(WSIO3),
wsio_uninstall_drv_event_handler(WSIO3)
NAME

wsio_intr_activate (WSIO3) – Enable an interrupt object.

SYNOPSIS

#include <sys/wsio.h>

int wsio_intr_activate (struct isc_table_type *isc,
                        wsio_intr_object_t obj);

PARAMETERS

isc Pointer to the driver’s isc_table entry.
obj Interrupt object to enable.

DESCRIPTION

The wsio_intr_activate() WSIO function activates an interrupt object that was allocated with wsio_intr_alloc(). The interrupt object must be activated before the system will call the device driver’s ISR (as specified in wsio_intr_alloc()). It is assumed that (if possible) the device will not generate interrupts until after this function is called. The wsio_intr_deactivate() or wsio_intr_deactivate_nowait() service undoes the effects of this function.

Attempting to activate an interrupt object that is already active is an error condition that returns WSIO_INTR_ACTIVATED, without modifying the interrupt object.

RETURN VALUES

WSIO_OK Operation succeeded.
WSIO_ERROR Failure; could not enable obj.
WSIO_INTR_INV_OBJ Must call wsio_intr_set_cpu_spec() or wsio_intr_set_irq_line() first.
WSIO_INTR_ACTIVATED obj already active.
WSIO_PARM_ERROR
Invalid parameters.

CONSTRAINTS

EXAMPLE

/* Allocate a line based interrupt and activate it */
wsio_intr_object_t obj;
int status;

/* allocate an interrupt object */
status = wsio_intr_alloc(isc, driver_isr,
    (uintptr_t)isc, 0, &obj);
if (status != WSIO_OK) {
    return(ERROR);
}
/* Get a Level Sensitive IRQ */
status = wsio_intr_set_irq_line(isc, obj,
    WSIO_IRQ_LINE_AUTO, 0);
if (status != WSIO_OK) {
    return(ERROR);
}
/* activate the interrupt */
status = wsio_intr_activate(isc, obj);
if (status != WSIO_OK) {
    return(ERROR);
}

SEE ALSO

wsio_intr_alloc(WSIO3), wsio_intr_deactivate(WSIO3),
wsio_intr_deactivate_nowait(WSIO3),
wsio_intr_set_cpu_spec(WSIO3), wsio_intr_set_line(WSIO3)
NAME

wsio_intr_alloc(WSIO3) - Allocate an interrupt object.

SYNOPSIS

#include <sys/wsio.h>

int wsio_intr_alloc (struct isc_table_type *isc,
                     wsio_drv_isr_t isr,
                     uintptr_t arg,
                     uint64_t flags,
                     wsio_intr_object_t *obj);

PARAMETERS

isc Pointer to the driver's isc_table entry.
isr Address of the interrupt service routine.
arg Argument to be passed to the driver_isr.
flags Shared or exclusive flag.
obj Interrupt object.

DESCRIPTION

The \texttt{wsio_intr_alloc()} function allocates and initializes an interrupt object that will field interrupts generated by the given device associated with the \texttt{isc}. This routine may sleep and thus must be called from a thread context. This routine might allocate hardware resource, so it should be used with care.

The \texttt{flags} parameter should be either \texttt{WSIO_INTR_EXCLUSIVE} if the device driver's ISR cannot be shared, or zero if the ISR can be shared. If the \texttt{WSIO_INTR_EXCLUSIVE} flag is not present, the driver's ISR may be called even if the device did not generate an interrupt. If the device driver has no way of determining if the card needs servicing, the device driver's ISR cannot be shared. For such a card, running the device driver's ISR indicates the card needs servicing.
RETURN VALUES

- **WSIO_OK**: Successful completion.
- **WSIO_ERROR**: Failure; no interrupt services available.
- **WSIO_INTR_INV_FLAG**: Must specify 0 (shared) or **WSIO_INTR_EXCLUSIVE** only.
- **WSIO_MEM_ALLOC_FAILED**: Interrupt services failed to allocate memory.
- **WSIO_PARM_ERROR**: Invalid parameters

CONSTRAINTS

EXAMPLE

```c
/* Allocate a line based interrupt */
wsio_intr_object_t obj;
int status;

/* allocate an interrupt object for a shared interrupt */
status = wsio_intr_alloc(isc, isr,
                        (uintptr_t)isc, 0, &obj);
if (status != WSIO_OK) {
    return(ERROR);
}
```

SEE ALSO

- wsio_intr_free(WSIO3)
NAME

wsio_intr_deactivate(WSIO3) – Disable an interrupt object.

SYNOPSIS

#include <sys/wsio.h>

int wsio_intr_deactivate (struct isc_table_type *isc,
                           wsio_intr_object_t obj);

PARAMETERS

isc Pointer to the driver’s isc_table entry.
obj Interrupt object.

DESCRIPTION

The wsio_intr_deactivate() WSIO function deactivates an interrupt object that was previously activated with wsio_intr_activate(). By deactivating the interrupt object the system will stop calling the device driver’s ISR (as specified in wsio_intr_alloc()). It is assumed that (if possible) the device’s interrupts will be disabled before this function is called. This function will sleep.

Attempting to deactivate an interrupt object that has not been activated is an error condition that returns WSIO_INTR_DEACTIVATED, without modifying the interrupt object.

RETURN VALUES

WSIO_OK Successful completion.
WSIO_INTR_DEACTIVATED obj not active.
WSIO_PARM_ERROR Invalid parameters.
WSIO_ERROR Could not deactivate obj.
CONSTRAINTS

Must not be called in an interrupt context.

EXAMPLE

/* Allocate a line based interrupt and activate it */
wsio_intr_object_t obj;
int status;

/* allocate an interrupt object */
status = wsio_intr_alloc(isc, driver_isr,(uintptr_t)isc, 0, &obj);
if (status != WSIO_OK) {
    return(ERROR);
}

/* Get a Level Sensitive IRQ */
status = wsio_intr_set_irq_line(isc, obj,
                                WSIO_IRQ_LINE_AUTO, 0);
if (status != WSIO_OK) {
    return(ERROR);
}

/* activate the interrupt */
status = wsio_intr_activate(isc, obj);
if (status != WSIO_OK) {
    return(ERROR);
}

..............

/* deactivate the interrupt */
status = wsio_intr_deactivate(isc, obj);
if (status != WSIO_OK) {
    return(ERROR);
}

SEE ALSO

wsio_intr_activate(WSIO3), wsio_intr_alloc(WSIO3),
wsio_intr_deactivate_nowait(WSIO3),
wsio_intr_set_cpu_spec(WSIO3), wsio_intr_set_line(WSIO3),
NAME

wsio_intr_deactivate_nowait(WSIO3) – Disable an interrupt object with callback.

SYNOPSIS

#include <sys/wsio.h>

int wsio_intr_deactivate_nowait (struct isc_table_type *isc,
                                 wsio_intr_object_t obj,
                                 wsio_intr_deact_cb_t callback_func,
                                 uintptr_t arg);

PARAMETERS

isc Pointer to the driver's isc_table entry.
obj Interrupt object.
callback_func A driver callback function that will be executed.
arg Parameter passed back to a driver in the callback.

DESCRIPTION

The wsio_intr_deactivate_nowait() WSIO function is a mirror of wsio_intr_deactivate() except that this one does not sleep. It deactivates an interrupt object that was previously activated with wsio_intr_activate(). By deactivating the interrupt object the system will stop calling the device driver's ISR (as specified in wsio_intr_alloc()). It is assumed that, if possible, the device's interrupts will be disabled before this function is called.

Attempting to deactivate an interrupt object that has not been activated is an error condition that returns WSIO_INTR_DEACTIVATED, without modifying the interrupt object.

This function will return immediately. Upon receiving confirmation that the interrupts have been disabled, the callback_func will be executed passing the arg back to the driver. Since this call will not sleep, it can be called in a non-blocking context.
RETURN VALUES

WSIO_OK Successful completion.
WSIO_INTR_DEACTIVATED
obj not active.
WSIO_ERROR Operation failed.
WSIO_INTR_INV_OBJ
Must call wsio_intr_set_cpu_spec() or
wsio_intr_set_irq_line() first.
WSIO_NO_INTR_CB
No call back function passed in.
WSIO_PARM_ERROR
Invalid parameters.

CONSTRAINTS

EXAMPLE

/* Driver’s callback function for
wsio_intr_deactivate_nowait() */
void my_callback(struct isc_table_type *isc)
{
...........
}

/* Allocate a line based interrupt and activate it */
wsio_intr_object_t obj;
int status;

/* allocate an interrupt object */
status = wsio_intr_alloc(isc, driver_isr,
                        (uintptr_t)isc, 0, &obj);
if (status != WSIO_OK) {
    return(ERROR);
}
/* Get a Level Sensitive IRQ */
status = wsio_intr_set_irq_line(isc, obj,
                              WSIO_IRQ_LINE_AUTO, 0);
if (status != WSIO_OK) {

return(ERROR);
}
/* activate the interrupt */
status = wsio_intr_activate(isc, obj);
if (status != WSIO_OK) {
    return(ERROR);
}

............
/* deactivate the interrupt */
status = wsio_intr_deactivate_nowait(isc,
    obj, my_callback, isc);
if (status != WSIO_OK) {
    return(ERROR);
}

SEE ALSO

wsio_intr_activate(WSIO3), wsio_intr_alloc(WSIO3),
wsio_intr_deactivate(WSIO3), wsio_intr_set_cpu_spec(WSIO3),
wsio_intr_set_line(WSIO3)
NAME

wsio_intr_free(WSIO3) – Free an interrupt object.

SYNOPSIS

#include <sys/wsio.h>

int wsio_intr_free (struct isc_table_type *isc, wsio_intr_object_t obj);

PARAMETERS

isc Pointer to the driver’s isc_table entry.
obj Interrupt object.

DESCRIPTION

Before this function returns, the services guarantee that all outstanding
ISRs have been run to completion. The obj object must not be active
(either never activated, or deactivated with wsio_intr_deactivate(),
or wsio_intr_deactivate_nowait()) when this function is called.
This function may block, so it must be called in a thread context.

RETURN VALUES

WSIO_OK Successful completion.
WSIO_INTR_ACTIVATED Must call wsio_intr_deactivate or
wsio_intr_deactivate_nowait first.
WSIO_PARM_ERROR Invalid parameters.

CONSTRAINTS

Must not be called in an interrupt context.
EXAMPLE

/* Allocate a line based interrupt and activate it */
wsio_intr_object_t obj;
int status;

/* allocate an interrupt object */
status = wsio_intr_alloc(isc, driver_isr,
               (uintptr_t)isc, 0, &obj);
if (status != WSIO_OK) {
    return(ERROR);
}
/* Get a Level Sensitive IRQ */
status = wsio_intr_set_irq_line(isc, obj,
                 WSIO_IRQ_LINE_AUTO, 0);
if (status != WSIO_OK) {
    return(ERROR);
}
/* activate the interrupt */
status = wsio_intr_activate(isc, obj);
if (status != WSIO_OK) {
    return(ERROR);
}
/* deactivate the interrupt */
status = wsio_intr_deactivate(isc, obj);
if (status != WSIO_OK) {
    return(ERROR);
}
/* free the interrupt obj */
status = wsio_intr_free(isc, obj);
if (status != WSIO_OK) {
    return(ERROR);
}
obj = NULL;

SEE ALSO

wsio_intr_alloc(WSIO3)
NAME

wsio_intr_get_assigned_cpu(WSIO3) - Get the currently assigned CPU for the interrupt object.

SYNOPSIS

#include <sys/wsio.h>

int wsio_intr_get_assigned_cpu (wsio_intr_object_t, intptr_t *cpu_spec);

PARAMETERS

obj    Interrupt object.

cpu_spec  CPU specification.

DESCRIPTION

The wsio_intr_get_assigned_cpu() WSIO function returns the currently assigned CPU for the passed interrupt object. The return type for cpu_spec will depend on platform. On PA, it will return assigned CPU in the location pointed by cpu_spec.

RETURN VALUES

WSIO_OK    Successful completion.

WSIO_ERROR  Indicates functionality is not supported on platform.

WSIO_PARM_ERROR  Invalid parameters.

CONSTRAINTS
EXAMPLE

/* Allocate a transaction based interrupt */
wsio_intr_object_t obj;
int status;
intptr_t cpu_spec;

/* allocate an interrupt object */
status = wsio_intr_alloc(isc, driver_isr,
               (uintptr_t)isc, 0, &obj);
if (status != WSIO_OK) {
    return(ERROR);
}

/* Get a TXN based interrupt */
status = wsio_intr_set_cpu_spec(isc, obj, WSIO_INTR_CPU_ANY);
if (status != WSIO_OK) {
    return(ERROR);
}

/* activate the interrupt */
status = wsio_intr_activate(isc, obj);
if (status != WSIO_OK) {
    return(ERROR);
}

/* get the CPU spec */
status = wsio_intr_get_assigned_cpu(obj, &cpu_spec);
if (status != WSIO_OK) {
    return(ERROR);
}

printf("assigned cpu for the interrupt object is = %d\n",
    cpu_spec);

SEE ALSO

wsio_intr_activate(WSIO3), wsio_intr_alloc(WSIO3),
wsio_intr_set_cpu_spec(WSIO3)
NAME

**wsio_intr_get_irq_line(WSIO3)** - Get the interrupt line number.

SYNOPSIS

```c
#include <sys/wsio.h>

int wsio_intr_get_irq_line (struct isc_table_type *isc,
                           wsio_intr_object_t obj,
                           intptr_t *irq_line_num);
```

PARAMETERS

- **isc** Pointer to the driver's *isc_table* entry.
- **obj** Interrupt object.
- **irq_line_num** The interrupt line number.

DESCRIPTION

The `wsio_intr_get_irq_line()` WSIO function returns the line number that the given *obj* is currently using. In most cases this routine is not necessary, as `WSIO_IRQ_LINE_AUTO`, can be given to `wsio_intr_set_irq_line()`. However, if for some reason a device driver needs to know the interrupt line that a card should use, this function provides the necessary data. On a running system the line number assigned to an *obj* may change. For example, when a processor is deallocated, the driver services will reassign the interrupt line on behalf of the driver.

RETURN VALUES

- **WSIO_OK** Successful completion.
- **WSIO_INTR_INV_OBJ** Must call `wsio_intr_set_irq_line()` first.
- **WSIO_ERROR** Failed to get line number.
- **WSIO_PARM_ERROR** Invalid parameters.
CONSTRAINTS

EXAMPLE

/* Allocate a line based interrupt and activate it */
wsio_intr_object_t obj;
int status;
intptr_t irq;

/* allocate an interrupt object */
status = wsio_intr_alloc(isc, driver_isr,
   (uintptr_t)isc, 0, &obj);
if (status != WSIO_OK) {
   return(ERROR);
}

/* Get a Level Sensitive IRQ */
status = wsio_intr_set_irq_line(isc, obj,
   WSIO_IRQ_LINE_AUTO, 0);
if (status != WSIO_OK) {
   return(ERROR);
}

/* activate the interrupt */
status = wsio_intr_activate(isc, obj);
if (status != WSIO_OK) {
   return(ERROR);
}

/* get the line number */
status = wsio_intr_get_irq_line(isc, obj, &irq);
if (status != WSIO_OK) {
   return(ERROR);
}

printf ("my line number is %ld\n", irq);

SEE ALSO

wsio_intr_activate(WSIO3), wsio_intr_alloc(WSIO3),
wsio_intr_set_line(WSIO3)
NAME

wsio_intr_get_txn_info(WSIO3) - Get the transaction address and data value.

SYNOPSIS

#include <sys/wsio.h>

int wsio_intr_get_txn_info (struct isc_table_type *isc,
   wsio_intr_object_t obj,
   intptr_t *txn_addr,
   intptr_t *txn_data);

PARAMETERS

isc Pointer to the driver's isc_table entry.
obj Interrupt object.
txn_addr Transaction address value.
txn_data Transaction data value.

DESCRIPTION

The wsio_intr_get_txn_info() WSIO function returns the transaction address and transaction data associated with a transaction based obj.

RETURN VALUES

WSIO_OK Successful completion.
WSIO_INTR_INV_OBJ Must be a transaction based obj; call wsio_intr_set_cpu_spec() first.
WSIO_ERROR Failed to get values.
WSIO_PARM_ERROR Invalid parameters.

CONSTRAINTS
EXAMPLE

/* Allocate a transaction based interrupt */
wsio_intr_object_t obj;
int status;
intptr_t txn_addr, txn_data;

/* allocate an interrupt object */
status = wsio_intr_alloc(isc, driver_isr,
                        (uintptr_t)isc, 0, &obj);
if (status != WSIO_OK) {
    return(ERROR);
}

/* Get a TXN based interrupt */
status = wsio_intr_set_cpu_spec(isc, obj, WSIO_INTR_CPU_ANY);
if (status != WSIO_OK) {
    return(ERROR);
}

/* activate the interrupt */
status = wsio_intr_activate(isc, obj);
if (status != WSIO_OK) {
    return(ERROR);
}

/* get the TXN values */
status = wsio_intr_get_txn_info(isc, obj,
                                &txn_addr, &txn_data);
if (status != WSIO_OK) {
    return(ERROR);
}
printf ("txn_addr = %ld\n", txn_addr);
printf ("txn_data = %ld\n", txn_data);

SEE ALSO

wsio_intr_activate(WSIO3), wsio_intr_alloc(WSIO3),
wsio_intr_set_cpu_spec(WSIO3)
NAME

wsio_intr_set_cpu_spec(WSIO3) - Initialize and distribute transaction based interrupts.

SYNOPSIS

```c
#include <sys/wsio.h>

int wsio_intr_set_cpu_spec (struct isc_table_type *isc,
                          wsio_intr_object_t obj,
                          intptr_t cpu_spec);
```

PARAMETERS

- `isc` Pointer to the driver’s isc_table entry.
- `obj` Interrupt object.
- `cpu_spec` CPU load balancing specification. If `cpu_spec` is not one of the two flags described below but is a `txn_addr`, the interrupt will be bound to this CPU.

<table>
<thead>
<tr>
<th><code>cpu_spec</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSIO_INTR_CPU_ANY</td>
<td>The services will select any processor. The device driver does not care which processor it is bound to.</td>
</tr>
<tr>
<td>WSIO_INTR_CPU_ANY_UNIQUE</td>
<td>The services will not select the same processor for an <code>obj</code> (i.e., only one <code>obj</code> per processor for this <code>isc</code>).</td>
</tr>
<tr>
<td><code>txn_addr</code></td>
<td>Bind to this CPU address.</td>
</tr>
</tbody>
</table>

DESCRIPTION

The `wsio_intr_set_cpu_spec()` WSIO function is used to initialize and distribute transaction based interrupts.
RETURN VALUES

WSIO_OK   Successful completion.
WSIO_INTR_INV_OBJ   Must be a transaction based obj; call
                    wsio_intr_set_cpu_spec() first.
WSIO_ERROR   Failed to set cpu_spec.
WSIO_EXCLUSIVE_FAILED   Can not get an exclusive interrupt.
WSIO_INTR_ACTIVATED   obj is currently active; must call
                      wsio_intr_deactivate() first.
WSIO_PARM_ERROR   Invalid parameters.

CONSTRAINTS

EXAMPLE

/* Allocate a transaction based interrupt */
wsio_intr_object_t obj;
int status;
intptr_t txn_addr, txn_data;

/* allocate an interrupt object */
status = wsio_intr_alloc(isc, driver_isr,
                        (uintptr_t)isc, 0, &obj);
if (status != WSIO_OK) {
    return(ERROR);
}

/* Get a TXN based interrupt */
status = wsio_intr_set_cpu_spec(isc, obj, WSIO_INTR_CPU_ANY);
if (status != WSIO_OK) {
    return(ERROR);
}

/* activate the interrupt */
status = wsio_intr_activate(isc, obj);
if (status != WSIO_OK) {
    return(ERROR);
/* get the TXN values */
status = wsio_intr_get_txn_info(isc, obj,
       &txn_addr, &txn_data);
if (status != WSIO_OK) {
    return(ERROR);
}
printf ("txn_addr = %ld\n", txn_addr);
printf ("txn_data = %ld\n", txn_data);

SEE ALSO

wsio_intr_activate(WSIO3), wsio_intr_alloc (WSIO3),
wsio_intr_set_cpu_spec (WSIO3)
NAME

wsio_intr_set_irq_line(WSIO3) – Set the interrupt line number.

SYNOPSIS

```c
#include <wsio/wsio.h>

int wsio_intr_set_irq_line (struct isc_table_type *isc,
                           wsio_intr_object_t obj,
                           intptr_t irq_line_num,
                           uint64_t flags);
```

PARAMETERS

isc Pointer to the driver's isc_table entry.

obj Interrupt object.

irq_line_num The interrupt line number, or
               WSIO_IO_INT_LINE_AUTO

flags Zero (level sensitive) or WSIO_INTR_EDGE_SENSITIVE

DESCRIPTION

The `wsio_intr_set_irq_line()` WSIO function is used to setup a line
based obj. The WSIO_INTR_ACTIVATED error code will be returned if this
function is called on an active interrupt object. Most drivers will use
WSIO_IRQ_LINE_AUTO for the value of the irq_line_num parameter
which forces the services to determine the interrupt line value for the
particular device.

If the device generates level-sensitive interrupts, the flags parameter
should be zero. If the device generates edge-sensitive interrupts, the
flags parameter should be WSIO_INTR_EDGE_SENSITIVE.

RETURN VALUES

WSIO_OK Operation succeeded.

WSIO_INTR_INV_OBJ Must not be a transaction based interrupt.
wsio_intr_set_irq_line(WSIO)

WSIO_INTR_ACTIVATED
Object is active; do not call wsio_intr_activate() first.

WSIO_ERROR
Failed to set line number.

WSIO_INTR_INV_FLAG
Must be zero (level) or WSIO_INTR_EDGE_SENSITIVE.

WSIO_PARM_ERROR
Invalid parameters.

CONSTRAINTS

EXAMPLE

/* Allocate a line based interrupt */
wsio_intr_object_t obj;
int status;

/* allocate an interrupt object */
status = wsio_intr_alloc(isc, driver_isr,
    (uintptr_t)isc, 0, &obj);
if (status != WSIO_OK) {
    return(ERROR);
}

/* Get a Level Sensitive IRQ */
status = wsio_intr_set_irq_line(isc, obj,
    WSIO_IRQ_LINE_AUTO, 0);
if (status != WSIO_OK) {
    return(ERROR);
}

SEE ALSO

wsio_intr_activate(WSIO3), wsio_intr_alloc(WSIO3),
wsio_intr_set_line(WSIO3)
NAME

wsio_io_sync(WSIO3) – Perform a sync of shared memory if necessary.

SYNOPSIS

```c
#include <wsio/wsio.h>

void wsio_io_sync (struct isc_table_type * isc);
```

PARAMETERS

isc Pointer to the driver's isc_table entry.

DESCRIPTION

The `wsio_io_sync()` WSIO function synchronizes a device's view and the host's view of memory. This functionality can also be achieved by doing a PIO read of a card register, but in some implementations this function may provide a lower latency mechanism.

`wsio_io_sync()` can be called in a non-blocking context.

RETURN VALUES

None

CONSTRAINTS

EXAMPLE

```c
wsio_io_sync(isc_entry);
```

SEE ALSO

WSIO_ORDERED_INTERRUPTS(WSIO3)
NAME

wsio_iova_to_phys(WSIO3) - Translate an I/O virtual address to a physical address.

SYNOPSIS

#include <sys/wsio.h>

caddr_t wsio_iova_to_phys (struct isc_table_type *isc,
                          void *dma_handle, wsio_iova_t iova);

PARAMETERS

isc    Pointer to the driver's isc_table entry.
dma_handle DMA handle allocated using
           wsio_allocate_dma_handle().
iova   I/O virtual address to be translated.

DESCRIPTION

The wsio_iova_to_phys() WSIO function is called by a device driver to translate an I/O virtual address to a physical address.

wsio_iova_to_phys() can be called in a non-blocking context.

RETURN VALUES

The physical address corresponding to iova or NULL if a translation does not exist.

CONSTRAINTS
**EXAMPLE**

caddr_t phys_address = NULL;

phys_address =
wsio_iova_to_phys(isc_entry,dma_handle,io_address);

/* The 'phys_address' variable will now contain the physical
* address that 'iova' translates to. */

**SEE ALSO**

wsio_allocate_dma_handle(WSIO3),
wsio_allocate_shared_mem(WSIO3), wsio_dma_pass_thru(WSIO3),
wsio_fastmap_dma_buffer(WSIO3), wsio_free_dma_handle(WSIO3),
wsio_free_shared_mem(WSIO3), wsio_flush_shared_mem(WSIO3),
wsio_init_map_context(WSIO3), wsio_map_dma_buffer(WSIO3),
wsio_remap_dma_buffer(WSIO3),
wsio_set_device_attributes(WSIO3),
wsio_unmap_dma_buffer(WSIO3).
NAME

wsio_map_dma_buffer(WSIO3) - Map an existing memory object for packet DMA.

SYNOPSIS

#include <wsio/wsio.h>

wsio_map_status_t wsio_map_dma_buffer (  
    struct isc_table_type *isc,  
    void *dma_handle,  
    wsio_map_context_t *context,  
    wsio_dma_buffer_hints_t hints,  
    wsio_range_type_t range_type,  
    wsio_dma_map_t *host_range,  
    wsio_dma_map_t *io_range);

PARAMETERS

isc Pointer to the driver's isc_table entry.

dma_handle DMA handle allocated using wsio_allocate_dma_handle().

context Pointer to the context used for mapping.

hints Bitmask that provides mapping hints. The allowable hints are as follows:

<table>
<thead>
<tr>
<th>hints</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSIO_DMA_SAFE</td>
<td>Forces coherent transactions to be used even for full-cacheline transactions. In some implementations semi-coherent transactions are used to enhance performance when it is known that the entire cacheline will be modified. Affects inbound DMA only.</td>
</tr>
</tbody>
</table>
WSIO_DMA_LOCK
In some implementations this allows atomic access to memory for devices using bus-lock primitives.

WSIO_DMA_FLUSH_ON_USE
In some implementations this hint tells the hardware to flush resources associated with this buffer after they are used. This inhibits coalescing transactions into larger transfers for cases where it is not beneficial.

WSIO_DMA_IGN_ALIGNMENT
Indicates map_dma_buffer should not automatically set the IO_SAFE hint for partial cacheline buffers.

WSIO_DMA_CONTIGUOUS
Indicates map_dma_buffer must allocate a single contiguous IOVA. If wsio_map_dma_buffer is unable to do this, it will return WSIO_MAP_E_PARAMETER_ERROR. This hint implies WSIO_IO_IGN_ALIGNMENT.

WSIO_DMA_NO_SEQ
Inhibits hardware prefetching for outbound DMA in some implementations.

WSIO_DMA_INBOUND
Indicates the buffer will be used exclusively for inbound DMA.

WSIO_DMA_OUTBOUND
Indicates the buffer will be used exclusively for outbound DMA.

WSIO_DMA_NULL
Forces all hint values to zero. A hint value of zero tells the BN-CDIO to take hint values from the DMA object.
### DESCRIPTION

The `wsio_map_dma_buffer()` WSIO function is called by a device driver to map an existing memory object for packet DMA. If continuous DMA is required, `wsio_allocate_shared_mem()` should be used. Continuous DMA should be used for control structures, circular buffers, or any kind of buffer that needs to be accessed on a continuous basis by the I/O device. Packet DMA should be used when the mappings are temporary, or when pre-existing memory objects must be mapped for DMA.

If the entire host range cannot be mapped, the buffer may only be partially mapped. This will be indicated by a return value of `WSIO_MAP_W_PARTIAL`. In this case, `wsio_map_dma_buffer()` will need to be called again to map more of the host range. Using this method of calling `wsio_map_dma_buffer()` multiple times, the entire host range can be mapped into multiple I/O virtual ranges.

All mappings remain in effect until `wsio_unmap_dma_buffer()` or `wsio_remap_dma_buffer()` are called to remove or change them.

Callers are guaranteed buffers of at least 4K Bytes can be mapped into a contiguous range of I/O virtual address.

---

<table>
<thead>
<tr>
<th><code>range_type</code></th>
<th>Indicates the type of host memory being mapped. It can be:</th>
</tr>
</thead>
<tbody>
<tr>
<td>KERNELSPACE</td>
<td>Indicates <code>host_range</code> is a kernel virtual buffer.</td>
</tr>
<tr>
<td>PHYSICAL</td>
<td>Indicates <code>host_range</code> is a physical buffer.</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Indicates <code>host_range</code> is in user space, and this will be the space ID of the virtual address.</td>
</tr>
</tbody>
</table>

| `host_range` | Pointer to an address/length structure that contains information about the host space to map. If the mapping was only partially completed, this will contain information about the remaining space to be mapped when the call completes. |

| `io_range` | Pointer to an address/length structure that will contain information about the I/O space that was mapped. |
The `context` parameter is used to insure I/O resources are used efficiently. This structure should be initialized via a call to `wsio_init_map_context()`. The same context should be used for any group of mappings that will all be unmapped at the same time. If multiple buffers will not be mapped into one context, `NULL` can be passed in instead of a valid context.

Device drivers can set up a callback routine that will come into play if resources are not available at the time a mapping is attempted. If this callback is set up, `WSIO_MAP_W_CALLBACK` will be returned instead of a no resource error. When resources become available, the callback routine will be called to indicate this to the device driver. For more information on how to setup and use a callback, see the `wsio_set_dma_callback()` manpage.

`wsio_map_dma_buffer()` can be called in a non-blocking context.

**RETURN VALUES**

- `WSIO_MAP_OK` Returned if the entire buffer has been mapped.
- `WSIO_MAP_W_PARTIAL` Returned if only part of the buffer has been mapped.
- `WSIO_MAP_W_CALLBACK` Returned if no resources are available and a callback function exists.
- `WSIO_MAP_E_NO_RESOURCES` Returned if no resources are available and no callback function exists.
- `WSIO_MAP_E_RESOURCE_ERROR` Returned if the request cannot and will never succeed.
- `WSIO_MAP_E_HIGH_ADDR` Returned if the call failed because the device cannot reach the destination address.
- `WSIO_MAP_E_PARAMETER_ERROR` Returned if an invalid parameter has caused failure of the call.
- `WSIO_MAP_E_UNKNOWN_ERROR` Returned for hardware or other errors.
CONSTRAINTS

EXAMPLE

```c
void *dma_handle;
wsio_map_context_t dma_con;
wsio_dma_map_t host_range, io_range;

dma_handle = wsio_allocate_dma_handle(isc_entry);
wsio_init_map_context(&dma_con);

host_range.iov_base = host_virtual_address;
host_range.iov_len = dma_buffer_length;

if (wsio_map_dma_buffer(isc_entry,dma_handle,dma_con,
    WSIO_DMA_OUTBOUND,KERNELSPACE,&host_range,&io_range
)! = WSIO_MAP_OK) {
    /* Unable to perform the mapping so return an error */
    return(ERROR);
}

/* The host virtual buffer represented in the above code by host_virtual_address’ is now mapped. Note that this code does not handle the case where the buffer is only partially mapped. In that case, a more complete example would call wsio_map_dma_buffer() again each time WSIO_MAP_PARTIAL was returned and save each io_range that was returned. */
```

SEE ALSO

wsio_allocate_dma_handle(WSIO3),
wsio_allocate_shared_mem(WSIO3),wsio_dma_pass_thru(WSIO3),
wsio_fastmap_dma_buffer(WSIO3),wsio_free_dma_handle(WSIO3),
wsio_free_shared_mem(WSIO3),wsio_flush_shared_mem(WSIO3),
wsio_init_map_context(WSIO3),wsio_iova_to_phys(WSIO3),
wsio_remap_dma_buffer(WSIO3),
wsio_set_device_attributes(WSIO3),
wsio_set_dma_attributes(WSIO3),
wsio_unmap_dma_buffer(WSIO3),
NAME

wsio_mem_alloc_attrib_t(WSIO3) – Memory allocation attributes

SYNOPSIS

#include <sys/wsio.h>

typedef int wsio_mem_alloc_attrib_t;

PARAMETERS

WSIO_OPTIMIZE_FOR_DEVICE
    Allocate memory close to the device.

WSIO_OPTIMIZE_FOR_CPU
    Allocate memory close to the current CPU. This is the default behavior

WSIO_32BIT_MEMORY
    The buffer must be allocated before 4 GB.

WSIO_IO_CONTIGUOUS
    On platforms without an IOPDIR physically contiguous memory will be allocated.

WSIO_ALIGN_ON_SIZE
    With this attribute, buffers will be aligned with the same alignment as their size.

DESCRIPTION

Flags passed to the service wsio_alloc_mem_handle() to indicate what type of memory will be allocated with the handle.

EXAMPLE

SEE ALSO
NAME

WSIO_ORDERED_INTERRUPTS(WSIO3) – Macro to indicate whether interrupts are ordered with respect to DMA transactions.

SYNOPSIS

#include <sys/wsio.h>

int WSIO_ORDERED_INTERRUPTS (struct isc_table_type * isc);

PARAMETERS

isc Pointer to the driver's isc_table entry.

DESCRIPTION

The WSIO_ORDERED_INTERRUPTS() macro reports to a device driver about whether interrupts are ordered with respect to DMA transactions. If true (1) is returned, nothing needs to be done. However, if false (0) is returned, interrupts are not ordered, and drivers must ensure DMA transactions have completed by reading a status register, or by calling wsio_io_sync().

WSIO_ORDERED_INTERRUPTS() can be called in a non-blocking context.

RETURN VALUES

Returns a one (1) if interrupts are ordered with respect to DMA transactions, and a zero (0) otherwise.

CONSTRAINTS

EXAMPLE

if (WSIO_ORDERED_INTERRUPTS(isc_entry)) {
    wsio_io_sync(isc_entry);
} else {
    /* No syncing necessary */
}
SEE ALSO

wsio_io_sync(WSIO3)
NAME

wsio_query_supported_function(WSIO_DRV) - Return a pointer to a function supported by WSIO.

SYNOPSIS

#include <sys/wsio.h>

void * wsio_query_supported_function (wsio_func_ptr_type_t func_id);

PARAMETERS

func_id Identifies what function pointer a driver is interested in.

<table>
<thead>
<tr>
<th>func_id</th>
<th>Function pointers returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSIO_GET_HANDLER_REG_FUNC</td>
<td>A WSIO function to register a driver’s event handler.</td>
</tr>
<tr>
<td>WSIO_GET_HANDLER_UNREG_FUNC</td>
<td>A WSIO function to unregister a driver’s event handler.</td>
</tr>
<tr>
<td>WSIO_GET_INSTALL_DRV_FUNC</td>
<td>A WSIO function to register a driver’s function.</td>
</tr>
<tr>
<td>WSIO_GET_MASK_REG_FUNC</td>
<td>A WSIO function to register a driver’s supported event mask.</td>
</tr>
</tbody>
</table>

DESCRIPTION

The wsio_query_supported_function() WSIO function queries WSIO to see if certain functions are supported. The purpose is that a driver can be installed on systems with different versions of wsio(11i or newer.) By doing this query, a driver can determine what functionality is supported and tailor its capabilities. If a function is supported, a pointer will be returned (otherwise a NULL); a driver can then call this function pointer. This mechanism will allow a driver to be able to link on multiple versions of wsio(11i or newer.)
RETURN VALUES

Returns a function pointer if supported, otherwise a NULL.

CONSTRAINTS

EXAMPLE

```c
static wsio_drv_info_t my_drv_info {
    ....
}
void my_event_handler()
{
    ....
}
int my_install(void)
{
    wsio_install_handler_func_ptr_t handler_reg_ptr;

    if (wsio_install_driver(&my_drv_info) {
        /* Query wsio to get a pointer to its registration function */
        handler_reg_ptr = wsio_query_supported_function(WSIO_GET_HANDLER_REG_FUNC);
        if (handler_reg_ptr) {
            /* Now register our event handler with wsio */
            ret = handler_reg_ptr(&my_drv_info,
                                my_event_handler);
            ....
        }
    }
    ....
}
```

SEE ALSO

wsio_install_drv_event_handler(WSIO3),
wsio_install_drv_func(WSIO3),
wsio_reg_drv_capability_mask(WSIO3),
wsio_uninstall_drv_event_handler(WSIO3)
NAME

wsio_remap_dma_buffer(WSIO3) – Map pre-allocated IOVAs to new host ranges.

SYNOPSIS

#include <sys/wsio.h>

wsio_map_status_t wsio_remap_dma_buffer (  
    struct isc_table_type * isc,  
    void * dma_handle,  
    wsio_range_type_t range_type,  
    wsio_dma_map_t * host_range,  
    wsio_dma_map_t * io_range);

PARAMETERS

isc Pointer to the driver’s isc_table entry.

dma_handle DMA handle allocated using  
    wsio_allocate_dma_handle().

range_type Indicates the type of host memory being mapped. It can be:
    KERNELSPACE Indicates host_range is a kernel  
        virtual buffer.
    PHYSICAL Indicates host_range is a physical  
        buffer.
    >0 Indicates host_range is in user  
        space, and this will be the space ID of  
        the virtual address.

host_range Pointer to an address/length structure that contains  
    information about the host space to map. If the  
    mapping was only partially completed, this will contain  
    information about the remaining space to be mapped  
    when the call completes.

io_range Pointer to an address/length structure that will contain  
    information about the I/O space that was mapped.
DESCRIPTION

The `wsio_remap_dma_buffer()` WSIO function is called by a device driver to map a new host memory address to existing I/O virtual addresses (IOVA). The IOVAs must have been previously allocated via a call to `wsio_map_dma_buffer()`, `wsio_remap_dma_buffer()`, or `wsio_fastmap_dma_buffer()`.

The `io_range` must use exactly the same number of mapping resources as the previous mapping. This can be ensured by making sure the buffers are page-aligned and of equal sizes.

All mappings will remain in effect until `wsio_unmap_dma_buffer()` or `wsio_remap_dma_buffer()` are called to remove or change them.

A callback function is not necessary for `wsio_remap_dma_buffer()` because DMA resources were allocated when the initial mapping took place.

`wsio_remap_dma_buffer()` can be called in a non-blocking context.

RETURN VALUES

- **WSIO_MAP_OK**  
  Returned if the entire buffer has been mapped.

- **WSIO_MAP_E_HIGH_ADDR**  
  Returned if the call failed because the device cannot reach the destination address.

- **WSIO_MAP_E_PARAMETER_ERROR**  
  Returned if an invalid parameter has caused failure of the call.

- **WSIO_MAP_E_UNKNOWN_ERROR**  
  Returned for hardware or other errors.

CONSTRAINTS
EXAMPLE

```c
void *dma_handle;
wsio_map_context_t dma_con;
wsio_dma_map_t host_range, new_host_range, io_range;

dma_handle = wsio_allocate_dma_handle(isc_entry);
wsio_init_map_context(&dma_con);

host_range.iov_base = host_virtual_address;
host_range.iov_len = dma_buffer_length;

if (wsio_map_dma_buffer(isc_entry,dma_handle,
    dma_con,WSIO_DMA_OUTBOUND KERNELSPACE,
    &host_range,&io_range)!= WSIO_MAP_OK) {
    return(ERROR);
}

new_host_range.iov_base = new_host_virtual_address;
new_host_range.iov_len = dma_buffer_length;

if (wsio_remap_dma_buffer(isc_entry,dma_handle,
    KERNELSPACE,&new_host_range, &io_range
    ) != WSIO_MAP_OK) {
    return(ERROR);
}

/* The host virtual buffer represented in the above
 * code by ‘host_virtual_address’ was mapped. After that,
 * a second host virtual buffer represented by
 * ‘new_host_virtual_address’ was mapped using the
 * same I/O virtual address as the initial mapping.
 * This example doesn’t take into account a
 * WSIO_MAP_W_PARTIAL return code being returned by the
 * call to wsio_map_dma_buffer().
 */
```
SEE ALSO

wsio_allocate_dma_handle(WSIO3),
wsio_allocate_shared_mem(WSIO3), wsio_dma_pass_thru(WSIO3),
wsio_fastmap_dma_buffer(WSIO3), wsio_free_dma_handle(WSIO3),
wsio_free_shared_mem(WSIO3), wsio_flush_shared_mem(WSIO3),
wsio_init_map_context(WSIO3), wsio_iova_to_phys(WSIO3),
wsio_map_dma_buffer(WSIO3),
wsio_set_device_attributes(WSIO3),
wsio_set_dma_attributes(WSIO3), wsio_unmap_dma_buffer(WSIO3)
NAME

wsio_reg_drv_capability_mask(WSIO3) - Register a driver’s capability mask.

SYNOPSIS

#include <sys/wsio.h>

int wsio_reg_drv_capability_mask (struct isc_table_type *isc,
                                 wsio_event_mask_t event_mask);

PARAMETERS

isc Pointer to the driver’s isc_table entry.

event_mask A mask representing the operations the driver will deal with in its event handler.

DESCRIPTION

The wsio_reg_drv_capability_mask() WSIO function is called by a device driver to register with WSIO what operations the driver is capable of handling. The mask is an OR of wsio_event_t and is for an instance of the driver. This function should be called in a driver’s attach routine after isc_claim() is executed. This function can be called more than once to disable and enable certain functionalities. A new mask will supersede a previous one. If a platform does not support event handling, WSIO_HA_NA will be returned. In this case, a driver should ignore the error.

RETURN VALUES

WSIO_OK Returned on success.

WSIO_INVALID_COMBIN_EVENTS Invalid combination of events.

WSIO_HA_NA Platform does not support HA events.

WSIO_INVALID_EVENT Invalid event.
WSIO_INVALID_ISC
Invalid isc

WSIO_NO_DRV_HANDLER
A driver's event handler has not been installed.

CONSTRAINTS

EXAMPLE

my_driver_attach() {
    int ret;
    wsio_event_mask_t my_mask = WSIO_EVENT_SUSPEND | WSIO_EVENT_RESUME;
    isc_claim(isc, &my_drv_info);
    ret = wsio_reg_drv_capability_mask(isc, my_mask);
    if ( (ret != WSIO_OK) && (ret != WSIO_HA_NA) ) {
        isc_unclaim(isc, &my_drv_info);
        free resource as needed
    }
    run the rest of the attach chain
}

SEE ALSO

wsio_install_drv_event_handler(WSIO3),
wsio_query_supported_function(WSIO_DRV)
NAME

wsio_set_dma_attributes(WSIO3) - Associate DMA hints with a DMA handle.

SYNOPSIS

#include <sys/wsio.h>

wsio_map_status_t wsio_set_dma_attributes (    
    struct isc_table_type* isc,            
    void* dma_handle,                   
    wsio_dma_attribute_t attribute,      
    wsio_dma_attr_param_t param);

PARAMETERS

isc        Pointer to the driver’s isc_table entry.
dma_handle DMA handle allocated using      
            wsio_allocate_dma_handle().
attribute   Indicates which hint to set for the device associated with dma_handle. The possible attributes are:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
</table>
| WSIO_DMA_ATTR_ADDR_WIDTH | Bits of addressing supported by the device. This is used to determine whether a device can DMA directly to memory buffers  
                         | Default = 32                                                                |
| WSIO_DMA_ATTR_ALIGNMENT | Byte alignment of DMA buffer required for device.                           
                         | Default = H/W Dep.                                                         |
WSIO_DMA_ATTR_ATM
ATM hint; used by hardware in some implementations.

   0 = not ATM
   1 = ATM48 (optimize for 48-byte transfers)
   2 = ATM192 (optimize for 192-byte transfers)

Default = 0

WSIO_DMA_ATTR_BANDWIDTH
Specifies the bandwidth of the card in MB/Sec. Valid values are: 100, 200, 400, and 800.

WSIO_DMA_ATTR_BLK_SIZE
Indicates the size in number of bytes of the average block of data used in DMA by the device. It is used to set the prefetch depth on platforms that have the hardware support.

WSIO_DMA_ATTR_CALLBACK
Specifies a function to call when resources become available.

   Default = NULL

WSIO_DMA_ATTR_CALLBACK_ARG
Specifies an argument to the callback function.

   Default = 0

WSIO_DMA_ATTR_FLUSH_ON_USE
 Specifies the cacheline should be flushed from any intermediate buffers as soon as it is referenced. This inhibits any coalescing of data by bus bridges.

   Default = 0
WSIO_DMA_ATTR_IGN_ALIGN
Specifies the mapping service should not handle cacheline fragments in a special way.
Default = 0

WSIO_DMA_ATTR_INTERLEAVE
IOVA allocation model
0 = DMA streams are normally interleaved (mass storage)
1 = DMA streams are normally not interleaved (networking)
Default = 0

WSIO_DMA_ATTR_PREFETCH
Specifies how aggressively hardware should prefetch for outbound DMA.
0 = no prefetch
1 = moderate prefetch
2 = aggressive prefetch
Default = 1

WSIO_DMA_ATTR_SAFE
Specifies that the most conservative coherency model should be used for inbound DMA. Inhibits semi-coherent transactions such as WRITE_PURGE unless it is guaranteed that no data in processor caches will be lost.
1 = ON
2 = OFF
Default = 0

WSIO_DMA_ATTR_TXN_SIZE
Specifies the default transaction size used by the device. This is used by hardware to optimize conversion of transactions between busses.
Default = H/W Dep.
WSIO_DMA_ATTR_INBOUND
DMA buffers will be used exclusively for inbound DMA.
Default = 0

WSIO_DMA_ATTR_OUTBOUND
DMA buffers will be used exclusively for outbound DMA.
Default = 0

WSIO_DMA_ATTR_STABLE
Indicates that the buffer will not be modified by another entry while mapped for DMA.

1 = true for data buffers
0 = false for control structures

Default = 0

param Value to set attribute to.

DESCRIPTION
The wsio_set_dma_attributes() WSIO function is used to associate DMA transaction hints and attributes with a specific DMA handle. These hints override any hints set via wsio_dma_set_device_attributes(), and are overridden by some hints passed in as parameters to wsio_map_dma_buffer(). wsio_set_dma_attributes() can be called in a non-blocking context.

If WSIO_DMA_ATTR_INTERLEAVE is set to 1, a subsequent wsio_allocate_shared_mem() or wsio_map_dma_buffer() can only successfully request a buffer of a maximum of one page(4K) in length and this buffer cannot cross a page boundary. This is the limitation placed by the underlying platform. If a larger buffer is desirable, use the default value of 0. This larger buffer can be used for control structures rather than packet DMAs.

Do not call wsio_set_dma_attributes() to set the WSIO_DMA_ATTR_INTERLEAVE to the default value of 0. The call will fail.
RETURN VALUES

WSIO_MAP_OK Successful completion.

WSIO_MAP_E_PARAMETER_ERROR

Returned if an invalid parameter has caused failure of the call.

CONSTRAINTS

EXAMPLE

```c
if (wsio_set_dma_attributes(isc_entry, dma_handle, 
    WSIO_DMA_ATTR_INTERLEAVE, 1) != WSIO_MAP_OK) {
    /* There was a parameter error */
    return(ERROR);
} else {
    /* DMA streams are now not normally interleaved for 
     * all DMA associated with dma_handle 
     */
    return(0);
}
```

SEE ALSO

wsio_allocate_dma_handle(WSIO3),
wsio_allocate_shared_mem(WSIO3), wsio_dma_pass_thru(WSIO3),
wsio_dma_set_device_attributes(WSIO3),
wsio_fastmap_dma_buffer(WSIO3), wsio_free_dma_handle(WSIO3),
wsio_free_shared_mem(WSIO3), wsio_flush_shared_mem(WSIO3),
wsio_init_map_context(WSIO3), wsio_iova_to_phys(WSIO3),
wsio_map_dma_buffer(WSIO3), wsio_remap_dma_buffer(WSIO3),
wsio_unmap_dma_buffer(WSIO3)
NAME

wsio_set_dma_callback(WSIO3) – Set the callback function and argument for DMA

SYNOPSIS

#include <wsio/wsio.h>

wsio_map_status_t wsio_set_dma_callback (struct isc_table_type *isc, 
void *dma_handle, 
void *func, 
void *arg);

PARAMETERS

isc Pointer to the driver’s isc_table entry.
dma_handle DMA handle allocated using wsio_allocate_dma_handle.
func Function pointer to be used as a callback.
arg Argument to be passed to the callback when it is called.

DESCRIPTION

The wsio_set_dma_callback() WSIO function is called by a device driver to setup the callback function for certain DMA transactions. If resources are not available when wsio_map_dma_buffer(), wsio_fastmap_dma_buffer(), or wsio_allocate_shared_memory() are called, and a callback function is setup, WSIO_MAP_W_CALLBACK is returned to the caller, and the callback function will be called when resources become available. This eliminates the need to continuously loop to attempt to obtain DMA resources.

RETURN VALUES

WSIO_MAP_OK Successful completion.
WSIO_MAP_E_PARAMETER_ERROR Returned if an invalid parameter has caused failure of the call.
Manpages

wsio_set_dma_callback(WSIO3)

CONTRAINSTS

EXAMPLE

if (wsio_set_dma_callback(isc_entry, dma_handle,
                        callback_func, callback_arg)
    != WSIO_MAP_OK) {
    /* There was a parameter error */
    return(ERROR);
}

SEE ALSO

wsio_allocate_dma_handle(WSIO3),
wsio_allocate_shared_mem(WSIO3), wsio_dma_pass_thru(WSIO3),
wsio_fastmap_dma_buffer(WSIO3), wsio_free_dma_handle(WSIO3),
wsio_free_shared_mem(WSIO3), wsio_flush_shared_mem(WSIO3),
wsio_init_map_context(WSIO3), wsio_map_dma_buffer(WSIO3),
wsio_remap_dma_buffer(WSIO3),
wsio_set_device_attributes(WSIO3),
wsio_set_dma_attributes(WSIO3) wsio_unmap_dma_buffer(WSIO3),
NAME

wsio_setup_dma32(WSIO3) - Allocates a DMA buffer below 4 GB

SYNOPSIS

#include<sys/wsio.h>

int wsio_alloc_status_t
wsio_setup_dma32 (struct isc_table_type *isc,
                    void *drv_buf,
                    void **tmp_buf,
                    size_t size,
                    wsio_dma_buffer_hints_t trans)

PARAMETERS

isc                  Pointer to the device isc structure.
drv_buf              Pointer to the driver’s buffer.
tmp_buf              Pointer to a pointer; the address of the temporary buffer will be saved in the address referenced by the pointer. The buffer allocated by the service.
size                 The size of the buffer in bytes
trans                Indicates the direction of the DMA action:
                      WSIO_DMA_INBOUND  Writing to memory
                      WSIO_DMA_OUTBOUND Reading from memory

If (trans == WSIO_DMA_OUTBOUND) and a temporary buffer was allocated, the service will copy the data from the original driver buffer to the temporary one.

DESCRIPTION

This service allocates a temporary buffer, and if DMA is outbound, copies the data from the driver’s buffer to the temporary one.

This service does not block. If resources can not be obtained, it returns to the caller with an out-of-resource status. When called, this service always assumes buffering is required, and allocates a temporary buffer.
Drivers should only call this routine if they know they can not DMA directly to the driver buffer.

The WSIO DMA mapping services return the error status of MAP_E_HIGH_ADDR if a driver attempts to map a buffer that it can not DMA directly to.

**RETURN VALUES**

- **WSIO_BUFFER_ALLOCATED**
  - No buffering is required.
- **WSIO_ALLOC_OUT_OF_RESOURCES**
  - The service was unable to allocate a buffer.

**CONSTRAINTS**

**EXAMPLE**

```c
/* Allocates a DMA buffer for outbound DMA */

wsio_alloc_status_t status;
void *tmp_buffer = NULL;

status = wsio_setup_dma32(my_isc, data_buffer, tmp_buffer, buf_size, wsio_dma_outbound);
if (status == WSIO_ALLOC_OK){
    /* do DMA */
    ............
}
```

**SEE ALSO**

wsio_cleanup_dma32()
NAME

wsio_unmap_dma_buffer(WSIO3) – Remove a DMA packet mapping.

SYNOPSIS

```c
#include <sys/wsio.h>

wsio_map_status_t wsio_unmap_dma_buffer (  
    struct isc_table_type* isc,  
    void* dma_handle,  
    wsio_dma_map_t* io_range);
```

PARAMETERS

- **isc** Pointer to the driver’s `isc_table` entry.
- **dma_handle** DMA handle allocated using `wsio_allocate_dma_handle()`.
- **io_range** Pointer to an address/length structure that contains the information about the mapping to remove.

DESCRIPTION

The `wsio_unmap_dma_buffer()` WSIO function is called by a device driver to remove a packet DMA memory mapping, and to free all resources associated with such a mapping. `io_range` must have been obtained via a previous call to `wsio_map_dma_buffer()`, `wsio_remap_dma_buffer()`, or `wsio_fastmap_dma_buffer()`. `wsio_unmap_dma_buffer()` can be called in a non-blocking context.

RETURN VALUES

- **WSIO_MAP_OK** Successful completion.
- **WSIO_MAP_E_PARAMETER_ERROR** Returned if an invalid parameter has caused failure of the call. The buffer will not be unmapped.
CONSTRAINTS

EXAMPLE

```c
void *dma_handle;
wsio_map_context_t dma_con;
wsio_dma_map_t host_range, new_host_range, io_range;

dma_handle = wsio_allocate_dma_handle(isc_entry);
wsio_init_map_context(&dma_con);

host_range.iov_base = host_virtual_address;
host_range.iov_len = dma_buffer_length;

if (wsio_map_dma_buffer(isc_entry, dma_handle,
                         dma_con, WSIO_DMA_OUTBOUND, KERNELSPACE, &host_range,
                         &io_range) != WSIO_MAP_OK) {
    return(ERROR);
}

if (wsio_unmap_dma_buffer(isc_entry, dma_handle, &io_range)
    != WSIO_MAP_OK) {
    /* There must have been a parameter error. */
    return(ERROR);
}

/* The host virtual buffer represented in the above
 * code by 'host_virtual_address' was mapped and then
 * immediately unmapped. This example doesn’t take into
 * account a WSIO_MAP_W_PARTIAL return code being
 * returned by the call to wsio_map_dma_buffer(). */
```
SEE ALSO

wsio_allocate_dma_handle(WSIO3),
wsio_allocate_shared_mem(WSIO3), wsio_dma_pass_thru(WSIO3),
wsio_fastmap_dma_buffer(WSIO3), wsio_free_dma_handle(WSIO3),
wsio_free_shared_mem(WSIO3), wsio_flush_shared_mem(WSIO3),
wsio_init_map_context(WSIO3), wsio_iova_to_phys(WSIO3),
wsio_map_dma_buffer(WSIO3), wsio_remap_dma_buffer(WSIO3),
wsio_set_device_attributes(WSIO3),
wsio_set_dma_attributes(WSIO3)
Manpages

wsio_unmap_dma_buffer(WSIO3)