masd_get_data

Name

`masd_get_data -` Retrieve data from a port.

Synopsis

```
#include “mas/mas_dpi.h”
int32 masd_get_data(int32 portnum, struct mas_data** data);
```

Description

Retrieves the next mas_data struct from the data queue on the specified port, resetting the-queue’s head and clearing mas_data’s "next" member. Use this function to retrieve data from a port.

Return value

Returns

- 0 on success
- MERR_NULLPTR if there’s no data in the port
- MERR_NOTDEF if the specified port number isn’t defined

Examples

See Also

masd_post_data

Name

`masd_post_data` - Posts data to a port.

Synopsis

```
#include “mas/mas_dpi.h”
int32 masd_post_data(int32 portnum, struct mas_data* data);
```

Description

Adds the mas_data struct to the end of the specified port’s data queue. Use this function to put data in a port.

Return value

Returns

- 0 on success
- MERR_NOTDEF if the specified port number isn’t defined

Examples

See Also

masd_get_port_by_name

Name

`masd_get_port_by_name` - Gets a port number from a port name string.

Synopsis

```
#include “mas/mas_dpi.h”

int32 masd_get_port_by_name(int32 device_instance, char* name, int32*
retval_portnum);
```

Description

Uses "retval_portnum" to return the port number of a port on the given device with the given name. If device_instance is -1, then the name is assumed to be unique over all devices and the first instance of the name in the master port list is returned. The caller must allocate the memory to hold the value pointed to by retval_portnum.

Return value

Returns

- 0 on success
- MERR_NOTDEF if the specified port number isn’t defined

Examples

See Also

masd_get_cmatrix_from_name

Name

`masd_get_cmatrix_from_name` - Gets the named characteristic matrix.

Synopsis

```
#include “mas/mas_dpi.h”

int32 masd_get_cmatrix_from_name(int32 device_instance, char*
cmatrix_name, struct mas_characteristic_matrix** retval_cmatrix );
```

Description

Uses "retval_cmatrix" to return a characteristic matrix of the given device with the given name.

Return value

Returns

- 0 on success
- MERR_NOTDEF if the specified device instance isn’t defined

Examples

See Also

masd_set_port_type

Name

`masd_set_port_type` - Sets a port's type.

Synopsis

```
#include "mas/mas_dpi.h"

int32 masd_set_port_type(int32 portnum, int16 type);
```

Description

Set the type of the specified port to one of: MAS_SOURCE, MAS_SINK, MAS_REACTION, MAS_RESPONSE.

Return value

Returns

- 0 on success
- MERR_NOTDEF if the specified port number isn't defined

Examples

See Also

masd_set_port_name

Name

`masd_set_port_name` - Sets a port's name.

Synopsis

```
#include "mas/mas_dpi.h"

int32 masd_set_port_name(int32 portnum, char* name);
```

Description

Set the name string of a port.

Return value

Returns

- 0 on success
- MERR_NOTDEF if the specified port number isn't defined

Examples

See Also

masd_set_port_cmatrix

Name

`masd_set_port_cmatrix -` Sets the port's characteristic matrix.

Synopsis

```
#include "mas/mas_dpi.h"

int32 masd_set_port_cmatrix(int32 portnum, struct
mas_characteristic_matrix* cmatrix);
```

Description

Set the characteristic matrix for the specified port.

Return value

Returns

- 0 on success
- MERR_NOTDEF if the specified port number isn't defined

Examples

See Also

masd_get_port_by_name

Name

`masd_get_port_by_name` - Gets a port number from a port name string.

Synopsis

```
#include “mas/mas_dpi.h”

int32 masd_get_port_by_name(int32 device_instance, char* name, int32*
retval_portnum);
```

Description

Uses "retval_portnum" to return the port number of a port on the given device with the given name. If device_instance is -1, then the name is assumed to be unique over all devices and the first instance of the name in the master port list is returned. The caller must allocate the memory to hold the value pointed to by retval_portnum.

Return value

Returns

- 0 on success
- MERR_NOTDEF if the specified port number isn’t defined

Examples

See Also

masd_get_state

Name

`masd_get_state` - Get the device's state pointer.

Synopsis

```
#include "mas/mas_dpi.h"

int32 masd_get_state(int32 device_instance, void** retval_state);
```

Description

Retrieves the device's state pointer. Devices can use the get/set state functions to store state information associated with the device instance number. The caller will need to cast the void pointer to something else.

Return value

Returns

- 0 on success
- MERR_NOTDEF if the specified device instance number isn't defined

Examples

See Also

masd_set_state

Name

`masd_set_state` - Set the device's state pointer.

Synopsis

```
#include "mas/mas_dpi.h"

int32 masd_set_state( int32 device_instance, void* state );
```

Description

Sets the device's state pointer. Devices can use the get/set state functions to store state information associated with the device instance number. The caller will need to cast the void pointer to something else.

Return value

Returns

- 0 on success
- MERR_NOTDEF if the specified port number isn't defined

Examples

See Also

masd_get_data_characteristic

Name

`masd_get_data_characteristic -` Get the data characteristic of a configured port.

Synopsis

```
#include “mas/mas_dpi.h”

int32 masd_get_data_characteristic(int32 portnum, struct
mas_data_characteristic** dc );
```

Description

If a port is configured, this function will retrieve the configured data characteristic.

Return value

Returns

- 0 on success
- MERR_NOTDEF if the specified port number isn’t defined
- MERR_INVALID if the port wasn’t configured

Examples

See Also

masd_init_dynamic_ports

Name

`masd_init_dynamic_ports -` Initialize the dynamic port handler.

Synopsis

```
#include “mas/mas_dpi.h”

int32 masd_init_dynamic_ports( struct masd_dynamic_port_node** head );
```

Description

The dynamic port functions operate on a linked list. This function initializes the head of that list. Devices that will use dynamic ports should call this function to initialize their local list.

Return value

Returns

- 0 on success
- MERR_MEMORY if there isn’t enough memory

Examples

See Also

`masd_request_dynamic_port`

masd_request_dynamic_port

Name

`masd_request_dynamic_port -` Request a dynamic port entry, to be used later when masd_get_dynamic_port is called.

Synopsis

```
#include “mas/mas_dpi.h”

int32 masd_request_dynamic_port( int32 device_instance, struct
masd_dynamic_port_node* head );
```

Description

The dynamic port subsystem allows devices to have ports that are created or destroyed during device run-time. This functionality augments the static port creation employed in the device profile and is absolutely fundamental to devices like mas_mix and mas_net that are capable of accepting many connections.

For each call, `masd_request_dynamic_port` generates a unique port name, adds an entry to the dynamic port queue with head *`head`*, and queues a mas_asm_make_port action on the reaction port.

NOTE *The port is not created until the device returns from its current action and the reaction queue can be processed. A subsequent call to masd_get_dynamic_port will retrieve the port number of the new port.*

Devices may choose to allocate a pool of dynamic ports in the mas_dev_init action.

Return value

Returns

- 0 on success
- MERR_MEMORY if there isn’t enough memory

Examples

See Also

`masd_request_dynamic_ports`
`masd_get_dynamic_port`
`masd_init_dynamic_ports`

masd_request_dynamic_ports

Name

`masd_request_dynamic_ports -` Request many dynamic port entries, to be used later when masd_get_dynamic_port is called.

Synopsis

```
#include “mas/mas_dpi.h”

int32 masd_request_dynamic_ports( int32 device_instance, struct
masd_dynamic_port_node* head, int16 num );
```

Description

This is a wrapper for masd_request_dynamic_port. It calls it *num* times. On error, it returns immediately and does not request the remaining ports.

Return value

Returns

- 0 on success
- MERR_MEMORY if there isn’t enough memory

Examples

See Also

`masd_request_dynamic_port`

masd_get_dynamic_port

Name

`masd_get_state` - Get the device's state pointer.

Synopsis

```
#include "mas/mas_dpi.h"

int32 masd_get_dynamic_port( int32 device_instance, struct
masd_dynamic_port_node* head, int32* retval_portnum);
```

Description

Retrieves the number of the next available dynamic port from the linked list of requested dynamic ports beginning with *head*. The port number is stored in the value pointed to by retval_portnum. The entry in the linked list is removed.

Return value

Returns

- 0 on success
- MERR_NULLPTR if the list head is NULL
- MERR_NOTDEF if there are no available ports in the list

Examples

See Also

masd_destroy_dynamic_port

Name

`masd_destroy_dynamic_port -` Destroy a requested dynamic port.

Synopsis

```
#include “mas/mas_dpi.h”

int32 masd_destroy_dynamic_port( int32 device_instance, struct
masd_dynamic_port_node* head );
```

Description

masd_destroy_dynamic_port destroys a previously requested dynamic port that hasn’t been retrieved with a call to masd_get_dynamic_port. It queues a mas_asm_destroy_port action on the reaction port and removes the entry from the end of the linked list beginning with *head*. Use this function if you’ve requested too many dynamic ports and you wish to play nice *[allocate resources efficiently]* with the rest of the devices.

Return value

Returns

- 0 on success
- MERR_NULLPTR if the list head is NULL
- MERR_NOTDEF if there are no available ports in the list

Examples

See Also

`mas_asm_destroy_port [?]`

masd_reaction_queue_action

Name

`masd_reaction_queue_action -` Construct and queue an event on the reaction port.

Synopsis

```
#include “mas/mas_dpi.h”

int32 masd_reaction_queue_action(int32 reaction_portnum,
int32 target_device_instance, const char* action_name, void* predicate,
int32 predicate_len, uint32 secs, uint32 frac, int time_is_rel,
int priority, uint32 period, int32 num_port_dependencies, int32*
port_dependencies);
```

Description

With the reaction port subsystem, devices can emit a sequence of events to be inserted into the scheduler queue. These events typically trigger actions on the same device, allowing internal device logic to decide what the device should do next - instead of inserting the logic into the scheduler’s program.

To use this feature, the assembler creates a new type of source port, called a reaction port (port type *`MAS_REACTION`*) for each device instance. Reaction ports cannot be connected to another port; instead, they are written by the device and read by the scheduler. Like all other ports, the reaction port passes data with mas_data structures. Events are passed using the data segment to hold a void pointer, pointing to the event structure. At the end of each action, the scheduler processes every *`mas_data`* structure that was posted to the reaction port, casts its data segment to a pointer to an event, and inserts the event into its queue.

`masd_reaction_queue_action` facilitates access to the reaction port from the device. It forms an event out of its arguments, packs the event into a mas_data structure, and queues the event on the reaction port. For events that do not require timing information or dependencies, use the `masd_reaction_queue_action_simple`. For events that do not require timing information, but require dependencies, use `masd_reaction_queue_action_simple_dep`. If the programmer chooses to construct her own event, use `masd_reaction_queue_event`.

Device responses to actions are also queued on the reaction port; they’re identified by the action name `mas_sch_response`.

Return value

Returns

- 0 on success

•MERR_MEMORY if there isn't enough memory
•MERR_NODEF if one of the specified port numbers isn't defined

Examples

See Also

`masd_reaction_queue_action_simple`
`masd_reaction_queue_action_simple_dep`
`masd_reaction_queue_event`

masd_reaction_queue_action_simple

Name

`masd_reaction_queue_action_simple` - Construct and queue a simple event on the reaction port.

Synopsis

```
#include “mas/mas_dpi.h”

int32 masd_reaction_queue_action_simple(int32 reaction_portnum,
int32 target_device_instance, char* action_name, void* predicate,
int32 predicate_len);
```

Description

(See `masd_reaction_queue_action` for a full description of reaction ports.)

`masd_reaction_queue_action_simple` facilitates access to the reaction port from the device for events that don’t require the full set of timing arguments. This function is a wrapper around `masd_reaction_queue_action`. It uses default zero values for the additional arguments.

Return value

Returns

- 0 on success
- MERR_MEMORY if there isn’t enough memory
- MERR_NODEF if one of the specified port numbers isn’t defined

Examples

See Also

`masd_reaction_queue_action_simple_dep`

masd_reaction_queue_action_simple_dep

Name

`masd_reaction_queue_action_simple_dep` - Construct and queue a simple event with dependencies on the reaction port.

Synopsis

```
#include “mas/mas_dpi.h”

int32 masd_reaction_queue_action_simple_dep(int32 reaction_portnum,
int32 target_device_instance, char* action_name, void* predicate,
int32 predicate_len, int32 num_port_dependencies,
int32* port_dependencies );
```

Description

(See `masd_reaction_queue_action` for a full description of reaction ports.)

`masd_reaction_queue_action_simple_`dep facilitates access to the reaction port from the device for events that don’t require the full set of timing arguments but do require dependency information. This function is a wrapper around `masd_reaction_queue_action`. It uses default zero values for the additional arguments.

Return value

Returns

- 0 on success
- MERR_MEMORY if there isn’t enough memory
- MERR_NODEF if one of the specified port numbers isn’t defined

Examples

See Also

`masd_reaction_queue_action_simple`

masd_reaction_queue_event

Name

`masd_reaction_queue_event` - Queue an event on the reaction port.

Synopsis

```
#include “mas/mas_dpi.h”

int32 masd_reaction_queue_event( int32 reaction_portnum,
struct mas_event* event );
```

Description

(See `masd_reaction_queue_action` for a full description of reaction ports.)

`masd_reaction_queue_event` queues an event on the reaction port. Unlike `masd_reaction_queue_action*`, this function requires the caller to construct and populate an event structure. As in `masd_reaction_queue_action`, a pointer to the event structure is stuffed into the data segment and queued on the reaction port.

NOTE The caller must not free the memory used for the event structure; the scheduler will do that.

Return value

Returns

- 0 on success
- MERR_MEMORY if there isn’t enough memory
- MERR_NODEF if one of the specified port numbers isn’t defined

Examples

See Also

`masd_reaction_queue_response`

masd_reaction_queue_response

Name

masd_reaction_queue_response - Queue an action response on the reaction port.

Synopsis

```
#include "mas/mas_dpi.h"

int32 masd_reaction_queue_response(int32 reaction_portnum,
void* response, int32 response_len);
```

Description

(See masd_reaction_queue_action for a full description of reaction ports.)

Action responses allow devices to return information to a port indicated by the entity that scheduled the action. This feature relies on response ports (type *MAS_RESPONSE*) that are sinks and cannot be connected to other sources. They are written by the scheduler and read by the device. Just as reaction ports allow devices to schedule actions, response ports allow devices to receive the resultant information from actions.

The *response* is packed as a predicate to the scheduler action *mas_sch_response*, forming an event that is queued on the reaction port. During reaction port processing, as described in masd_reaction_queue_action, if the scheduler finds an action named *mas_sch_response*, it examines the *response_port* member of the just-completed event. If *response_port* is nonzero, the scheduler immediately forms a data segment from the predicate of the *mas_sch_response* and posts it to the port indicated by *response_port*. If *response_port* is zero, no response is required and the *mas_sch_response* event is discarded.

Return value

Returns

- 0 on success
- MERR_MEMORY if there isn't enough memory
- MERR_NODEF if one of the specified port numbers isn't defined

Examples

See Also

masd_reaction_queue_event

masd_get_state
masd_get_state

masd_get_state

Name

`masd_get_state -` Get the device's state pointer.

Synopsis

```
#include "mas/mas_dpi.h"

int32 masd_get_state(int32 device_instance, void** retval_state);
```

Description

Retrieves the device's state pointer. Devices can use the get/set state functions to store state information associated with the device instance number. The caller will need to cast the void pointer to something else.

Return value

Returns

- 0 on success
- MERR_NOTDEF if the specified device instance number isn't defined

Examples

See Also

masd_get_pre

Name

masd_get_pre - Prepare to handle a mas_get action.

Synopsis

```
#include “mas/mas_dpi.h”

int32 masd_get_pre( void* predicate, int32* retport_r, char** key_r,
struct mas_package** arg_r );
```

Description

Along with masd_get_post, these functions set-up and tear-down much of the necessary data structures required to implement a standard mas_get action handler. See **Examples** below for typical usage.

masd_get_pre requires only the predicate to your mas_get action. On return, *retport_r contains the port to which masd_get_post will post the response, *key_r contains the key string of the request, and *arg_r contains the argument package. Memory will be allocated for all of these.

See the mas_get core API documentation for a description of the standard mas_get action.

Return value

Returns 0

Example

```
/* Use the standard wrapper. */
err = masd_get_pre( predicate, &retport, &key, &arg );
if ( err < 0 ) return err;

/* call our handler */
r_package = handle_get_nugget( key, arg, state->mch );
if ( r_package == NULL ) return mas_error(MERR_INVALID);

/* post the response where it belongs and free the data structures
 * we abused */
err = masd_get_post( state->reaction, retport, key, arg, r_package );
```

See Also

masd_get_post

Name

masd_get_pre - Tear-down after handling a mas_get action.

Synopsis

```
#include “mas/mas_dpi.h”

int32 masd_get_post( int32 reaction, int32 retport, char* key, struct
mas_package* arg, struct mas_package* r_package );
```

Description

Along with masd_get_pre, these functions set-up and tear-down much of the necessary data structures required to implement a standard mas_get action handler. See **Examples** below for typical usage.

masd_get_post frees memory used in handling a mas_get action and posts the response to the specified port. Three of the arguments were constructed by masd_get_pre: *retport*, *key*, and *arg*. Additionally, this function requires the reaction port for the device and the action handler’s response package *r_package*.

See the mas_get core API documentation for a description of the standard mas_get action.

Return value

Returns 0

Example

```
/* Use the standard wrapper. */
err = masd_get_pre( predicate, &retport, &key, &arg );
if ( err < 0 ) return err;

/* call our handler */
r_package = handle_get_nugget( key, arg, state->mch );
if ( r_package == NULL ) return mas_error(MERR_INVALID);

/* post the response where it belongs and free the data structures
 * we abused */
err = masd_get_post( state->reaction, retport, key, arg, r_package );
```

See Also

masd_set_pre

Name

`masd_set_pre -` Prepare to handle a mas_set action.

Synopsis

```
#include “mas/mas_dpi.h”

int32 masd_set_pre( void* predicate, char** key_r, struct mas_package**
arg_r );
```

Description

Along with `masd_set_post`, these functions set-up and tear-down much of the necessary data structures required to implement a standard `mas_set` action handler. See **Examples** below for typical usage.

`masd_set_pre` requires only the predicate to your mas_get action. On return, *`*key_r`* contains the key string of the request, and *`*arg_r`* contains the argument package. Memory will be allocated for these.

See the `mas_set` core API documentation for a description of the standard `mas_set` action.

Return value

Returns 0

Example

```
/* Use the standard get_nugget wrapper. */
err = masd_set_pre( predicate, &key, &arg );
if ( err < 0 ) return err;

/* call our platform-independent handler */
err = handle_set_nugget( key, arg, state->mch );
if ( err < 0 ) return err;

/* call our platform-DEpendent handler */
err = oss_handle_set_nugget( state, key, arg );
if ( err < 0 ) return err;

/* cleanup after our mess */
err = masd_set_post( key, arg );
```

See Also

masd_set_post

Name

`masd_set_pre -` Tear-down after handling a mas_set action.

Synopsis

```
#include “mas/mas_dpi.h”

int32 masd_set_post( char* key, struct mas_package* arg );
```

Description

Along with `masd_set_pre`, these functions set-up and tear-down much of the necessary data structures required to implement a standard mas_get action handler. See **Examples** below for typical usage.

`masd_set_post` frees memory used in handling a `mas_set` action. The two arguments *`key`*, and *`arg`* were constructed by a prior call to `masd_set_pre`.

See the `mas_get` core API documentation for a description of the standard `mas_get` action.

Return value

Returns 0

Example

```
/* Use the standard get_nugget wrapper. */
err = masd_set_pre( predicate, &key, &arg );
if ( err < 0 ) return err;

/* call our platform-independent handler */
err = handle_set_nugget( key, arg, state->mch );
if ( err < 0 ) return err;

/* call our platform-DEpendent handler */
err = oss_handle_set_nugget( state, key, arg );
if ( err < 0 ) return err;

/* cleanup after our mess */
err = masd_set_post( key, arg );
```

See Also