

SISCI-API Reference Manual

Generated by Doxygen 1.5.3

Wed Jun 4 15:48:28 2008

Contents

1 SISI-API File Documentation

1

1 SISI-API File Documentation

1.1 sisci_api.h File Reference

Low-level SCI software functional specification.

Functions

- SISI_API_EXPORT void [SCIInitialize](#) (unsigned int flags, sci_error_t *error)
Initializes the SISI library.
- SISI_API_EXPORT void [SCITerminate](#) (void)
Terminates the SISI library.
- SISI_API_EXPORT void [SCIOpen](#) (sci_desc_t *sd, unsigned int flags, sci_error_t *error)
Opens an SCI virtual device.
- SISI_API_EXPORT void [SCIClose](#) (sci_desc_t sd, unsigned int flags, sci_error_t *error)
Closes an open SCI virtual device.
- SISI_API_EXPORT void [SCIConnectSegment](#) (sci_desc_t sd, sci_remote_segment_t *segment, unsigned int nodeId, unsigned int segmentId, unsigned int localAdapterNo, sci_cb_remote_segment_t callback, void *callbackArg, unsigned int timeout, unsigned int flags, sci_error_t *error)
Connects an application to a memory segment made available on a remote node.
- SISI_API_EXPORT void [SCIDisconnectSegment](#) (sci_remote_segment_t segment, unsigned int flags, sci_error_t *error)
SCIDisconnectSegment disconnects from the give mapped shared memory segment.
- SISI_API_EXPORT unsigned int [SCIGetRemoteSegmentSize](#) (sci_remote_segment_t segment)
SCIGetRemoteSegmentSize returns the size in bytes of a remote segment after it has been connected with SCIConnectSegment or SCIConnectSCISpace.
- SISI_API_EXPORT sci_segment_cb_reason_t [SCIWaitForRemoteSegmentEvent](#) (sci_remote_segment_t segment, sci_error_t *status, unsigned int timeout, unsigned int flags, sci_error_t *error)

SCIWaitForRemoteSegmentEvent blocks a program until an event concerning the remote segment has arrived.

- `SISCI_API_EXPORT` volatile void * [SCIMapRemoteSegment](#) ([sci_remote_segment_t](#) segment, [sci_map_t](#) *map, unsigned int offset, unsigned int size, void *addr, unsigned int flags, [sci_error_t](#) *error)

SCIMapRemoteSegment maps an area of a remote segment connected with either *SCIConnectSegment* or *SCIConnectSCISpace* into the addressable space of the program and returns a pointer to the beginning of the mapped area.

- `SISCI_API_EXPORT` void * [SCIMapLocalSegment](#) ([sci_local_segment_t](#) segment, [sci_map_t](#) *map, unsigned int offset, unsigned int size, void *addr, unsigned int flags, [sci_error_t](#) *error)

SCIMapLocalSegment maps an area of a memory segment created with *SCICreateSegment* into the addressable space of the program and returns a pointer to the beginning of the mapped area.

- `SISCI_API_EXPORT` void [SCIUnmapSegment](#) ([sci_map_t](#) map, unsigned int flags, [sci_error_t](#) *error)

SCIUnmapSegment unmaps from the programs address space a segment that was mapped either with *SCIMapLocalSegment* or with *SCIMapRemoteSegment*.

- `SISCI_API_EXPORT` void [SCICreateSegment](#) ([sci_desc_t](#) sd, [sci_local_segment_t](#) *segment, unsigned int segmentId, unsigned int size, [sci_cb_local_segment_t](#) callback, void *callbackArg, unsigned int flags, [sci_error_t](#) *error)

SCICreateSegment allocates a memory segment and creates and initializes a descriptor for a local segment.

- `SISCI_API_EXPORT` [sci_segment_cb_reason_t](#) [SCIWaitForLocalSegmentEvent](#) ([sci_local_segment_t](#) segment, unsigned int *sourcenoId, unsigned int *localAdapterNo, unsigned int timeout, unsigned int flags, [sci_error_t](#) *error)

SCIWaitForLocalSegmentEvent blocks a program until an event concerning the local segment has arrived.

- `SISCI_API_EXPORT` void [SCIPrepareSegment](#) ([sci_local_segment_t](#) segment, unsigned int localAdapterNo, unsigned int flags, [sci_error_t](#) *error)

SCIPrepareSegment guarantees that a local segment is accessible by an SCI adapter.

- `SISCI_API_EXPORT` void [SCIRemoveSegment](#) ([sci_local_segment_t](#) segment, unsigned int flags, [sci_error_t](#) *error)

SCIRemoveSegment frees the resources used by a local segment.

- `SISCI_API_EXPORT` void [SCISetSegmentAvailable](#) ([sci_local_segment_t](#) segment, unsigned int localAdapterNo, unsigned int flags, [sci_error_t](#) *error)

SCISetSegmentAvailable makes a local segment visible to remote nodes, that can then connect to it.

- `SISCI_API_EXPORT void SCISetSegmentUnavailable (sci_local_segment_t segment, unsigned int localAdapterNo, unsigned int flags, sci_error_t *error)`
SCISetSegmentUnavailable hides an available segment to remote nodes; no new connections will be accepted on that segment.
- `SISCI_API_EXPORT void SCICreateMapSequence (sci_map_t map, sci_sequence_t *sequence, unsigned int flags, sci_error_t *error)`
SCICreateMapSequence creates and initializes a new sequence descriptor that can be used to check for errors occurring in a transfer of data from or to a mapped segment.
- `SISCI_API_EXPORT void SCIRemoveSequence (sci_sequence_t sequence, unsigned int flags, sci_error_t *error)`
SCIRemoveSequence destroys a sequence descriptor.
- `SISCI_API_EXPORT sci_sequence_status_t SCIStartSequence (sci_sequence_t sequence, unsigned int flags, sci_error_t *error)`
SCIStartSequence performs the preliminary check of the error flags on the SCI adapter before starting a sequence of read and write operations on the concerned mapped segment.
- `SISCI_API_EXPORT sci_sequence_status_t SCICheckSequence (sci_sequence_t sequence, unsigned int flags, sci_error_t *error)`
SCICheckSequence checks if any error has occurred in a data transfer controlled by a sequence since the last check.
- `SISCI_API_EXPORT void SCIStoreBarrier (sci_sequence_t sequence, unsigned int flags)`
SCIStoreBarrier synchronizes all the accesses to a mapped segment.
- `SISCI_API_EXPORT void SCIFlushReadBuffers (sci_sequence_t sequence)`
SCIFlushReadBuffers flushes the prefetch buffers associated with a sequence.
- `SISCI_API_EXPORT int SCIProbeNode (sci_desc_t sd, unsigned int localAdapterNo, unsigned int nodeId, unsigned int flags, sci_error_t *error)`
SCIProbeNode checks if a remote node is reachable.
- `SISCI_API_EXPORT unsigned int SCIGetCSRRegister (sci_desc_t sd, unsigned int localAdapterNo, unsigned int SCINodeId, unsigned int CSROffset, unsigned int flags, sci_error_t *error)`
SISCI Priveleged function SCIGetCSRRegister reads the value contained in a location of the CSR space of an SCI node.
- `SISCI_API_EXPORT void SCISetCSRRegister (sci_desc_t sd, unsigned int localAdapterNo, unsigned int SCINodeId, unsigned int CSROffset, unsigned int CSRValue, unsigned int flags, sci_error_t *error)`
SISCI Priveleged function SCISetCSRRegister writes a value to a location of the CSR space of an SCI node.

- `SISCI_API_EXPORT` unsigned int `SCIGetLocalCSR` (`sci_desc_t` sd, unsigned int localAdapterNo, unsigned int CSROffset, unsigned int flags, `sci_error_t *error`)
SISCI Priveleged function Further function description missing.
- `SISCI_API_EXPORT` void `SCISetLocalCSR` (`sci_desc_t` sd, unsigned int localAdapterNo, unsigned int CSROffset, unsigned int CSRValue, unsigned int flags, `sci_error_t *error`)
SISCI Priveleged function Further function description missing.
- `SISCI_API_EXPORT` void `SCIAttachPhysicalMemory` (`sci_ioaddr_t` ioaddress, void *address, unsigned int busNo, unsigned int size, `sci_local_segment_t` segment, unsigned int flags, `sci_error_t *error`)
SISCI Priveleged function SCIAttachPhysicalMemory enables usage of physical devices and memory regions where the Physical PCI bus address (and mapped CPU address) are already known.
- `SISCI_API_EXPORT` void `SCIQuery` (unsigned int command, void *data, unsigned int flags, `sci_error_t *error`)
SCIQuery provides some information about the underlying SCI system.
- `SISCI_API_EXPORT` void `SCIGetLocalNodeId` (unsigned int adapterNo, unsigned int *nodeId, unsigned int flags, `sci_error_t *error`)
Get local node id.
- `SISCI_API_EXPORT` void `SCIGetNodeIdByAdapterName` (char *adaptername, `dis_nodeId_list_t` *nodeId, `dis_adapter_type_t` *type, unsigned int flags, `sci_error_t *error`)
Funtion description missing.
- void `SCIGetNodeInfoByAdapterName` (char *adaptername, unsigned int *adapterNo, `dis_nodeId_list_t` *nodeIdlist, `dis_adapter_type_t` *type, unsigned int flags, `sci_error_t *error`)
Funtion description missing.
- `SISCI_API_EXPORT` void `SCICreateDMAQueue` (`sci_desc_t` sd, `sci_dma_queue_t` *dq, unsigned int localAdapterNo, unsigned int maxEntries, unsigned int flags, `sci_error_t *error`)
SCICreateDMAQueue allocates resources for a queue of DMA transfers and creates and initializes a descriptor for the new queue.
- `SISCI_API_EXPORT` void `SCIRemoveDMAQueue` (`sci_dma_queue_t` dq, unsigned int flags, `sci_error_t *error`)
SCIRemoveDMAQueue frees the resources allocated for a DMA queue and destroys the corresponding descriptor.

- SISI_API_EXPORT `sci_dma_queue_state_t` `SCIEnqueueDMATransfer` (`sci_dma_queue_t` dq, `sci_local_segment_t` localSegment, `sci_remote_segment_t` remoteSegment, unsigned int localOffset, unsigned int remoteOffset, unsigned int size, unsigned int flags, `sci_error_t` *error)

SCIEnqueueDMATransfer adds the specification of a new transfer to a DMA queue.
- SISI_API_EXPORT void `SCIPostDMAQueue` (`sci_dma_queue_t` dq, `sci_cb_dma_t` callback, void *callbackArg, unsigned int flags, `sci_error_t` *error)

SCIPostDMAQueue starts the execution of a DMA queue.
- SISI_API_EXPORT void `SCIAbortDMAQueue` (`sci_dma_queue_t` dq, unsigned int flags, `sci_error_t` *error)

SCIAbortDMAQueue aborts a DMA transfer initiated with `SCIPostDMAQueue`.
- SISI_API_EXPORT void `SCIResetDMAQueue` (`sci_dma_queue_t` dq, unsigned int flags, `sci_error_t` *error)

SCIResetDMAQueue resets a DMA queue (without removing it), so it can be reused for another chain of transfers.
- SISI_API_EXPORT `sci_dma_queue_state_t` `SCIDMAQueueState` (`sci_dma_queue_t` dq)

SCIDMAQueueState returns the state of a DMA queue (see `sci_dma_queue_state_t`).
- SISI_API_EXPORT `sci_dma_queue_state_t` `SCIWaitForDMAQueue` (`sci_dma_queue_t` dq, unsigned int timeout, unsigned int flags, `sci_error_t` *error)

SCIWaitForDMAQueue blocks a program until a DMA queue has finished (because of the completion of all the transfers or due to an error) or the timeout has expired.
- SISI_API_EXPORT void `SCIphDmaEnqueue` (`sci_dma_queue_t` dmaqueue, unsigned int size, `sci_ioaddr_t` localBusAddr, unsigned int remote_nodeid, unsigned int remote_highaddr, unsigned int remote_lowaddr, unsigned int flags, `sci_error_t` *error)

Funtion description missing.
- SISI_API_EXPORT `sci_dma_queue_state_t` `SCIphDmaStart` (`sci_dma_queue_t` dmaqueue, `sci_cb_dma_t` callback, void *callbackArg, unsigned int flags, `sci_error_t` *error)

Funtion description missing.
- SISI_API_EXPORT void `SCICreateInterrupt` (`sci_desc_t` sd, `sci_local_interrupt_t` *interrupt, unsigned int localAdapterNo, unsigned int *interruptNo, `sci_cb_interrupt_t` callback, void *callbackArg, unsigned int flags, `sci_error_t` *error)

SCICreateInterrupt creates an interrupt resource and make it available to remote nodes and initializes a descriptor for the interrupt.
- SISI_API_EXPORT void `SCIRemoveInterrupt` (`sci_local_interrupt_t` interrupt, unsigned int flags, `sci_error_t` *error)

SCIRemoveInterrupt deallocates an interrupt resource and destroys the corresponding descriptor.

- SISI_API_EXPORT void [SCIWaitForInterrupt](#) ([sci_local_interrupt_t](#) interrupt, unsigned int timeout, unsigned int flags, [sci_error_t](#) *error)

SCIWaitForInterrupt blocks a program until an interrupt is received.

- SISI_API_EXPORT void [SCIConnectInterrupt](#) ([sci_desc_t](#) sd, [sci_remote_interrupt_t](#) *interrupt, unsigned int nodeId, unsigned int localAdapterNo, unsigned int interruptNo, unsigned int timeout, unsigned int flags, [sci_error_t](#) *error)

SCIConnectInterrupt connects the caller to an interrupt resource available on a remote node (see *SCICreateInterrupt*).

- SISI_API_EXPORT void [SCIDisconnectInterrupt](#) ([sci_remote_interrupt_t](#) interrupt, unsigned int flags, [sci_error_t](#) *error)

SCIDisconnectInterrupt disconnects an application from a remote interrupt resource and deallocates the corresponding descriptor.

- SISI_API_EXPORT void [SCITriggerInterrupt](#) ([sci_remote_interrupt_t](#) interrupt, unsigned int flags, [sci_error_t](#) *error)

SCITriggerInterrupt triggers an interrupt on a remote node, after having connected to it with *SCIConnectInterrupt*.

- SISI_API_EXPORT void [SCIRegisterInterruptFlag](#) (unsigned int localAdapterNo, [sci_local_interrupt_t](#) *interrupt, [sci_local_segment_t](#) segment, unsigned int offset, [sci_cb_interrupt_t](#) callback, void *callbackArg, unsigned int flags, [sci_error_t](#) *error)

SCIRegisterInterruptFlag register an "interrupt flag" that is identified as an unique location within a local segment.

- SISI_API_EXPORT void [SCIEnableConditionalInterrupt](#) ([sci_local_interrupt_t](#) interrupt, unsigned int flags, [sci_error_t](#) *error)

SCIEnableConditionalInterrupt make sure that another HW interrupt will take place the next time the corresponding interrupt flag is triggered by a "conditional interrupt" operation.

- SISI_API_EXPORT void [SCIDisableConditionalInterrupt](#) ([sci_local_interrupt_t](#) interrupt, unsigned int flags, [sci_error_t](#) *error)

SCIDisableConditionalInterrupt prevent subsequent "conditional interrupt" trigger operations for the specified interrupt flag from causing HW interrupt and handler invocations.

- SISI_API_EXPORT void [SCIGetConditionalInterruptTrigCounter](#) ([sci_local_interrupt_t](#) interrupt, unsigned int *interruptTrigCounter, unsigned int flags, [sci_error_t](#) *error)

SCIGetConditionalInterruptTrigCounter returns a value that indicates the number of times that this flag has been triggered since the last time it was enabled or disabled.

- SISI_API_EXPORT void `SCITransferBlock` (`sci_map_t` sourceMap, unsigned int sourceOffset, `sci_map_t` destinationMap, unsigned int destinationOffset, unsigned int size, unsigned int flags, `sci_error_t` *error)
SCITransferBlock copies a block of data between two mapped segments.
- SISI_API_EXPORT void `SCITransferBlockAsync` (`sci_map_t` sourceMap, unsigned int sourceOffset, `sci_map_t` destinationMap, unsigned int destinationOffset, unsigned int size, `sci_block_transfer_t` *block, `sci_cb_block_transfer_t` callback, void *callbackArg, unsigned int flags, `sci_error_t` *error)
SCITransferBlockAsync copies a block of data between two mapped segments.
- SISI_API_EXPORT void `SCIWaitForBlockTransfer` (`sci_block_transfer_t` block, unsigned int timeout, unsigned int flags, `sci_error_t` *error)
SCIWaitForBlockTransfer suspends the program waiting for an asynchronous block transfer to complete.
- SISI_API_EXPORT void `SCIAbortBlockTransfer` (`sci_block_transfer_t` block, unsigned int flags, `sci_error_t` *error)
SCIAbortBlockTransfer aborts an ongoing asynchronous block transfer started with SCITransferBlockAsync.
- SISI_API_EXPORT void `SCIMemWrite` (void *memAddr, volatile void *remoteAddr, unsigned int size, unsigned int flags, `sci_error_t` *error)
Function description missing.
- SISI_API_EXPORT void `SCIMemWrite_mcast` (void *memAddr, volatile void *remoteAddr[], unsigned int size, unsigned int dst_num, unsigned int flags, `sci_error_t` *error)
Function description missing.
- SISI_API_EXPORT void `SCIMemWrite_dual` (volatile void *memAddr, volatile void *remoteAddr1, `sci_map_t` remoteMap1, volatile void *remoteAddr2, `sci_map_t` remoteMap2, unsigned int size, unsigned int flags, `sci_error_t` *error)
Function description missing.
- SISI_API_EXPORT void `SCIMemCpy` (`sci_sequence_t` sequence, void *memAddr, `sci_map_t` remoteMap, unsigned int remoteOffset, unsigned int size, unsigned int flags, `sci_error_t` *error)
SCIMemCpy transfers efficiently a block of data from local memory to a mapped segment using the shared memory mode.
- SISI_API_EXPORT void `SCIMemCopy` (void *memAddr, `sci_map_t` remoteMap, unsigned int remoteOffset, unsigned int size, unsigned int flags, `sci_error_t` *error)
SCIMemCopy transfers efficiently a block of data from local memory to a mapped segment using the shared memory mode.

- SISI_API_EXPORT void [SCIMemCpy_dual](#) ([sci_sequence_t](#) sequence1, [sci_sequence_t](#) sequence2, void *memAddr, [sci_map_t](#) remoteMap1, [sci_map_t](#) remoteMap2, unsigned int remoteOffset, unsigned int size, unsigned int flags, [sci_error_t](#) *error)

This function is experimental and subject to change, DO NOT USE.
- SISI_API_EXPORT void [SCIMemCpy_mcast](#) ([sci_sequence_t](#) sequence[], void *memAddr, [sci_map_t](#) remoteMap[], unsigned int remoteOffset[], unsigned int size, unsigned int dst_num, unsigned int flags, [sci_error_t](#) *error)

This function is experimental and subject to change, DO NOT USE.
- SISI_API_EXPORT void [SCIRegisterSegmentMemory](#) (void *address, unsigned int size, [sci_local_segment_t](#) segment, unsigned int flags, [sci_error_t](#) *error)

SCIRegisterSegmentMemory associates an area memory allocated by the program (e.g.
- SISI_API_EXPORT void [SCIConnectSCISpace](#) ([sci_desc_t](#) sd, unsigned int localAdapterNo, [sci_remote_segment_t](#) *segment, [sci_address_t](#) address, unsigned int size, unsigned int flags, [sci_error_t](#) *error)

SISI Privileged function SCIConnectSCISpace connects an application directly to a window in the SCI address space, defined by its base address and its size, without any restriction.
- SISI_API_EXPORT void [SCIAAttachLocalSegment](#) ([sci_desc_t](#) sd, [sci_local_segment_t](#) *segment, unsigned int segmentId, unsigned int *size, [sci_cb_local_segment_t](#) callback, void *callbackArg, unsigned int flags, [sci_error_t](#) *error)

SCIAAttachLocalSegment() permits an application to "attach" to an already existing local segment, implying that two or more application want share the same local segment.
- SISI_API_EXPORT void [SCIShareSegment](#) ([sci_local_segment_t](#) segment, unsigned int flags, [sci_error_t](#) *error)

SCIShareSegment() permits other application to "attach" to an already existing local segment, implying that two or more application want share the same local segment.
- SISI_API_EXPORT void [SCIFlush](#) ([sci_sequence_t](#) sequence, unsigned int flags)

SCIFlush flushes the CPU buffers and the PSB buffers.

1.1.1 Detailed Description

Low-level SCI software functional specification.

1.1.2 Function Documentation

1.1.2.1 `SISCI_API_EXPORT void SCIIInitialize (unsigned int flags, sci_error_t * error)`

Initializes the SISCO library.

SCIIInitialize must be called before SCIOpen().

Parameters:

flags see below

error error information

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value. No specific error codes for this function.

1.1.2.2 `SISCI_API_EXPORT void SCITerminate (void)`

Terminates the SISCO library.

SCITerminate must be called after SCIClose().

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value. No specific error codes for this function.

1.1.2.3 `SISCI_API_EXPORT void SCIOpen (sci_desc_t * sd, unsigned int flags, sci_error_t * error)`

Opens an SCI virtual device.

SCIOpen opens an SCI virtual device, that is a channel to the driver. It creates and initializes a new descriptor for an SCI virtual device, to be used in subsequent calls to API functions. The same virtual device can be used to talk to different remote nodes.

If the flag `SCI_FLAG_THREAD_SAFE` is specified, operations on resources depending on the virtual device are executed in a thread-safe manner.

Parameters:

sd handle to the new SCI virtual device descriptor

flags see below

error error information

Flags:

- `SCI_FLAG_THREAD_SAFE`

Operations on resources associated with this descriptor will be performed in a thread-safe manner.

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value. No specific errors for this function.

1.1.2.4 `SISCI_API_EXPORT void SCIClose (sci_desc_t sd, unsigned int flags, sci_error_t * error)`

Closes an open SCI virtual device.

`SCIClose` closes an open SCI virtual device, destroying its descriptor. After this call the handle to the descriptor becomes invalid and should not be used. `SCIClose` does not deallocate possible resources that are still in use, rather it fails if some of them exist.

Parameters:

sd handle to an open SCI virtual device descriptor

flags not used

error error information

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value; otherwise:
- `SCI_ERR_BUSY`
some resources depending on this virtual device are still in use

1.1.2.5 `SISCI_API_EXPORT void SCIConnectSegment (sci_desc_t sd, sci_remote_segment_t * segment, unsigned int nodeId, unsigned int segmentId, unsigned int localAdapterNo, sci_cb_remote_segment_t callback, void * callbackArg, unsigned int timeout, unsigned int flags, sci_error_t * error)`

Connects an application to a memory segment made available on a remote node.

`SCIConnectSegment()` connects an application to a memory segment made available on a remote node (see `SCISetSegmentAvailable()`) and creates and initializes a descriptor for the connected segment. A call to this function enters the state diagram for a remote segment shown in Figure 2.3. If a timeout different from `SCI_INFINITE_TIMEOUT` is passed to the function, the attempt to connect gives up after the specified number of milliseconds.

The connection operation is by default synchronous: the function returns only when the operation has completed; a failure exits the state diagram and gives back a handle that is not valid and that should not be used. If the flag `SCI_FLAG_ASYNCHRONOUS_CONNECT` is specified the connection is instead asynchronous: the function returns

immediately with a valid handle. In case of failure, the descriptor has to be explicitly destroyed calling `SCIDisconnectSegment()`.

A callback function can be specified to be invoked when an event concerning the segment happens; the intention to use the callback has to be explicitly declared with the flag `SCI_FLAG_USE_CALLBACK`. Alternatively, interesting events can be caught using the function `SCIWaitForRemoteSegmentEvent`.

Once a memory segment has been connected, it can either be mapped in the address space of the program (see `SCIMapRemoteSegment()`) or be used directly for DMA transfers (see `SCIEnqueueDMATransfer`). A successful connection also generates an `SCI_CB_CONNECT` event directed to the application that created the segment (see `SCICreateSegment` and `sci_cb_local_segment_t`).

Parameters:

- sd* handle to an open SCI virtual device descriptor
- segment* handle to the new connected segment descriptor
- nodeId* identifier of the node where the segment is allocated
- segmentId* identifier of the segment to connect
- localAdapterNo* number of the local adapter used for the connection
- callback* function called when an asynchronous event affecting the segment occurs
- callbackArg* user-defined parameter passed to the callback function
- timeout* time in milliseconds to wait for the connection to complete
- flags* see below
- error* error information

Flags:

- `SCI_FLAG_USE_CALLBACK`
The specified callback is active
- `SCI_FLAG_ASYNCHRONOUS_CONNECT`
The connection is asynchronous

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value; otherwise to one of the following.
- `SCI_ERR_NO_SUCH_SEGMENT`
The segment to connect could not be found
- `SCI_ERR_CONNECTION_REFUSED`
The connection attempt has been refused by the remote node

- `SCI_ERR_TIMEOUT`
The function timed out
- `SCI_ERR_NO_LINK_ACCESS`
It was not possible to communicate via the local adapter
- `SCI_ERR_NO_REMOTE_LINK_ACCESS`
it was not possible to communicate via a remote switch port
- `SCI_ERR_SYSTEM`
The callback thread could not be created

1.1.2.6 `SISCI_API_EXPORT void SCIDisconnectSegment (sci_remote_segment_t segment, unsigned int flags, sci_error_t * error)`

`SCIDisconnectSegment` disconnects from the give mapped shared memory segment.

`SCIDisconnectSegment()` disconnects from a remote segment connected by calling `SCIDisconnectSegment()` or `SCIDisconnectSCISpace()` and deallocates the corresponding descriptor. After this call the handle to the descriptor becomes invalid and should not be used.

If the segment was connected using `SCIDisconnectSegment()` the execution of `SCIDisconnectSegment` also generates an `SCI_CB_DISCONNECT` event directed to the application that created the segment (see `SCIDisconnectSegment()` and `sci_cb_local_segment_t`).

Parameters:

- segment* handle to the connected segment descriptor
- flags* not used
- error* error information

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value; otherwise:
- `SCI_ERR_BUSY`
The segment is currently mapped or in use

1.1.2.7 `SISCI_API_EXPORT unsigned int SCIGetRemoteSegmentSize (sci_remote_segment_t segment)`

`SCIGetRemoteSegmentSize` returns the size in bytes of a remote segment after it has been connected with `SCIDisconnectSegment` or `SCIDisconnectSCISpace`.

Parameters:

- segment* handle to the connected segment descriptor

Returns:

- The function returns the size in bytes of the remote segment.

Errors:

- No error information is provided by the function.

1.1.2.8 SISI_API_EXPORT `sci_segment_cb_reason_t` `SCIWaitForRemoteSegmentEvent` (`sci_remote_segment_t` *segment*, `sci_error_t` * *status*, `unsigned int` *timeout*, `unsigned int` *flags*, `sci_error_t` * *error*)

`SCIWaitForRemoteSegmentEvent` blocks a program until an event concerning the remote segment has arrived.

If a timeout different from `SCI_INFINITE_TIMEOUT` is specified the function gives up when the timeout expires. `SCIWaitForRemoteSegmentEvent` cannot be used if a callback associated with the remote segment is active (see `SCIConnectSegment`).

Parameters:

- segment* handle to the connected segment descriptor
- status* istatus information
- timeout* time in milliseconds to wait before giving up
- flags* not used
- error* error information

Returns:

- If successful, the function returns the reason that generated the received event.

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value; otherwise to one of the following:
 - `SCI_ERR_TIMEOUT`
The function timed out after specified timeout value.
 - `SCI_ERR_ILLEGAL_OPERATION`
Illegal operation.
 - `SCI_ERR_CANCELLED`
The segment has been disconnected. The handle is invalid when this error is returned

1.1.2.9 SISI_API_EXPORT volatile void* SCIMapRemoteSegment (sci_remote_segment_t segment, sci_map_t * map, unsigned int offset, unsigned int size, void * addr, unsigned int flags, sci_error_t * error)

SCIMapRemoteSegment maps an area of a remote segment connected with either SCIConnectSegment or SCIConnectSCISpace into the addressable space of the program and returns a pointer to the beginning of the mapped area.

The function also creates and initializes a descriptor for the mapped segment. If a virtual address is suggested, together with the flag SCI_FLAG_FIXED_MAP_ADDR, the function tries first to map the segment at that address. If the flag SCI_FLAG_READONLY_MAP is specified, the remote segment is mapped in read-only mode.

Flags:

- SCI_FLAG_SHARED_MAP
The low level physical map may be shared by other applications.
- SCI_FLAG_FIXED_MAP_ADDR
Map at the suggested virtual address
- SCI_FLAG_READONLY_MAP
The segment is mapped in read-only mode
- SCI_FLAG_LOCK_OPERATION
Enable Lock operations (fetch and add)
- SCI_FLAG_READ_PREFETCH_AGGR_HOLD_MAP
Enable aggressive prefetch with speculative hold.
- SCI_FLAG_READ_PREFETCH_HOLD_MAP
The PSB66 will prefetch 128 bytes. The PSB64 will prefetch 64 bytes. The stream will keep the prefetched data until all bytes have been read. As soon as some bytes have been read, the stream may be used by another request.
- SCI_FLAG_READ_PREFETCH_NO_HOLD_MAP
The PSB66 will prefetch 128 bytes. The PSB64 will prefetch 64 bytes. As soon as the PCI read retry has been accepted, the stream will change state to FREE, even if less than 128 bytes were actually read.
- SCI_FLAG_IO_MAP_IOSPACE
Mapping using non-prefetch space (iospace) No prefetching, or speculative hold enabled.
- SCI_FLAG_WRITES_ENABLE_GATHER_MAP
Enable write gathering in non-prefetch space (iospace)

- `SCI_FLAG_DMOVE_MAP`
Enable DMOVE packet type. The stream will be set into FREE state immediately.
- `SCI_FLAG_WRITES_DISABLE_GATHER_MAP`
Disable use of gather.
- `SCI_FLAG_DISABLE_128_BYTES_PACKETS`
Disable use of 128-Byte packets
- `SCI_FLAG_CONDITIONAL_INTERRUPT_MAP`
Write operations through this map will cause an atomic "fetch-and-add-one" operation on remote memory, but in addition an interrupt will be generated if the target memory location contained a "null value" before the add operation was carried out. The conditional interrupt flag must also be specified in the `SCIRegisterInterruptFlag()` function.
- `SCI_FLAG_UNCONDITIONAL_INTERRUPT_MAP`
Write operations through this map will cause an interrupt for the remote adapter "in addition to" updating the corresponding remote memory location with the data being written. The unconditional interrupt flag must also be specified in the `SCIRegisterInterruptFlag()` function.
- `SCI_FLAG_WRITE_BACK_CACHE_MAP`
Enable cacheing of the mapped region. Writes through this map will be written to a write back cache, hence no remote SCI updates until the cache line is flushed. The application is responsible for the cache flush operation. The `SCImemCpy()` function will handle this correctly by doing cache flushes internally. If this flag has been used for `SciMapRemoteSegment` it has to be set for `SciMemWrite`. This feature is architecture dependent and not be available on all platforms.
- `SCI_FLAG_NO_MEMORY_LOOPBACK_MAP`
Forces a map to a remote segment located in the local machine to be mapped using SCI loopback. This is useful i.e. if you want to use a regular map access to be serialized with lock operations. The default behaviour is to access a remote segment located in the local machine as a local MMU operation.
- `SCI_FLAG_READ_ORDERING_MAP`
Forces all outstanding SCI requests to be completed before a read operation is accepted. (Not tested)
- `SCI_FLAG_WRITE_ORDERING_MAP`
Forces all outstanding SCI requests to be completed before a write operation is accepted. (Not tested)

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value; otherwise to one of the following:
 - `SCI_ERR_OUT_OF_RANGE`
The sum of the offset and size is larger than the segment size.
 - `SCI_ERR_SIZE_ALIGNMENT` Size is not correctly aligned as required by the implementation.
 - `SCI_ERR_OFFSET_ALIGNMENT`
Offset is not correctly aligned as required by the implementation.

1.1.2.10 `SISCI_API_EXPORT void* SCIMapLocalSegment (sci_local_segment_t segment, sci_map_t * map, unsigned int offset, unsigned int size, void * addr, unsigned int flags, sci_error_t * error)`

`SCIMapLocalSegment` maps an area of a memory segment created with `SCICreateSegment` into the addressable space of the program and returns a pointer to the beginning of the mapped area.

The function also creates and initializes a descriptor for the mapped segment. If a virtual address is suggested, together with the flag `SCI_FLAG_FIXED_MAP_ADDR`, the function tries first to map the segment at that address. If the flag `SCI_FLAG_READONLY_MAP` is specified, the local segment is mapped in read-only mode.

Parameters:

segment handle to the descriptor of the local segment to be mapped
map handle to the new mapped segment descriptor
offset offset inside the local segment where the mapping should start
size size of the area of the local segment to be mapped, starting from offset
addr suggested virtual address where the segment should be mapped
flags see below
error error information

Flags::

- `SCI_FLAG_FIXED_MAP_ADDR`
The function should try first to map at the suggested virtual address
- `SCI_FLAG_READONLY_MAP`
The segment is mapped in read-only mode

Returns:

- If successful, the function returns a pointer to the beginning of the mapped area. In case of error it returns 0.

Errors:

- On successful completion, error points to the SCI_ERR_OK value; otherwise to one of the following:
- SCI_ERR_OUT_OF_RANGE
The sum of the offset and size is larger than the segment size.
- SCI_ERR_SIZE_ALIGNMENT
Size is not correctly aligned as required by the implementation.
- SCI_ERR_OFFSET_ALIGNMENT
Offset is not correctly aligned as required by the implementation.

1.1.2.11 SISI_API_EXPORT void SCIUnmapSegment (sci_map_t map, unsigned int flags, sci_error_t * error)

SCIUnmapSegment unmaps from the programs address space a segment that was mapped either with SCIMapLocalSegment or with SCIMapRemoteSegment.

It also destroys the corresponding descriptor, therefore after this call the handle to the descriptor becomes invalid and should not be used.

Parameters:

- map* handle to the mapped segment descriptor
flags not used
error error information

Errors:

- On successful completion, error points to the SCI_ERR_OK value; otherwise to one of the following:
- SCI_ERR_BUSY
The map is currently in use.

1.1.2.12 SISI_API_EXPORT void SCICreateSegment (sci_desc_t sd, sci_local_segment_t * segment, unsigned int segmentId, unsigned int size, sci_cb_local_segment_t callback, void * callbackArg, unsigned int flags, sci_error_t * error)

SCICreateSegment allocates a memory segment and creates and initializes a descriptor for a local segment.

A host-wide unique identifier is associated to the new segment. This function causes a local segment to enter its state diagram, shown in Figure 2.2. A callback function can be specified to be invoked when an event concerning the segment happens (see sci_segment_cb_reason_t); the intention to use the callback has to be explicitly declared

with the flag `SCI_FLAG_USE_CALLBACK`. Alternatively, interesting events can be caught using the function `SCIWaitForLocalSegmentEvent`. If the flag `SCI_FLAG_EMPTY` is specified, no memory is allocated for the segment and only the descriptor is initialized. Using the flag `SCI_FLAG_PRIVATE` declares that the segment will never be made available for external connections (see `SCISetSegmentAvailable`); in this case the specified segment identifier is meaningless, avoiding the internal check for its uniqueness. These two flags are useful to transform a user-allocated piece of memory (e.g. via `malloc`) into a mapped segment to be used in a block transfer (see `SCITransferBlock` and `SCITransferBlockAsync`): an empty and private segment is first created and then associated to the user-allocated memory (see `SCIRegisterSegmentMemory`); the segment can then be transformed in a mapped segment (see `SCIMapLocalSegment`) and possibly prepared for a DMA transfer (see `SCIPrepareSegment`).

Parameters:

- sd* handle to an open SCI virtual device descriptor
- segment* handle to the new local segment descriptor
- segmentId* segment identifier
- size* segment size; if `SCI_FLAG_EMPTY` is specified, size means the maximum size of the memory area that can be associated with this local segment
- callback* callback function called when an asynchronous event affecting the local segment occurs
- callbackArg* user-defined argument passed to the callback function
- flags* not used
- error* error information

Flags:

- `SCI_FLAG_USE_CALLBACK` The callback function will be invoked for events on this segment.
- `SCI_FLAG_EMPTY` No memory will be allocated for the segment.
- `SCI_FLAG_PRIVATE` The segment will be private meaning it will never be any connections to it.

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value; otherwise to one of the following:
 - `SCI_ERR_SEGMENTID_USED`
The segment with this `segmentId` is already used
 - `SCI_ERR_SIZE_ALIGNMENT`
Size is not correctly aligned as required by the implementation.
 - `SCI_ERR_SYSTEM`
The callback thread could not be created

1.1.2.13 SISI_API_EXPORT sci_segment_cb_reason_t SCISCIWaitForLocalSegmentEvent (*sci_local_segment_t segment*, unsigned int * *sourcenodeId*, unsigned int * *localAdapterNo*, unsigned int *timeout*, unsigned int *flags*, sci_error_t * *error*)

SCISCIWaitForLocalSegmentEvent blocks a program until an event concerning the local segment has arrived.

If a timeout different from SCI_INFINITE_TIMEOUT is specified the function gives up when the timeout expires. SCISCIWaitForLocalSegmentEvent cannot be used if a callback associated with the local segment is active (see SCISCICreateSegment).

Parameters:

- segment* handle to local segment descriptor
- sourcenodeId* identifier of the node that have generated the event
- localAdapterNo* number of the local adapter that receive the event
- timeout* time in milliseconds to wait before giving up
- flags* not used
- error* error information

Returns:

- If successful, the function returns the reason corresponding to the received event.

Errors:

- On successful completion, error points to the SCI_ERR_OK value; otherwise to one of the following:
- SCI_ERR_TIMEOUT
The function timed out after specified timeout value.
- SCI_ERR_CANCELLED
The wait operation has been cancelled due to a SCISCIRemoveSegment() on the same handle. The handle is invalid when this error is returned.

1.1.2.14 SISI_API_EXPORT void SCISCIPrepareSegment (*sci_local_segment_t segment*, unsigned int *localAdapterNo*, unsigned int *flags*, sci_error_t * *error*)

SCISCIPrepareSegment guarantees that a local segment is accessible by an SCI adapter.

Parameters:

- segment* handle to the local segment descriptor
- localAdapterNo* number of the adapter for which the segment is prepared
- flags* not used

error error information

Flags:

- `SCI_FLAG_DMA_SOURCE_ONLY`
The segment will be used as a source segment for DMA operations. On some system types this will enable the SISI driver to use performance improving features.

Errors:

- On successful completion, *error* points to the `SCI_ERR_OK` value. No specific errors for this function.

1.1.2.15 SISI_API_EXPORT void SCIRemoveSegment (sci_local_segment_t segment, unsigned int flags, sci_error_t * error)

`SCIRemoveSegment` frees the resources used by a local segment.

The physical memory is deallocated only if it was allocated when the segment was created with `SCICreateSegment`. The function also destroys the descriptor associated with the local segment; after this call the handle to the descriptor becomes invalid and should not be used. `SCIRemoveSegment` fails if other resources, either locally or remotely, depend on it (see Figure 2.1). Before calling this function, the program should consider the use of `SCISetSegmentUnavailable` with the flags `NOTIFY` or `FORCE_DISCONNECT`.

Parameters:

segment handle to local segment descriptor

flags see below

error error information

Flags:

- `SCI_FLAG_FORCE_REMOVE` Force the removal of the segment even if there still exists active connections.

Errors:

- On successful completion, *error* points to the `SCI_ERR_OK` value; otherwise:
- `SCI_ERR_BUSY`
Unable to remove the segment. The segment is currently in use.

Warning:

The '`SCI_FLAG_FORCE_REMOVE`' is `_NOT_` intended for general use. Use with caution and preferably only after consulting with Dolphin. Incorrect use may

cause uncontrolled remote access to unintended memory and may have severe impact on system security and stability. If 'SCI_FLAG_FORCE_REMOVE' is used on segments with attached physical memory, it's the responsibility of the user to assure proper management of that memory and to assure that all remote connections is closed prior to (possibly) releasing that memory.

1.1.2.16 SISI_API_EXPORT void SCISetSegmentAvailable (sci_local_segment_t segment, unsigned int localAdapterNo, unsigned int flags, sci_error_t * error)

SCISetSegmentAvailable makes a local segment visible to remote nodes, that can then connect to it.

According to the state diagram shown in Figure 2.2 a local segment can be made available only after it has been prepared (see SCIPrepareSegment).

Parameters:

segment handle to local segment descriptor

localAdapterNo number of the local adapter where the local segment is made available for connections

flags not used.

error error information

Errors:

- On successful completion, error points to the SCI_ERR_OK value; otherwise to one of the following:
- SCI_ERR_SEGMENT_NOT_PREPARED
The segment has not been prepared for access from this adapter.
- SCI_ERR_ILLEGAL_OPERATION
The segment is created with the SCI_FLAG_PRIVATE flag specified and therefore has no segmentId.

1.1.2.17 SISI_API_EXPORT void SCISetSegmentUnavailable (sci_local_segment_t segment, unsigned int localAdapterNo, unsigned int flags, sci_error_t * error)

SCISetSegmentUnavailable hides an available segment to remote nodes; no new connections will be accepted on that segment.

If the flag SCI_FLAG_NOTIFY is specified, the operation is notified to the remote nodes connected to the local segment. The notification should be interpreted as an invitation to disconnect. If the flag SCI_FLAG_FORCE_DISCONNECT is specified, the remote nodes are forced to disconnect. These two flags can be used to implement a smooth removal of a local segment (see SCIRemoveSegment).

Parameters:

- segment* handle to the local segment descriptor
- localAdapterNo* number of the local adapter where the local segment was made available
- flags* see below
- error* error information

Flags:

- `SCI_FLAG_FORCE_DISCONNECT`
The connected nodes are forced to disconnect
- `SCI_FLAG_NOTIFY`
The connected nodes receive a notification of the operation

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value; otherwise:
- `SCI_ERR_ILLEGAL_OPERATION`
The operation is illegal in the current state of the segment

1.1.2.18 `SISCI_API_EXPORT void SCICreateMapSequence (sci_map_t map, sci_sequence_t * sequence, unsigned int flags, sci_error_t * error)`

`SCICreateMapSequence` creates and initializes a new sequence descriptor that can be used to check for errors occurring in a transfer of data from or to a mapped segment.

If the flag `SCI_FLAG_FAST_BARRIER` is specified, when a store barrier operation is applied to the sequence, it is executed in the fastest possible way allowed by the SCI adapter. There could be a limited number of fast store barrier resources available, therefore `SCICreateMapSequence` fails if none are left.

Parameters:

- map* handle to a valid mapped segment descriptor
- sequence* handle to the new sequence descriptor
- flags* see below
- error* error information

Flags:

- `SCI_FLAG_FAST_BARRIER`
Use the fast store barrier

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value. No specific errors for this function.

1.1.2.19 SISI_API_EXPORT void SCIRemoveSequence (sci_sequence_t *sequence*, unsigned int *flags*, sci_error_t * *error*)

SCIRemoveSequence destroys a sequence descriptor.

After this call the handle to the descriptor becomes invalid and should not be used.

Parameters:

sequence handle to the sequence descriptor

flags not used

error error information

Errors:

- On successful completion, error points to the SCI_ERR_OK value. No specific errors for this function.

1.1.2.20 SISI_API_EXPORT sci_sequence_status_t SCIStartSequence (sci_sequence_t *sequence*, unsigned int *flags*, sci_error_t * *error*)

SCIStartSequence performs the preliminary check of the error flags on the SCI adapter before starting a sequence of read and write operations on the concerned mapped segment.

Subsequent checks are done calling SCICheckSequence, as far as no errors occur, in which case SCIStartSequence shall be called again until it returns SCI_SEQ_OK. If the return value is SCI_SEQ_PENDING there is a pending error and the program is required to call SCIStartSequence until it succeeds, before doing other transfer operations on the segment.

Parameters:

sequence handle to the sequence descriptor

flags not used

error error information

Returns:

- The function returns the status of the sequence.

Errors:

- On successful completion, error points to the SCI_ERR_OK value. No specific errors for this function.

1.1.2.21 SISI_API_EXPORT sci_sequence_status_t SCICheckSequence (sci_sequence_t *sequence*, unsigned int *flags*, sci_error_t * *error*)

SCICheckSequence checks if any error has occurred in a data transfer controlled by a sequence since the last check.

The previous check can have been done by calling either SCIStartSequence, that also initiates the sequence, or SCICheckSequence itself. SCICheckSequence can be invoked several times in a row without calling SCIStartSequence, as far as it does not fail, returning SCI_SEQ_OK (i.e. there were no transmission errors in the sequence). If the return value is SCI_SEQ_RETRIABLE the operation can be immediately retried. A return value SCI_SEQ_NOT_RETRIABLE means that there have been a fatal error, probably also notified via callbacks to the corresponding mapped segment; it is not legal to execute other read or write operations on the segment until a call to SCIStartSequence does not fail. As well, if the return value is SCI_SEQ_PENDING it is not legal to perform read or write operations on the segment until a call to SCIStartSequence does not fail. The default behaviour of SCICheckSequence is to flush the write buffers of the SCI adapter and to wait for all the outstanding write requests to be completed. To prevent this actions the caller has to use specific flags.

Parameters:

sequence handle to a sequence descriptor

flags see below

error error information

Flags:

- SCI_FLAG_NO_FLUSH
Do not flush the write buffers
- SCI_FLAG_NO_STORE_BARRIER
Do not wait for outstanding write requests

Returns:

- The function returns the status of the sequence.

Errors:

- On successful completion, error points to the SCI_ERR_OK value. No specific error values for this function.

1.1.2.22 SISI_API_EXPORT void SCIStoreBarrier (sci_sequence_t *sequence*, unsigned int *flags*)

SCIStoreBarrier synchronizes all the accesses to a mapped segment.

When the function returns the write buffers have been flushed and all outstanding SCI transactions related to the mapped segment have completed.

Parameters:

sequence handle to the mapped segment descriptor

flags not used

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value. No specific errors for this function.

1.1.2.23 SISI_API_EXPORT void SCIFlushReadBuffers (sci_sequence_t *sequence*)

`SCIFlushReadBuffers` flushes the prefetch buffers associated with a sequence.

Parameters:

sequence handle to the sequence descriptor.

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value. No information error is provided by the function.

1.1.2.24 SISI_API_EXPORT int SCIProbeNode (sci_desc_t *sd*, unsigned int *localAdapterNo*, unsigned int *nodeId*, unsigned int *flags*, sci_error_t * *error*)

`SCIProbeNode` checks if a remote node is reachable.

Parameters:

sd handle to an open SCI virtual device descriptor

localAdapterNo number of the local adapter used for the check

nodeId identifier of the remote node

flags not used

error error information

Returns:

- The function returns 1 when the remote node can be reached, otherwise returns 0.

Errors:

- Errors If the function returns 1, the error value points to `SCI_ERR_OK`; otherwise it points to one of the following:
- `SCI_ERR_NO_LINK_ACCESS`
It was not possible to communicate via the local adapter.
- `SCI_ERR_NO_REMOTE_LINK_ACCESS`
It was not possible to communicate via a remote switch port.

1.1.2.25 SISI_API_EXPORT unsigned int SCIGetCSRRegister (sci_desc_t *sd*, unsigned int *localAdapterNo*, unsigned int *SCINodeId*, unsigned int *CSROffset*, unsigned int *flags*, sci_error_t * *error*)

SISCI Privileged function SCIGetCSRRegister reads the value contained in a location of the CSR space of an SCI node.

A node is specified using its physical node identifier. The location is determined by an offset from the base address of the CSR space.

Parameters:

sd handle to an open SCI virtual device descriptor
localAdapterNo number of the local adapter used for the communication
SCINodeId physical identifier of the remote node where the CSR space resides
CSROffset offset in the CSR space
flags not used
error error information

Errors:

- On successful completion, error points to the SCI_ERR_OK value; otherwise:
- SCI_ERR_NO_LINK_ACCESS
It was not possible to communicate via the local adapter.
- SCI_ERR_NO_REMOTE_LINK_ACCESS
It was not possible to communicate via a remote switch port.

1.1.2.26 SISI_API_EXPORT void SCISetCSRRegister (sci_desc_t *sd*, unsigned int *localAdapterNo*, unsigned int *SCINodeId*, unsigned int *CSROffset*, unsigned int *CSRValue*, unsigned int *flags*, sci_error_t * *error*)

SISCI Privileged function SCISetCSRRegister writes a value to a location of the CSR space of an SCI node.

A node is specified using its physical node identifier. The location is determined by an offset from the base address of the CSR space.

Parameters:

sd handle to an open SCI virtual device descriptor
localAdapterNo number of the local adapter used for the communication
SCINodeId physical identifier of the remote node where the CSR space resides
CSROffset location offset in the CSR space
CSRValue value to write in the location
flags not used
error error information

Flags:

- `SCI_ERR_NO_LINK_ACCESS`
It was not possible to communicate via the local adapter.
- `SCI_ERR_NO_REMOTE_LINK_ACCESS`
It was not possible to communicate via a remote switch port.

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value; otherwise to one of the following:
- `SCI_ERR_NO_LINK_ACCESS`
It was not possible to communicate via the local adapter
- `SCI_ERR_NO_REMOTE_LINK_ACCESS`
It was not possible to communicate via a remote switch port

1.1.2.27 `SISCI_API_EXPORT unsigned int SCIGetLocalCSR (sci_desc_t sd, unsigned int localAdapterNo, unsigned int CSROffset, unsigned int flags, sci_error_t * error)`

SISCI Privileged function Further function description missing.

Parameters:

sd handle to an open SCI virtual device descriptor
localAdapterNo number of the local adapter used for CSR
CSROffset location offset in the CSR space
flags not used
error error information

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value. No specific errors for this function.

1.1.2.28 `SISCI_API_EXPORT void SCISetLocalCSR (sci_desc_t sd, unsigned int localAdapterNo, unsigned int CSROffset, unsigned int CSRValue, unsigned int flags, sci_error_t * error)`

SISCI Privileged function Further function description missing.

Parameters:

sd handle to an open SCI virtual device descriptor

localAdapterNo number of the local adapter used for CSR

CSROffset location offset in the CSR space

CSRValue

flags not used

error error information

Errors:

- On successful completion, error points to the SCI_ERR_OK value. No specific errors for this function.

1.1.2.29 `SISCI_API_EXPORT void SCIAttachPhysicalMemory (sci_ioaddr_t iaddress, void * address, unsigned int busNo, unsigned int size, sci_local_segment_t segment, unsigned int flags, sci_error_t * error)`

SISCI Priveleged function SCIAttachPhysicalMemory enables usage of physical devices and memory regions where the Physical PCI bus address (and mapped CPU address) are already known.

The function will register the physical memory as a SISCI segment which can be connected and mapped as a regular SISCI segment.

Requirements: SCICreateSegment() with flag SCI_FLAG_EMPTY must have been called in advance

Parameters:

iaddress This is the address on the PCI bus that a PCI bus master has to use to write to the specified memory

address This is the (mapped) virtual address that the application has to use to access the device. This means that the device has to be mapped in advance by the devices own driver. If the device is not to be accessed by the local CPU, the address pointer should be set to NULL.

busNo TBD

size TBD

segment TBD

flags TBD

error TBD

Errors:

- On successful completion, error points to the SCI_ERR_OK value. No specific errors for this function.

1.1.2.30 SISI_API_EXPORT void SCIQuery (unsigned int *command*, void * *data*, unsigned int *flags*, sci_error_t * *error*)

SCIQuery provides some information about the underlying SCI system.

The information can be vendor dependent, but some requests are specified in the API and shall be satisfied: the vendor identifier, the version of the API implemented, some adapter characteristics. Each request defines its own data structure to be used as input and output to SCIQuery. The memory management (allocation and deallocation) of the data structures has to be performed by the caller.

Parameters:

command type of information required

data generic data structure for possible sub-commands and output information

flags not used

error error information

Commands

- Three major commands are as below:
- SCI_Q_VENDORID
The vendor identifier is returned in a data structure of type sci_query_string
- SCI_Q_API
The version of the API implemented is returned in a data structure of type sci_query_string
- SCI_Q_ADAPTER
Certain adapter information, depending on the sub-command (see below), is returned in a data structure of type

Errors:

- On successful completion, error points to the SCI_ERR_OK value; otherwise:
- SCI_ERR_ILLEGAL_QUERY
Unrecognized command.

1.1.2.31 SISI_API_EXPORT void SCIGetLocalNodeId (unsigned int *adapterNo*, unsigned int * *nodeId*, unsigned int *flags*, sci_error_t * *error*)

Get local node id.

Parameters:

adapterNo number of the local adapter to get node id

nodeId identifier of the local node

flags not used

error error information

Errors:

- On successful completion, error points to the SCI_ERR_OK value, no specific errors for this function.

1.1.2.32 SISCI_API_EXPORT void SCIGetNodeIdByAdapterName (char * *adaptername*, dis_nodeId_list_t * *nodeId*, dis_adapter_type_t * *type*, unsigned int *flags*, sci_error_t * *error*)

Funtion description missing.

Parameters:

adaptername name of the adapter to query node id

nodeId

type

flags not used

error error information

Errors:

- On successful completion, error points to the SCI_ERR_OK value, no specific errors for this function.

1.1.2.33 void SCIGetNodeInfoByAdapterName (char * *adaptername*, unsigned int * *adapterNo*, dis_nodeId_list_t * *nodeIdlist*, dis_adapter_type_t * *type*, unsigned int *flags*, sci_error_t * *error*)

Funtion description missing.

Parameters:

adaptername names of the adapters to query node ids

adapterNo number of local adapter with the given adapter name

nodeIdlist

type

flags not used

error error information

Errors:

- On successful completion, error points to the SCI_ERR_OK value, no specific errors for this function.

1.1.2.34 SISI_API_EXPORT void SCICreateDMAQueue (sci_desc_t *sd*, sci_dma_queue_t * *dq*, unsigned int *localAdapterNo*, unsigned int *maxEntries*, unsigned int *flags*, sci_error_t * *error*)

SCICreateDMAQueue allocates resources for a queue of DMA transfers and creates and initializes a descriptor for the new queue.

After the creation the state of the queue is IDLE (see sci_dma_queue_state_t). All the segments involved in the transfers included in the same DMA queue must use the same adapter, which is specified as a parameter in this function. If a handle to an existing queue is passed to this function it is overwritten with the handle to a new queue. The old queue is not affected but it may not be accessible any more.

Parameters:

- sd* handle to an open SCI virtual device descriptor
- dq* handle to the new DMA queue descriptor
- localAdapterNo* number of the adapter whose DMA engine will be used for the transfers
- maxEntries* maximum number of entries allowed in the DMA queue
- flags* not used
- error* error information

Flags:

- SCI_FLAG_DMA_PHDMA
Create physical DMA queue. Please note that this is a privileged operation.

Errors:

- On successful completion, error points to the SCI_ERR_OK value, no specific errors for this function.

1.1.2.35 SISI_API_EXPORT void SCIRemoveDMAQueue (sci_dma_queue_t *dq*, unsigned int *flags*, sci_error_t * *error*)

SCIRemoveDMAQueue frees the resources allocated for a DMA queue and destroys the corresponding descriptor.

After this call the handle to the DMA queue descriptor becomes invalid and should not be used. As shown in the state diagram in Figure 2.4, this function can be called only if the queue is either in the initial (IDLE) or in a final (DONE, ERROR or ABORTED) state, otherwise the operation is illegal and the error is detected (see sci_dma_queue_state_t).

Parameters:

- dq* handle to the DMA queue descriptor
- flags* not used

error error information

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value; otherwise:
- `SCI_ERR_ILLEGAL_OPERATION`
Not allowed in this queue state.

1.1.2.36 SISI_API_EXPORT `sci_dma_queue_state_t` `SCIEnqueueDMATransfer` (`sci_dma_queue_t dq`, `sci_local_segment_t localSegment`, `sci_remote_segment_t remoteSegment`, `unsigned int localOffset`, `unsigned int remoteOffset`, `unsigned int size`, `unsigned int flags`, `sci_error_t * error`)

`SCIEnqueueDMATransfer` adds the specification of a new transfer to a DMA queue.

Either the source or the destination of the transfer must be a local segment. By default the transfer is from the local segment to the remote one; if the transfer is in the opposite direction, the flag `SCI_FLAG_DMA_READ` has to be specified. As shown in Figure 2.4, this function can be called only if the queue is either in the `IDLE` or in the `GATHER` states, otherwise the operation is illegal and the error is detected. If the function is successful the final state is `GATHER` (see `sci_dma_queue_state_t`). The local adapter used by the local and the remote segments must be the same than the one specified when the queue was created (see `SCICreateDMAQueue`).

Parameters:

dq handle to the DMA queue descriptor

localSegment handle to the local segment descriptor

remoteSegment handle to the remote segment descriptor

localOffset base address inside the local segment where data reside (or where data are transferred to, if the transfer is from the remote segment to the local one)

remoteOffset base address inside the remote segment where data are transferred to (or where data reside, if the transfer is from the remote segment to the local one)

size size of the data to be transferred

flags see below

error error information

Flags:

- `SCI_FLAG_DMA_READ`
The DMA will be remote \rightarrow local (default is local \rightarrow remote)

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value; otherwise to one of the following:
- `SCI_ERR_OUT_OF_RANGE`
The sum of the offset and size is larger than the segment size or larger than max DMA size.
- `SCI_ERR_MAX_ENTRIES`
The DMA queue is full
- `SCI_ERR_ILLEGAL_OPERATION`
Illegal operation
- `SCI_ERR_SIZE_ALIGNMENT`
Size is not correctly aligned as required by the implementation.
- `SCI_ERR_OFFSET_ALIGNMENT`
Offset is not correctly aligned as required by the implementation.
- `SCI_ERR_SEGMENT_NOT_PREPARED`
The local segment has not been prepared for access from the adapter associated with the queue.
- `SCI_ERR_SEGMENT_NOT_CONNECTED`
The remote segment is not connected through the adapter associated with the queue.

1.1.2.37 `SISCI_API_EXPORT void SCIPostDMAQueue (sci_dma_queue_t dq, sci_cb_dma_t callback, void * callbackArg, unsigned int flags, sci_error_t * error)`

`SCIPostDMAQueue` starts the execution of a DMA queue.

The function returns as soon as the transfer specifications contained in the queue are passed to the DMA engine of the SCI adapter. If a callback function is specified and explicitly activated using the flag `SCI_FLAG_USE_CALLBACK`, it is asynchronously invoked when all the transfers have completed or if an error occurs during a transfer. Alternatively, an application can block waiting for the queue completion calling `SCI-WaitForDMAQueue`. As shown in Figure 2.4, `SCIPostDMAQueue` can be called only if the queue is in the `GATHER` state, otherwise the operation is illegal and the error is detected. If the function is successful the final state is `POSTED` (see `sci_dma_queue_state_t`).

Parameters:

- dq* handle to the DMA queue descriptor
- callback* callback function to be invoked when all the DMA transfers have completed or in case an error occurs during a transfer
- callbackArg* user-defined parameter passed to the callback function

flags see below

error error information

Flags:

- **SCI_FLAG_USE_CALLBACK**
The end of the transfer will cause the callback function to be invoked.

Errors:

- On successful completion, error points to the **SCI_ERR_OK** value; otherwise to one of the following:
 - **SCI_ERR_ILLEGAL_OPERATION**
Illegal operation
 - **SCI_ERR_SYSTEM**
The callback thread could not be created

1.1.2.38 SISCI_API_EXPORT void SCIAbortDMAQueue (sci_dma_queue_t dq, unsigned int flags, sci_error_t * error)

SCIAbortDMAQueue aborts a DMA transfer initiated with SCIPostDMAQueue.

Calling this function is really meaningful only if the queue is in the POSTED state. If the function is successful the final state is ABORTED (see `sci_dma_queue_state_t`). If the state is already ABORTED or if it is DONE or ERROR, the call is equivalent to a no-op. In all the other cases the call is illegal and the error is detected. There is a potential race condition if the call happens when the state is already changing from POSTED to either DONE or ERROR because the transfer has completed or an error has occurred. To check what happened the program should call `SCIDMAQueueState`.

Parameters:

dq handle to the DMA queue descriptor

flags not used

error error information

Errors:

- On successful completion, error points to the **SCI_ERR_OK** value; otherwise:
 - **SCI_ERR_ILLEGAL_OPERATION**
Illegal operation

1.1.2.39 SISI_API_EXPORT void SCIResetDMAQueue (sci_dma_queue_t *dq*, unsigned int *flags*, sci_error_t * *error*)

SCIResetDMAQueue resets a DMA queue (without removing it), so it can be reused for another chain of transfers.

According to the state diagram in Figure 2.4, this function can be called when the queue is in any state other than the POSTED state. If the function is successful the final state is IDLE (see `sci_dma_queue_state_t`).

Parameters:

dq handle to the DMA queue descriptor

flags not used

error error information

Errors:

- On successful completion, error points to the SCI_ERR_OK value, otherwise:
- SCI_ERR_ILLEGAL_OPERATION
The state transition implied by this operation is not allowed in the current state of the queue.

1.1.2.40 SISI_API_EXPORT sci_dma_queue_state_t SCIDMAQueueState (sci_dma_queue_t *dq*)

SCIDMAQueueState returns the state of a DMA queue (see `sci_dma_queue_state_t`).

The call does not affect the state of the queue

Parameters:

dq handle to the DMA queue descriptor

Returns:

- The function returns the state of the DMA queue.

Errors:

- On successful completion, error points to the SCI_ERR_OK value. No specific errors for this function.

1.1.2.41 SISI_API_EXPORT sci_dma_queue_state_t SCIWaitForDMAQueue (sci_dma_queue_t *dq*, unsigned int *timeout*, unsigned int *flags*, sci_error_t * *error*)

SCIWaitForDMAQueue blocks a program until a DMA queue has finished (because of the completion of all the transfers or due to an error) or the timeout has expired.

If timeout is SCI_INFINITE_TIMEOUT the function blocks until a relevant event arrives. The function returns the current state of the queue. According to the state diagram shown in Figure 2.4, calling this function is really meaningful only if the queue is in the POSTED state. If the state is in the ABORTED, DONE or ERROR states, the call is equivalent to a no-op. In all the other cases the call is illegal and the error is detected (see sci_dma_queue_state_t). SCIWaitForDMAQueue cannot be used if a callback associated with the DMA queue is active.

Parameters:

dq handle to a DMA queue descriptor
timeout timeout in milliseconds to wait before giving up
flags not used
error error information

Returns:

- On successful completion, the function returns the current state of the DMA queue. In case of error the returned value is undefined.

Errors:

- On successful completion, error points to the SCI_ERR_OK value; otherwise to one of the followings:
- SCI_ERR_ILLEGAL_OPERATION
Illegal operation
- SCI_ERR_TIMEOUT
The function timed out after specified timeout value.
- SCI_ERR_CANCELLED
The wait was interrupted, due to arrival of signal.

1.1.2.42 SISCO_API_EXPORT void SCIPhDmaEnqueue (sci_dma_queue_t dmaqueue, unsigned int size, sci_ioaddr_t localBusAddr, unsigned int remote_nodeid, unsigned int remote_highaddr, unsigned int remote_lowaddr, unsigned int flags, sci_error_t * error)

Funtion description missing.

Parameters:

dmaqueue
size size size of the transfer
localBusAddr
remote_nodeid identifier of the remote node which ...(*)
remote_highaddr

remote_lowaddr

flags see below

error error informations

Flags:

- SCI_FLAG_DMA_READ

Errors:

- On successful completion, error points to the SCI_ERR_OK value. No specific errors for this function.

1.1.2.43 SISI_API_EXPORT sci_dma_queue_state_t SCIphDmaStart (sci_dma_queue_t *dmaqueue*, sci_cb_dma_t *callback*, void * *callbackArg*, unsigned int *flags*, sci_error_t * *error*)

Function description missing.

Parameters:

dmaqueue

callback function called when an asynchronous event affecting the segment occurs.

callbackArg user-defined parameter passed to the callback function.

flags see below

error error information

Flags:

- SCI_FLAG_DMA_WAIT
- SCI_FLAG_USE_CALLBACK
- SCI_FLAG_DMA_RESET

Errors:

- On successful completion, error points to the SCI_ERR_OK value, otherwise:
- SCI_ERR_SYSTEM
The callback thread could not be created

1.1.2.44 SISI_API_EXPORT void SCICreateInterrupt (sci_desc_t *sd*, sci_local_interrupt_t * *interrupt*, unsigned int *localAdapterNo*, unsigned int * *interruptNo*, sci_cb_interrupt_t *callback*, void * *callbackArg*, unsigned int *flags*, sci_error_t * *error*)

SCICreateInterrupt creates an interrupt resource and make it available to remote nodes and initializes a descriptor for the interrupt.

An interrupt is associated by the driver with a unique number. If the flag SCI_FLAG_FIXED_INTNO is specified, the function tries to use the number passed by the caller.

Parameters:

- sd* handle to an open SCI virtual device descriptor
- interrupt* handle to the new interrupt descriptor
- localAdapterNo* number of the local adapter used to make the interrupt
- interruptNo* number assigned to the interrupt
- callback* function called when the interrupt is triggered
- callbackArg* user-defined parameter passed to the callback function
- flags* see below
- error* error information

Flags:

- SCI_FLAG_USE_CALLBACK
The specified callback is active
- SCI_FLAG_FIXED_INTNO
The interrupt number is specified by the caller
- SCI_FLAG_SHARED_INT
- SCI_FLAG_COUNTING_INT
This flag will enable counting interrupts. This means that the number of triggered interrupts is equal to the number of received interrupts.

Errors:

- On successful completion, error points to the SCI_ERR_OK value; otherwise to one of the following:
- SCI_ERR_INTNO_USED
This interrupt number is already used
- SCI_ERR_SYSTEM
The callback thread could not be created

1.1.2.45 SISI_API_EXPORT void SCIRemoveInterrupt (sci_local_interrupt_t interrupt, unsigned int flags, sci_error_t * error)

SCIRemoveInterrupt deallocates an interrupt resource and destroys the corresponding descriptor.

After this call the handle to the descriptor becomes invalid and should not be used.

Parameters:

interrupt handle to the local interrupt descriptor

flags not used

error error information

Errors:

- On successful completion, error points to the SCI_ERR_OK value, otherwise:
- ECI_ERR_BUSY

The resource is used by a remote node. SCIRemoveInterrupt() should be called again to clean up the resource if the application wants to reuse the interrupt. If not, the driver will clean up when the application terminates.

1.1.2.46 SISI_API_EXPORT void SCIWaitForInterrupt (sci_local_interrupt_t interrupt, unsigned int timeout, unsigned int flags, sci_error_t * error)

SCIWaitForInterrupt blocks a program until an interrupt is received.

If a timeout different from SCI_INFINITE_TIMEOUT is specified the function gives up when the timeout expires. SCIWaitForInterrupt cannot be used if a callback associated with the interrupt is active (see SCICreateInterrupt).

Parameters:

interrupt handle to the local interrupt descriptor

timeout time in milliseconds to wait before giving up

flags not used

error error information

Errors:

- On successful completion, error points to the SCI_ERR_OK value, otherwise to one of the followings:
- SCI_ERR_TIMEOUT
The function timed out after specified timeout value.
- SCI_ERR_CANCELLED
The wait was interrupted by a call to SCIRemoveInterrupt or by the arrival of a signal. The handle is invalid when this error code is returned.

1.1.2.47 SISI_API_EXPORT void SCIConnectInterrupt (sci_desc_t *sd*, sci_remote_interrupt_t * *interrupt*, unsigned int *nodeId*, unsigned int *localAdapterNo*, unsigned int *interruptNo*, unsigned int *timeout*, unsigned int *flags*, sci_error_t * *error*)

SCIConnectInterrupt connects the caller to an interrupt resource available on a remote node (see SCICreateInterrupt).

The function creates and initializes a descriptor for the connected interrupt.

Parameters:

- sd* handle to an open SCI virtual device descriptor
- interrupt* handle to a new remote interrupt descriptor
- nodeId* identifier of the remote node where the interrupt has been created
- localAdapterNo* number of the local adapter used for the connection
- interruptNo* number assigned to the interrupt
- timeout* time in milliseconds to wait before giving up
- flags* see below
- error* error information

Flags:

- SCI_FLAG_COUNTING_INT
This flag will enable counting interrupts. This means that the number of triggered interrupts is equal to the number of received interrupts. If SCI_FLAG_COUNTING_INT is not used, the interface guarantees that there always will be an remote interrupt generated after the first and after the last trigger. If interrupts is triggered faster than the remote interrupt handler can handle, interrupts may be lost.

Errors:

- On successful completion, error points to the SCI_ERR_OK value, otherwise to one of the followings:
 - SCI_ERR_NO_SUCH_INTNO
No such interrupt number.
 - SCI_ERR_CONNECTION_REFUSED
Connection attempt refused by remote node.
 - SCI_ERR_TIMEOUT
The function timed out after specified timeout value.

1.1.2.48 SISI_API_EXPORT void SCIDisconnectInterrupt (sci_remote_interrupt_t interrupt, unsigned int flags, sci_error_t * error)

SCIDisconnectInterrupt disconnects an application from a remote interrupt resource and deallocates the corresponding descriptor.

After this call the handle to the descriptor becomes invalid and should not be used.

Parameters:

interrupt handle to the remote interrupt descriptor

flags not used

error error information

Errors:

- On successful completion, error points to the SCI_ERR_OK value. No specific errors for this function.

1.1.2.49 SISI_API_EXPORT void SCITriggerInterrupt (sci_remote_interrupt_t interrupt, unsigned int flags, sci_error_t * error)

SCITriggerInterrupt triggers an interrupt on a remote node, after having connected to it with SCIConnectInterrupt.

What happens to the remote application that made the interrupt resource available depends on what it specified at the time it called SCICreateInterrupt.

Parameters:

interrupt handle to the remote interrupt descriptor

flags not used

error error information

Errors:

- On successful completion, error points to the SCI_ERR_OK value. No specific errors for this function.

1.1.2.50 SISI_API_EXPORT void SCIRegisterInterruptFlag (unsigned int localAdapterNo, sci_local_interrupt_t * interrupt, sci_local_segment_t segment, unsigned int offset, sci_cb_interrupt_t callback, void * callbackArg, unsigned int flags, sci_error_t * error)

SCIRegisterInterruptFlag register an "interrupt flag" that is identified as a unique location within a local segment.

If successful, the resulting interrupt handle will have been associated with the specified local segment.

It is up to the (remote) client(s) to set up an "interrupt mapping" for the corresponding segment offset using either the

`SCI_FLAG_CONDITIONAL_INTERRUPT_MAP`

or the

`SCI_FLAG_UNCONDITIONAL_DATA_INTERRUPT_MAP`

option to `SCIMapRemoteSegment()`. - I.e. after having established a connection to the corresponding segment. A trigger operation can then be implemented using a store operation via the relevant "interrupt map".

Parameters:

localAdapterNo number of the local adapter used for the interrupt.

interrupt handle to the local interrupt descriptor

segment handle to the local segment descriptor

offset

callback function called when the interrupt is triggered (*)

callbackArg user-defined parameter passed to the callback function

flags see below

error error information

Flags:

- `SCI_FLAG_CONDITIONAL_INTERRUPT`
Triggering is to take place using "conditional interrupts".

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value. No specific errors for this function.

1.1.2.51 `SISCI_API_EXPORT void SCIEnableConditionalInterrupt (sci_local_interrupt_t interrupt, unsigned int flags, sci_error_t * error)`

`SCIEnableConditionalInterrupt` make sure that another HW interrupt will take place the next time the corresponding interrupt flag is triggered by a "conditional interrupt" operation.

Default semantics:

When successful, the client can rely on that the first subsequent trigger operation will cause a HW interrupt and subsequently cause the client handler function to be invoked.

If an interrupt was triggered in parallel with the enable operation, then the operation will fail (`SCI_ERR_COND_INT_RACE_PROBLEM`), and the client can not rely on another trigger operation will lead to handler invocation. Hence, any state checking normally associated with handling the corresponding interrupt should take place before attempting to enable again.

Parameters:

interrupt handle to the local interrupt descriptor
flags not used
error error information

Errors:

- On successful completion, error points to the SCI_ERR_OK value, otherwise:
- SCI_ERR_COND_INT_RACE_PROBLEM
The enable operation failed because an incoming trigger operation happened concurrently.

1.1.2.52 SISI_API_EXPORT void SCIDisableConditionalInterrupt (sci_local_interrupt_t *interrupt*, unsigned int *flags*, sci_error_t * *error*)

SCIDisableConditionalInterrupt prevent subsequent "conditional interrupt" trigger operations for the specified interrupt flag from causing HW interrupt and handler invocations.

Default semantics:

If successful, no subsequent HW interrupts will take place, but handler invocations that have already been scheduled may still take place.

Parameters:

interrupt handle to the local interrupt descriptor
flags not used
error error information

Errors:

- On successful completion, error points to the SCI_ERR_OK value. No specific errors for this function.

1.1.2.53 SISI_API_EXPORT void SCIGetConditionalInterruptTrigCounter (sci_local_interrupt_t *interrupt*, unsigned int * *interruptTrigCounter*, unsigned int *flags*, sci_error_t * *error*)

SCIGetConditionalInterruptTrigCounter returns a value that indicates the number of times that this flag has been triggered since the last time it was enabled or disabled.

Calling the SCIEnableConditionalInterrupt / SCIDisableConditionalInterrupt functions will reset the counter value.

Default semantics:

If successful, no subsequent HW interrupts will take place, but handler invocations that have already been scheduled may still take place.

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value, otherwise:
- `SCI_ERR_OVERFLOW`

The number of trig operations have exceeded the range that can be counted.

1.1.2.54 SISI_API_EXPORT `void SCITransferBlock (sci_map_t sourceMap, unsigned int sourceOffset, sci_map_t destinationMap, unsigned int destinationOffset, unsigned int size, unsigned int flags, sci_error_t * error)`

`SCITransferBlock` copies a block of data between two mapped segments.

The function returns only when the transfer has finished.

Parameters:

sourceMap handle to the mapped segment descriptor representing the transfer source

sourceOffset offset inside the source segment, where the transfer starts

destinationMap handle to the mapped segment descriptor representing the transfer destination

destinationOffset offset inside the destination segment, where the transfer starts

size size of the data to be transferred

flags not used

error error information

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value; otherwise to one of the following:
- `SCI_ERR_OUT_OF_RANGE`
The sum of the size and offset is larger than the corresponding map size.
- `SCI_ERR_SIZE_ALIGNMENT`
Size is not correctly aligned as required by the implementation.
- `SCI_ERR_OFFSET_ALIGNMENT`
Offset is not correctly aligned as required by the implementation.
- `SCI_ERR_TRANSFER_FAILED`
The data transfer failed.

1.1.2.55 SISI_API_EXPORT void SCITransferBlockAsync (sci_map_t sourceMap, unsigned int sourceOffset, sci_map_t destinationMap, unsigned int destinationOffset, unsigned int size, sci_block_transfer_t * block, sci_cb_block_transfer_t callback, void * callbackArg, unsigned int flags, sci_error_t * error)

SCITransferBlockAsync copies a block of data between two mapped segments.

The function posts the transfer and returns immediately. A block transfer descriptor is created and initialized; its lifetime is the lifetime of the transfer and it is automatically destroyed when the transfer has completed. A callback function can be specified to be invoked when the transfer completes or if an error occurs; the intention to use the callback has to be explicitly declared with the flag SCI_FLAG_USE_CALLBACK. Alternatively, the program can block and wait for the completion calling SCIWaitForBlockTransfer.

Parameters:

- sourceMap* Handle to the mapped segment descriptor representing the transfer source
- sourceOffset* offset inside the source segment, where the transfer starts
- destinationMap* handle to the mapped segment descriptor representing the transfer destination
- destinationOffset* offset inside the destination segment, where the transfer starts
- size* size of the data to be transferred
- block* TBD
- callback* function called when the transfer has completed or if there is a failure during the transfer
- callbackArg* user-defined argument passed to the callback function
- flags* see below
- error* error information

Flags:

- SCI_FLAG_BLOCK_READ
The data transfer is from the remote segment to the local memory.
- SCI_FLAG_USE_CALLBACK
The specified callback will be invoked when the transfer has completed.

Errors:

- On successful completion, error points to the SCI_ERR_OK value; otherwise to one of the following:
- SCI_ERR_OUT_OF_RANGE
The sum of the size and offset is larger than the corresponding map size.

- `SCI_ERR_SIZE_ALIGNMENT`
Size is not correctly aligned as required by the implementation.
- `SCI_ERR_OFFSET_ALIGNMENT`
Offset is not correctly aligned as required by the implementation.
- `SCI_ERR_TRANSFER_FAILED`
The data transfer failed.

1.1.2.56 `SISCI_API_EXPORT void SCIWaitForBlockTransfer (sci_block_transfer_t block, unsigned int timeout, unsigned int flags, sci_error_t * error)`

`SCIWaitForBlockTransfer` suspends the program waiting for an asynchronous block transfer to complete.

It is illegal to use this function if a callback is active on the same transfer.

Parameters:

- block* handle to the block transfer descriptor
- timeout* time in millisecond to wait before giving up
- flags* not used
- error* error information

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value; otherwise to one of the following:
- `SCI_ERR_ILLEGAL_OPERATION`
Illegal operation
- `SCI_ERR_TIMEOUT`
The function timed out after specified timeout value.

1.1.2.57 `SISCI_API_EXPORT void SCIAbortBlockTransfer (sci_block_transfer_t block, unsigned int flags, sci_error_t * error)`

`SCIAbortBlockTransfer` aborts an ongoing asynchronous block transfer started with `SCITransferBlockAsync`.

There is a potential race condition if the call happens when the transfer is completing. If the transfer has already completed a call to `SCIAbortBlockTransfer` fails because the block transfer descriptor has been already automatically destroyed and its handle has become invalid.

Parameters:

- block* handle to the block descriptor

flags not used

error error information

Errors:

- On successful completion, error points to the SCI_ERR_OK value; otherwise to one of the following:
- SCI_ERR_ILLEGAL_OPERATION

The call to the function is invoked when the transfer has already completedn

1.1.2.58 SISI_API_EXPORT void SCIMemWrite (void * *memAddr*, volatile void * *remoteAddr*, unsigned int *size*, unsigned int *flags*, sci_error_t * *error*)

Function description missing.

Parameters:

memAddr base address in virtual memory of the source

remoteAddr offset inside the mapped segment where the transfer should start

size size of the transfer

flags see below

error error informaiton

Flags:

- SCI_FLAG_WRITE_BACK_CACHE_MAP
Only implemented for PowerPC (see doc. for SciMapRemoteSegment)

Errors:

- On successful completion, error points to the SCI_ERR_OK value; otherwise to one of the following:
- SCI_ERR_SIZE_ALIGNMENT
Size is not correctly aligned as required by the implementation.
- SCI_ERR_OFFSET_ALIGNMENT
Offset is not correctly aligned as required

1.1.2.59 SISI_API_EXPORT void SCIMemWrite_mcast (void * *memAddr*, volatile void * *remoteAddr*[], unsigned int *size*, unsigned int *dst_num*, unsigned int *flags*, sci_error_t * *error*)

Function description missing.

Parameters:

memAddr base address in virtual memory of the source
remoteAddr offset inside the mapped segment where the transfer should start
size size of the transfer
dst_num
flags see below
error error information

Flags:

- SCI_FLAG_WRITE_BACK_CACHE_MAP
Only implemented for PowerPC (see doc for SciMapRemoteSegment)

Errors:

- On successful completion, error points to the SCI_ERR_OK value; otherwise to one of the following:
- SCI_ERR_SIZE_ALIGNMENT
Size is not correctly aligned as required by the implementation.
- SCI_ERR_OFFSET_ALIGNMENT
Offset is not correctly aligned as required

1.1.2.60 SISI_API_EXPORT void SCIMemWrite_dual (volatile void * *memAddr*, volatile void * *remoteAddr1*, sci_map_t *remoteMap1*, volatile void * *remoteAddr2*, sci_map_t *remoteMap2*, unsigned int *size*, unsigned int *flags*, sci_error_t * *error*)

Function description missing.

Parameters:

memAddr base address in virtual memory of the source
remoteAddr1
remoteMap1
remoteAddr2
remoteMap2
size size of the transfer
flags not used
error error information

Errors:

- On successful completion, error points to the SCI_ERR_OK value.

1.1.2.61 SISI_API_EXPORT void SCIMemCpy (sci_sequence_t *sequence*, void * *memAddr*, sci_map_t *remoteMap*, unsigned int *remoteOffset*, unsigned int *size*, unsigned int *flags*, sci_error_t * *error*)

SCIMemCpy transfers efficiently a block of data from local memory to a mapped segment using the shared memory mode.

If the flag SCI_FLAG_ERROR_CHECK is specified the function also checks if errors have occurred during the data transfer. If the flag SCI_FLAG_BLOCK_READ is specified, the transfer is from the mapped segment to the local memory.

Parameters:

sequence TBD

memAddr base address in virtual memory of the source

remoteMap handle to the descriptor of the mapped segment that is the destination of the transfer

remoteOffset offset inside the mapped segment where the transfer should start

size size of the transfer

flags see below

error error information

Flags:

- SCI_FLAG_BLOCK_READ
The data transfer is from the remote segment to the local memory
- SCI_FLAG_ERROR_CHECK
Perform error checking

Errors:

- On successful completion, error points to the SCI_ERR_OK value; otherwise to one of the following:
- SCI_ERR_OUT_OF_RANGE
The sum of the size and offset is larger than the corresponding map size.
- SCI_ERR_SIZE_ALIGNMENT
Size is not correctly aligned as required by the implementation.
- SCI_ERR_OFFSET_ALIGNMENT
Offset is not correctly aligned as required by the implementation.
- SCI_ERR_TRANSFER_FAILED
The data transfer failed.

1.1.2.62 SISI_API_EXPORT void SCIMemCopy (void * *memAddr*, sci_map_t *remoteMap*, unsigned int *remoteOffset*, unsigned int *size*, unsigned int *flags*, sci_error_t * *error*)

SCIMemCopy transfers efficiently a block of data from local memory to a mapped segment using the shared memory mode.

SCIMemCopy is normally not recommended for performance reasons. Please use SCIMemCpy() where possible.

If the flag SCI_FLAG_ERROR_CHECK is specified the function also checks if errors have occurred during the data transfer. If the flag SCI_FLAG_BLOCK_READ is specified, the transfer is from the mapped segment to the local memory.

Parameters:

- memAddr* base address in virtual memory of the source
- remoteMap* handle to the descriptor of the mapped segment that is the destination of the transfer
- remoteOffset* offset inside the mapped segment where the transfer should start
- size* size of the transfer
- flags* see below
- error* error information

Flags:

- SCI_FLAG_BLOCK_READ
The data transfer is from the remote segment to the local memory
- SCI_FLAG_ERROR_CHECK
Perform error checking

Errors:

- On successful completion, error points to the SCI_ERR_OK value; otherwise to one of the following:
- SCI_ERR_OUT_OF_RANGE
The sum of the size and offset is larger than the corresponding map size.
- SCI_ERR_SIZE_ALIGNMENT
Size is not correctly aligned as required by the implementation.
- SCI_ERR_OFFSET_ALIGNMENT
Offset is not correctly aligned as required by the implementation.
- SCI_ERR_TRANSFER_FAILED
The data transfer failed.

1.1.2.63 SISI_API_EXPORT void SCIMemCpy_dual (*sci_sequence_t sequence1*, *sci_sequence_t sequence2*, *void * memAddr*, *sci_map_t remoteMap1*, *sci_map_t remoteMap2*, *unsigned int remoteOffset*, *unsigned int size*, *unsigned int flags*, *sci_error_t * error*)

This function is experimental and subject to change, DO NOT USE.

Parameters:

sequence1

sequence2

memAddr base address in virtual memory of the source

remoteMap1

remoteMap2

remoteOffset offset inside the mapped segment where the transfer

size size of the transfer

flags see below

error error information

Errors:

- On successful completion, error points to the SCI_ERR_OK value.

1.1.2.64 SISI_API_EXPORT void SCIMemCpy_mcast (*sci_sequence_t sequence[]*, *void * memAddr*, *sci_map_t remoteMap[]*, *unsigned int remoteOffset[]*, *unsigned int size*, *unsigned int dst_num*, *unsigned int flags*, *sci_error_t * error*)

This function is experimental and subject to change, DO NOT USE.

Parameters:

sequence handle to the sequence descriptor

memAddr base address in virtual memory of the source

remoteMap handle to the descriptor of the mapped segment that is the destination of the transfer

remoteOffset offset inside the mapped segment where the transfer

size size of the transfer

dst_num

flags see below

error error information

Flags:

- SCI_FLAG_BLOCK_READ

- `SCI_FLAG_ERROR_CHECK`

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value; otherwise to one of the following:
 - `SCI_ERR_OUT_OF_RANGE`
The sum of the size and offset is larger than the corresponding map size.
 - `SCI_ERR_SIZE_ALIGNMENT`
Size is not correctly aligned as required by the implementation.
 - `SCI_ERR_OFFSET_ALIGNMENT`
Offset is not correctly aligned as required by the implementation.
 - `SCI_ERR_TRANSFER_FAILED`
The data transfer failed.

1.1.2.65 `SISCI_API_EXPORT void SCIRegisterSegmentMemory (void * address, unsigned int size, sci_local_segment_t segment, unsigned int flags, sci_error_t * error)`

`SCIRegisterSegmentMemory` associates an area memory allocated by the program (e.g. using `malloc`) with a local segment created passing the flag `SCI_FLAG_EMPTY` to `SCICreateSegment`. The memory area is identified by its base address in virtual address space and its size. It is illegal to use the same local segment to register different memory areas. The function can try to determine if the specified address is legal or not, but this highly depends on the underlying platform.

Parameters:

- address* base address of the user-allocated memory in the programs virtual address space
- size* size of the user-allocated memory to be associated with the local segment
- segment* handle to local segment descriptor
- flags* not used
- error* error information

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value; otherwise to one of the following:
 - `SCI_ERR_SIZE_ALIGNMENT`
Size is not correctly aligned as required by the implementation.

- `SCI_ERR_ILLEGAL_ADDRESS`
Illegal address.
- `SCI_ERR_OUT_OF_RANGE`
Size is larger than the maximum size for the local segment.

1.1.2.66 SISCI_API_EXPORT `void SCISConnectSCISpace (sci_desc_t sd, unsigned int localAdapterNo, sci_remote_segment_t * segment, sci_address_t address, unsigned int size, unsigned int flags, sci_error_t * error)`

SISCI Privileged function `SCISConnectSCISpace` connects an application directly to a window in the SCI address space, defined by its base address and its size, without any restriction.

The function creates and initializes a descriptor for the connected segment. The whole responsibility of handling the segment is left to the programmer: no state diagram is defined, no callbacks can be specified, no callbacks are invoked on the node where the memory actually resides. The function has only the synchronous version; if the connection fails the returned handle is not valid and should not be used; in this case the related descriptor need not be destroyed with `SCIDisconnectSegment`. Once the SCI window has been connected, it can either be mapped in the address space of the program (see `SCIMapRemoteSegment`) or be used directly for DMA transfers (see `SCIEnqueueDMATransfer`).

Parameters:

- sd* handle to an open SCI virtual device descriptor
- localAdapterNo* number of the local adapter used for the connection
- segment* handle to the new connected segment descriptor
- address* base address of the SCI window
- size* size of the SCI window
- flags* not used
- error* error information

Errors:

- On successful completion, error points to the `SCI_ERR_OK` value; otherwise to one of the following:
- `SCI_ERR_SIZE_ALIGNMENT`
Size is not correctly aligned as required by the implementation.
- `SCI_ERR_CONNECTION_REFUSED`
Connection attempt refused by remote node.

1.1.2.67 SISI_API_EXPORT void SCIAttachLocalSegment (sci_desc_t sd, sci_local_segment_t * segment, unsigned int segmentId, unsigned int * size, sci_cb_local_segment_t callback, void * callbackArg, unsigned int flags, sci_error_t * error)

SCIAttachLocalSegment() permits an application to "attach" to an already existing local segment, implying that two or more application want share the same local segment.

The prerequisite, is that the application which originally created the segment ("owner") has preformed a SCIShareSegment() in order to mark the segment "shareable". To detach from an attached segment use the SCIRemoveSegment() call.

Flags:

- SCI_FLAG_USE_CALLBACK

The callback function will be invoked for events on this segment.

Errors:

- On successful completion, error points to the SCI_ERR_OK value; otherwise to one of the following:
 - SCI_ERR_ACCESS
No such shared segment
 - SCI_ERR_NO_SUCH_SEGMENT
No such segment
 - SCI_ERR_SYSTEM
The callback thread could not be created.

Note:

- There are no difference in "ownership" of the shared segment between the original creator and the attached applications. If the original creator performs a remove segment with other applications attached to the segment, this becomes equal to a "detach". On global level, the segment wont be removed until all attached processes as well as the original creator has performed SCIRemoveSegment.
- Current impenentation will return SCI_ERR_ACCESS for both cases. This will change from next release. Application should handle both cases.

1.1.2.68 SISI_API_EXPORT void SCIShareSegment (sci_local_segment_t segment, unsigned int flags, sci_error_t * error)

SCIShareSegment() permits other application to "attach" to an already existing local segment, implying that two or more application want share the same local segment.

The prerequisite, is that the application which originally created the segment ("owner") has preformed a SCIShareSegment() in order to mark the segment "shareable".

Parameters:

- segment* handle to the descriptor of local segment.
- flags* not used
- error* error information

Errors:

- On successful completion, error points to the SCI_ERR_OK value. No specific error information provided.

1.1.2.69 SISI_API_EXPORT void SCIFlush (sci_sequence_t *sequence*, unsigned int *flags*)

SCIFlush flushes the CPU buffers and the PSB buffers.

Parameters:

- sequence* handle to the sequence descriptor
- flags* see below

Flags:

- SCI_FLAG_FLUSH_CPU_BUFFERS_ONLY
Only flush CPU buffers (Write combining etc buffers).

Errors:

- No error information provided by this function.

1.2 sisci_types.h File Reference

Data types.

Typedefs

- typedef struct sci_desc * [sci_desc_t](#)
A variable of type sci_desc_t represents an SCI virtual device, that is a communication channel with the driver.
- typedef struct sci_local_segment * [sci_local_segment_t](#)
A variable of type sci_local_segment_t represents a local memory segment and it is initialized when the segment is allocated by calling the function SCICreateSegment.
- typedef struct sci_remote_segment * [sci_remote_segment_t](#)
A variable of type sci_remote_segment_t represents a segment residing on a remote node.

- typedef struct sci_map * [sci_map_t](#)
A variable of type `sci_map_t` represents a memory segment mapped in the process address space.
- typedef struct sci_sequence * [sci_sequence_t](#)
A variable of type `sci_sequence_t` represents a sequence of operations involving communication with remote nodes.
- typedef struct sci_dma_queue * [sci_dma_queue_t](#)
A variable of type `sci_dma_queue_t` represents a chain of specifications of data transfers to be performed using the DMA engine available on the SCI adapter.
- typedef struct sci_remote_interrupt * [sci_remote_interrupt_t](#)
A variable of type `sci_remote_interrupt_t` represents an interrupt that can be triggered on a remote node.
- typedef struct sci_local_interrupt * [sci_local_interrupt_t](#)
A variable of type `sci_local_interrupt_t` represents an instance of an interrupt that an application has made available to remote nodes.
- typedef struct sci_block_transfer * [sci_block_transfer_t](#)
A variable of type `sci_block_transfer_t` represents an asynchronous transfer of a block of data.
- typedef [sci_callback_action_t](#)(* [sci_cb_local_segment_t](#))(void *arg, [sci_local_segment_t](#) segment, [sci_segment_cb_reason_t](#) reason, unsigned int nodeId, unsigned int localAdapterNo, [sci_error_t](#) error)
Local segment callback.
- typedef [sci_callback_action_t](#)(* [sci_cb_remote_segment_t](#))(void *arg, [sci_remote_segment_t](#) segment, [sci_segment_cb_reason_t](#) reason, [sci_error_t](#) status)
Remote segment callback.
- typedef [sci_callback_action_t](#)(* [sci_cb_dma_t](#))(void IN *arg, [sci_dma_queue_t](#) queue, [sci_error_t](#) status)
DMA queue callback.
- typedef int(* [sci_cb_block_transfer_t](#))(void *arg, [sci_block_transfer_t](#) block, [sci_error_t](#) status)
Block transfer callback.
- typedef [sci_callback_action_t](#)(* [sci_cb_interrupt_t](#))(void *arg, [sci_local_interrupt_t](#) interrupt, [sci_error_t](#) status)
Interrupt callback.

Enumerations

- enum `sci_segment_cb_reason_t` {
 `SCI_CB_CONNECT` = 1, `SCI_CB_DISCONNECT`, `SCI_CB_NOT-`
 `OPERATIONAL`, `SCI_CB_OPERATIONAL`,
 `SCI_CB_LOST` }
 Reasons for segment callbacks.
- enum `sci_dma_queue_state_t`
 DMA queue status.
- enum `sci_sequence_status_t` { `SCI_SEQ_OK`, `SCI_SEQ_RETRIABLE`, `SCI-`
 `SEQ_NOT_RETRIABLE`, `SCI_SEQ_PENDING` }
 Sequence status.
- enum `sci_callback_action_t` { `SCI_CALLBACK_CANCEL` = 1, `SCI-`
 `CALLBACK_CONTINUE` }
 Callback return values.

1.2.1 Detailed Description

Data types.

1.2.2 Typedef Documentation

1.2.2.1 typedef struct `sci_desc*` `sci_desc_t`

A variable of type `sci_desc_t` represents an SCI virtual device, that is a communication channel with the driver.

Many virtual devices can be opened by the same application. It is initialized by calling the function `SCIOpen`.

1.2.2.2 typedef struct `sci_local_segment*` `sci_local_segment_t`

A variable of type `sci_local_segment_t` represents a local memory segment and it is initialized when the segment is allocated by calling the function `SCICreateSegment`.

1.2.2.3 typedef struct `sci_remote_segment*` `sci_remote_segment_t`

A variable of type `sci_local_segment_t` represents a segment residing on a remote node.

It is initialized by calling either the function `SCIConnectSegment` or the function `SCIConnectSCISpace`.

1.2.2.4 typedef struct sci_map* sci_map_t

A variable of type `sci_map_t` represents a memory segment mapped in the process address space.

It is initialized by calling either the function `SCIMapRemoteSegment` or the function `SCIMapLocalSegment`.

1.2.2.5 typedef struct sci_sequence* sci_sequence_t

A variable of type `sci_sequence_t` represents a sequence of operations involving communication with remote nodes.

It is used to check if errors have occurred during a data transfer. The descriptor is initialized when the sequence is created by calling the function `SCICreateMapSequence`.

1.2.2.6 typedef struct sci_dma_queue* sci_dma_queue_t

A variable of type `sci_dma_queue_t` represents a chain of specifications of data transfers to be performed using the DMA engine available on the SCI adapter.

The descriptor is initialized when the chain is created by calling the function `SCICreateDMAQueue`.

1.2.2.7 typedef struct sci_remote_interrupt* sci_remote_interrupt_t

A variable of type `sci_remote_interrupt_t` represents an interrupt that can be triggered on a remote node.

It is initialized by calling the function `SCIConnectInterrupt`.

1.2.2.8 typedef struct sci_local_interrupt* sci_local_interrupt_t

A variable of type `sci_local_interrupt_t` represents an instance of an interrupt that an application has made available to remote nodes.

It is initialized when the interrupt is created by calling the function `SCICreateInterrupt`.

1.2.2.9 typedef struct sci_block_transfer* sci_block_transfer_t

A variable of type `sci_block_transfer_t` represents an asynchronous transfer of a block of data.

It is initialized when the function `SCITransferBlockAsync` is invoked.

1.2.2.10 typedef sci_callback_action_t(* sci_cb_local_segment_t)(void *arg, sci_local_segment_t segment, sci_segment_cb_reason_t reason, unsigned int nodeId, unsigned int localAdapterNo, sci_error_t error)

Local segment callback.

A function of type `sci_cb_local_segment_t` can be specified when a segment is created with `SCICreateSegment()` and will be invoked asynchronously when a remote node connects to or disconnects from the segment using respectively `SCIConnectSegment()` and `SCIDisconnectSegment()`. The same callback function is also invoked whenever a problem affects the connection.

Parameters:

- arg* user-defined argument passed to the callback function.
- segment* handle to the local segment descriptor affected by the asynchronous event.
- reason* reason why the callback function has been invoked.
- nodeId* identifier of the remote node that has provoked, directly or indirectly, the asynchronous event.
- localAdapterNo* number of the local adapter that received the asynchronous event.

Returns:

`SCI_CALLBACK_CANCEL` or `SCI_CALLBACK_CONTINUE`.

1.2.2.11 `typedef sci_callback_action_t(* sci_cb_remote_segment_t)(void *arg, sci_remote_segment_t segment, sci_segment_cb_reason_t reason, sci_error_t status)`

Remote segment callback.

A function of type `sci_cb_remote_segment_t` can be specified when the connection to a memory segment is requested calling `SCIConnectSegment` and will be invoked asynchronously when the connection completes (if `SCIConnectSegment` is asynchronous), when the local node asks for disconnecting (calling `SCISetSegmentUnavailable` with the `SCI_FLAG_NOTIFY` flag) or when a problem affects the connection.

Parameters:

- arg* user-defined argument passed to the callback function.
- segment* handle to the remote segment descriptor.
- reason* reason why the callback function has been invoked.
- status* status of the remote segment.

Returns:

`SCI_CALLBACK_CANCEL` or `SCI_CALLBACK_CONTINUE`.

1.2.2.12 `typedef sci_callback_action_t(* sci_cb_dma_t)(void IN *arg, sci_dma_queue_t queue, sci_error_t status)`

DMA queue callback.

A function of type `sci_cb_dma_t` can be specified when a DMA queue is posted using `SCIPostDMAQueue` and will be invoked when the transfer has completed, either successfully or with an error.

Parameters:

arg user-defined argument passed to the callback function.

queue handle to the DMA queue descriptor.

status status information.

Returns:

SCI_CALLBACK_CANCEL or SCI_CALLBACK_CONTINUE.

1.2.2.13 typedef int(* sci_cb_block_transfer_t)(void *arg, sci_block_transfer_t block, sci_error_t status)

Block transfer callback.

A function of type `sci_cb_block_transfer_t` can be specified when a remote memory block is transferred asynchronously with `SCITransferBlockAsync` and will be invoked when the operation completes or when an error occurs.

Parameters:

arg user-defined argument passed to the callback function.

queue handle to the block transfer descriptor.

status status information

Returns:**1.2.2.14 typedef sci_callback_action_t(* sci_cb_interrupt_t)(void *arg, sci_local_interrupt_t interrupt, sci_error_t status)**

Interrupt callback.

A function of type `sci_cb_interrupt_t` can be specified when an interrupt is created with `SCICreateInterrupt` and it will be invoked asynchronously when the interrupt will be triggered from a remote node.

Parameters:

arg user-defined argument passed to the callback function.

queue handle to the triggered interrupt descriptor.

status status information

Returns:

SCI_CALLBACK_CANCEL or SCI_CALLBACK_CONTINUE.

1.2.3 Enumeration Type Documentation

1.2.3.1 enum sci_segment_cb_reason_t

Reasons for segment callbacks.

It can either be used in local segment callback function or remote segment callback function to indicate the reasons for the segment callback.

Enumerator:

SCI_CB_CONNECT Used in local segment callback function, it indicates that a remote segment has connected to the local segment.

Used in remote segment callback function, it is currently not implemented.

SCI_CB_DISCONNECT Used in local segment callback function, it indicates that a previously connected segment has disconnected with the local segment.

Used in remote segment callback function, it indicates that the local segment has disconnected with a previously connected remote segment.

SCI_CB_NOT_OPERATIONAL Precise description needed.

SCI_CB_OPERATIONAL Precise description needed.

SCI_CB_LOST Precise description needed.

1.2.3.2 enum sci_dma_queue_state_t

DMA queue status.

Precise description needed.

1.2.3.3 enum sci_sequence_status_t

Sequence status.

The type `sci_sequence_status_t` enumerates the values returned by `SCIStartSequence()` and `SCICheckSequence()`. `SCIStartSequence()` can only return `SCI_SEQ_OK` or `SCI_SEQ_PENDING`.

Enumerator:

SCI_SEQ_OK no errors: the sequence of reads and writes can continue.

SCI_SEQ_RETRIABLE non-fatal error: the failed operation can be immediately retried.

SCI_SEQ_NOT_RETRIABLE fatal error (probably notified also via a callback to the segment): need to wait until the situation is normal again.

SCI_SEQ_PENDING an error is pending, `SCIStartSequence()` should be called until it returns `SCI_SEQ_OK`.

1.2.3.4 `enum sci_callback_action_t`

Callback return values.

Enumerator:

SCI_CALLBACK_CANCEL A `SCI_CALLBACK_CANCEL` return value represents that after the callback function has been executed it will be cancelled.

SCI_CALLBACK_CONTINUE A `SCI_CALLBACK_CONTINUE` return value represents that after the callback function has been executed it will continue exist, when the expected condition is satisfied next time, the callback function will be invoked again.

Index

sci_block_transfer_t
 sisci_types.h, 58

sci_callback_action_t
 sisci_types.h, 61

SCI_CALLBACK_CANCEL
 sisci_types.h, 62

SCI_CALLBACK_CONTINUE
 sisci_types.h, 62

sci_cb_block_transfer_t
 sisci_types.h, 60

SCI_CB_CONNECT
 sisci_types.h, 61

SCI_CB_DISCONNECT
 sisci_types.h, 61

sci_cb_dma_t
 sisci_types.h, 59

sci_cb_interrupt_t
 sisci_types.h, 60

sci_cb_local_segment_t
 sisci_types.h, 58

SCI_CB_LOST
 sisci_types.h, 61

SCI_CB_NOT_OPERATIONAL
 sisci_types.h, 61

SCI_CB_OPERATIONAL
 sisci_types.h, 61

sci_cb_remote_segment_t
 sisci_types.h, 59

sci_desc_t
 sisci_types.h, 57

sci_dma_queue_state_t
 sisci_types.h, 61

sci_dma_queue_t
 sisci_types.h, 58

sci_local_interrupt_t
 sisci_types.h, 58

sci_local_segment_t
 sisci_types.h, 57

sci_map_t
 sisci_types.h, 57

sci_remote_interrupt_t
 sisci_types.h, 58

sci_remote_segment_t
 sisci_types.h, 57

sci_segment_cb_reason_t
 sisci_types.h, 61

SCI_SEQ_NOT_RETRIABLE
 sisci_types.h, 61

SCI_SEQ_OK
 sisci_types.h, 61

SCI_SEQ_PENDING
 sisci_types.h, 61

SCI_SEQ_RETRIABLE
 sisci_types.h, 61

sci_sequence_status_t
 sisci_types.h, 61

sci_sequence_t
 sisci_types.h, 58

SCIAbortBlockTransfer
 sisci_api.h, 46

SCIAbortDMAQueue
 sisci_api.h, 34

SCIAttachLocalSegment
 sisci_api.h, 53

SCIAttachPhysicalMemory
 sisci_api.h, 28

SCICheckSequence
 sisci_api.h, 23

SCIClose
 sisci_api.h, 10

SCIConnectInterrupt
 sisci_api.h, 39

SCIConnectSCISpace
 sisci_api.h, 53

SCIConnectSegment
 sisci_api.h, 10

SCICreateDMAQueue
 sisci_api.h, 30

SCICreateInterrupt
 sisci_api.h, 37

SCICreateMapSequence
 sisci_api.h, 22

SCICreateSegment
 sisci_api.h, 17

SCIDisableConditionalInterrupt
 sisci_api.h, 43

SCIDisconnectInterrupt
 sisci_api.h, 40

SCIDisconnectSegment
 sisci_api.h, 12

SCIDMAQueueState
 sisci_api.h, 35

- SCIEnableConditionalInterrupt
 - [sisci_api.h, 42](#)
- SCIEnqueueDMATransfer
 - [sisci_api.h, 32](#)
- SCIFlush
 - [sisci_api.h, 55](#)
- SCIFlushReadBuffers
 - [sisci_api.h, 25](#)
- SCIGetConditionalInterruptTrigCounter
 - [sisci_api.h, 43](#)
- SCIGetCSRRegister
 - [sisci_api.h, 25](#)
- SCIGetLocalCSR
 - [sisci_api.h, 27](#)
- SCIGetLocalNodeId
 - [sisci_api.h, 29](#)
- SCIGetNodeIdByAdapterName
 - [sisci_api.h, 30](#)
- SCIGetNodeInfoByAdapterName
 - [sisci_api.h, 30](#)
- SCIGetRemoteSegmentSize
 - [sisci_api.h, 12](#)
- SCIInitialize
 - [sisci_api.h, 9](#)
- SCIMapLocalSegment
 - [sisci_api.h, 16](#)
- SCIMapRemoteSegment
 - [sisci_api.h, 13](#)
- SCIMemCopy
 - [sisci_api.h, 49](#)
- SCIMemCpy
 - [sisci_api.h, 48](#)
- SCIMemCpy_dual
 - [sisci_api.h, 50](#)
- SCIMemCpy_mcast
 - [sisci_api.h, 51](#)
- SCIMemWrite
 - [sisci_api.h, 47](#)
- SCIMemWrite_dual
 - [sisci_api.h, 48](#)
- SCIMemWrite_mcast
 - [sisci_api.h, 47](#)
- SCIOpen
 - [sisci_api.h, 9](#)
- SCIphDmaEnqueue
 - [sisci_api.h, 36](#)
- SCIphDmaStart
 - [sisci_api.h, 37](#)
- SCIPostDMAQueue
 - [sisci_api.h, 33](#)
- SCIPrepareSegment
 - [sisci_api.h, 19](#)
- SCIProbeNode
 - [sisci_api.h, 25](#)
- SCIQuery
 - [sisci_api.h, 28](#)
- SCIRegisterInterruptFlag
 - [sisci_api.h, 41](#)
- SCIRegisterSegmentMemory
 - [sisci_api.h, 52](#)
- SCIRemoveDMAQueue
 - [sisci_api.h, 31](#)
- SCIRemoveInterrupt
 - [sisci_api.h, 38](#)
- SCIRemoveSegment
 - [sisci_api.h, 20](#)
- SCIRemoveSequence
 - [sisci_api.h, 22](#)
- SCIResetDMAQueue
 - [sisci_api.h, 34](#)
- SCISetCSRRegister
 - [sisci_api.h, 26](#)
- SCISetLocalCSR
 - [sisci_api.h, 27](#)
- SCISetSegmentAvailable
 - [sisci_api.h, 21](#)
- SCISetSegmentUnavailable
 - [sisci_api.h, 21](#)
- SCIShareSegment
 - [sisci_api.h, 54](#)
- SCIStartSequence
 - [sisci_api.h, 23](#)
- SCIStoreBarrier
 - [sisci_api.h, 24](#)
- SCITerminate
 - [sisci_api.h, 9](#)
- SCITransferBlock
 - [sisci_api.h, 44](#)
- SCITransferBlockAsync
 - [sisci_api.h, 44](#)
- SCITriggerInterrupt
 - [sisci_api.h, 41](#)
- SCIUnmapSegment
 - [sisci_api.h, 17](#)
- SCIWaitForBlockTransfer
 - [sisci_api.h, 46](#)
- SCIWaitForDMAQueue
 - [sisci_api.h, 35](#)
- SCIWaitForInterrupt
 - [sisci_api.h, 39](#)

- SCIWaitForLocalSegmentEvent
 - sisci_api.h, 18
- SCIWaitForRemoteSegmentEvent
 - sisci_api.h, 13
- sisci_api.h, 1
 - SCIAbortBlockTransfer, 46
 - SCIAbortDMAQueue, 34
 - SCIAttachLocalSegment, 53
 - SCIAttachPhysicalMemory, 28
 - SCICheckSequence, 23
 - SCIClose, 10
 - SCIConnectInterrupt, 39
 - SCIConnectSCISpace, 53
 - SCIConnectSegment, 10
 - SCICreateDMAQueue, 30
 - SCICreateInterrupt, 37
 - SCICreateMapSequence, 22
 - SCICreateSegment, 17
 - SCIDisableConditionalInterrupt, 43
 - SCIDisconnectInterrupt, 40
 - SCIDisconnectSegment, 12
 - SCIDMAQueueState, 35
 - SCIEnableConditionalInterrupt, 42
 - SCIEnqueueDMATransfer, 32
 - SCIFlush, 55
 - SCIFlushReadBuffers, 25
 - SCIGetConditionalInterruptTrig-
Counter, 43
 - SCIGetCSRRegister, 25
 - SCIGetLocalCSR, 27
 - SCIGetLocalNodeId, 29
 - SCIGetLocalIdByAdapterName, 30
 - SCIGetNodeInfoByAdapterName,
30
 - SCIGetRemoteSegmentSize, 12
 - SCIInitialize, 9
 - SCIMapLocalSegment, 16
 - SCIMapRemoteSegment, 13
 - SCIMemCopy, 49
 - SCIMemCpy, 48
 - SCIMemCpy_dual, 50
 - SCIMemCpy_mcast, 51
 - SCIMemWrite, 47
 - SCIMemWrite_dual, 48
 - SCIMemWrite_mcast, 47
 - SCIOpen, 9
 - SCIphDmaEnqueue, 36
 - SCIphDmaStart, 37
 - SCIPostDMAQueue, 33
 - SCIPrepareSegment, 19
 - SCIProbeNode, 25
 - SCIQuery, 28
 - SCIRegisterInterruptFlag, 41
 - SCIRegisterSegmentMemory, 52
 - SCIRemoveDMAQueue, 31
 - SCIRemoveInterrupt, 38
 - SCIRemoveSegment, 20
 - SCIRemoveSequence, 22
 - SCIResetDMAQueue, 34
 - SCISetCSRRegister, 26
 - SCISetLocalCSR, 27
 - SCISetSegmentAvailable, 21
 - SCISetSegmentUnavailable, 21
 - SCIShareSegment, 54
 - SCIStartSequence, 23
 - SCIStoreBarrier, 24
 - SCITerminate, 9
 - SCITransferBlock, 44
 - SCITransferBlockAsync, 44
 - SCITriggerInterrupt, 41
 - SCIUnmapSegment, 17
 - SCIWaitForBlockTransfer, 46
 - SCIWaitForDMAQueue, 35
 - SCIWaitForInterrupt, 39
 - SCIWaitForLocalSegmentEvent, 18
 - SCIWaitForRemoteSegmentEvent,
13
- sisci_types.h
 - SCI_CALLBACK_CANCEL, 62
 - SCI_CALLBACK_CONTINUE, 62
 - SCI_CB_CONNECT, 61
 - SCI_CB_DISCONNECT, 61
 - SCI_CB_LOST, 61
 - SCI_CB_NOT_OPERATIONAL, 61
 - SCI_CB_OPERATIONAL, 61
 - SCI_SEQ_NOT_RETRIABLE, 61
 - SCI_SEQ_OK, 61
 - SCI_SEQ_PENDING, 61
 - SCI_SEQ_RETRIABLE, 61
- sisci_types.h, 55
 - sci_block_transfer_t, 58
 - sci_callback_action_t, 61
 - sci_cb_block_transfer_t, 60
 - sci_cb_dma_t, 59
 - sci_cb_interrupt_t, 60
 - sci_cb_local_segment_t, 58
 - sci_cb_remote_segment_t, 59
 - sci_desc_t, 57
 - sci_dma_queue_state_t, 61
 - sci_dma_queue_t, 58

sci_local_interrupt_t, 58
sci_local_segment_t, 57
sci_map_t, 57
sci_remote_interrupt_t, 58
sci_remote_segment_t, 57
sci_segment_cb_reason_t, 61
sci_sequence_status_t, 61
sci_sequence_t, 58