



SISCI API

Functional specification

Compiled January 18, 2018

Contents

1	SISCI-API specification	1
1.1	Preface	1
1.2	Introduction	1
1.2.1	Basic Concepts	1
1.3	Cluster Architecture	3
1.4	SISCI API Data types	3
1.4.1	Data formats	3
1.4.2	Descriptors	3
1.4.3	Flags	4
1.4.4	Errors	4
1.4.5	Other data types	5
1.5	General functions	6
1.6	Remote Shared Memory	6
1.7	Direct Memory Access (DMA)	8
1.8	Interrupts	9
1.9	Device to Device transfers	9
1.10	Reflective Memory / Multicast	10
1.11	Support Functions	10
1.12	SmartIO	10
1.13	Special Functions	11
2	File Documentation	11
2.1	sisci_api.h File Reference	11
2.1.1	Detailed Description	15
2.1.2	Function Documentation	16
2.2	sisci_api_introduction.txt File Reference	56
2.3	sisci_error.h File Reference	56
2.3.1	Detailed Description	57
2.3.2	Enumeration Type Documentation	57
2.4	sisci_types.h File Reference	58
2.4.1	Detailed Description	59
2.4.2	Macro Definition Documentation	59
2.4.3	Typedef Documentation	59
2.4.4	Enumeration Type Documentation	62
	Index	65

1 SISI-API specification

1.1 Preface

Clusters of commodity processors, memories and IO-devices interconnected by a fast remote memory access network is an attractive way to build economically large multiprocessor systems, clusters and embedded type control systems.

The SISI API Project (Software Infrastructure for Shared-Memory Cluster Interconnects, "SISI") has set itself as a goal to define a common Application Programming Interface ("API") to serve as a basis for porting major applications to heterogeneous multi vendor shared memory platforms.

The SISI software and underlying drivers simplifies the process of building remote shared memory based applications. The built in resource management enables multiple concurrent SISI programs to coexist and operate independent of each other.

PCI Express NTB networks created using PCI Express adapter cable cards or PCI Express enabled backplanes provides a non coherent distributed shared memory architecture which is ideal for a very efficient implementation of the SISI API.

The functional specification of the API presented in this document is defined in ANSI C, but can also be utilized using C++. Using a lightweight wrapper, it can also be used by applications written in C#, Java and Python.

A detailed introduction to the SISI API can be found at:

www.dolphinics.com/products/embedded-sisci-developers-kit.html

1.2 Introduction

This introductory chapter defines some terms that are used throughout the document and briefly describes the cluster architecture that represents the main objective of this functional specification.

Dolphins implementation of the SISI API comes with an extensive set of example and demo programs that can be used as a basis for a new program or to fully understand the detailed aspect of SISI programming. Newcomers to the SISI API is recommended to study these programs carefully before making important architecture decisions. The demo and example programs are included with the software distribution.

The SISI API is available in user space. Dolphin is offering the same functionality in kernel space, defined by the GENIF Kernel interface. The definition of the GENIF interface can be found in the source `DIS/src/IRM_G↔X/drv/src/genif.h` The SISI driver itself, found in `DIS/src/SISI/src` is a good example how to use the GENIF Kernel interface.

The SISI API is available and fully supported with Dolphins PCI Express NTB enabled hardware products. The software is also licensed to several OEM's providing their own PCI Express enabled products.

Dolphin is offering extensive support and assistance to migrate your application to the SISI API. Please contact your sales representative for more information or email sisci-support@dolphinics.com.

1.2.1 Basic Concepts

Processor

- A collection of one or more CPUs sharing memory using a local bus.

Host

- A processor with one or more PCI Express enabled switches or pluggable adapters.

Adapter number

- A unique number identifying one or more local adapters or PCI switches.

Fabric

- The PCI Express network interconnecting the hosts. A Hosts may be connected by several parallel Fabrics.

NodeId

- A fabric unique number identifying a host on the network.

SegmentId

- A Host unique number identifying a memory segment within a host. A segment can be uniquely identified by its SegmentId and local NodeId.

PIO

- Programmed IO. A load or store operation performed by a CPU towards a mapped address.

DMA

- A system or network resource that can be used as an alternative to PIO do move data between segments. The SISCI API provides a set of functions to control the PCIe or system DMA engines.

Global DMA

- A SISCI DMA resource that can address any remote segment without being mapped into the local address space.

System DMA

- A SISCI DMA resource provided by the CPU system, e.g. Intel Crystal Beach DMA.

Local segment

- A memory segment located on the same processor where the application runs and accessed using the host memory interface. Identified by its SegmentId.

Remote segment

- A memory segment accessed via the fabric.

Connected segment

- A remote segment for which the IO base address and the size are known.

Mapped segment

- A local or remote (connected) segment mapped in the addressable space of a program.

Reflective Memory

- Hardware based broadcast / multicast functionality that simultaneously forwards data to all hosts in the fabric. Only with all configurations.

IO Device.

- A PCI or PCI Express card / device, e.g. a GPU, NVMe or FPGA attached to a host system or expansion box attached to the network.

Peer to Peer communication

- IO Devices communicating directly - device to device - without the use of the memory system. The devices can be local to a host or separated by the interconnect fabric.

1.3 Cluster Architecture

The basic elements of a cluster are a collection of hosts interconnected. A host may be a single processor or an SMP containing several CPUs. Adapters connect a host to a fabric. It is possible for a host to be connected to several fabrics. This allows for the construction of complex topologies (e.g. a two dimensional mesh). It may also be used to add redundancy and/or improve the bandwidth. Usually such architectures are obtained by using several adapters on one host.

Adapters often contain sub units that implement specific SISI API functions such as transparent remote memory access, DMA, message mailboxes or interrupts. Most sub units have CSR registers that may be accessed locally via the host adapter interface or remotely over the PCI Express fabric.

1.4 SISI API Data types

This Application Programming Interface covers different aspects of the shared memory technology and how it can be accessed by a user. The API items can then be grouped in different categories. The specification of functions and data types, presented in Chapter 3, is then preceded by a short introduction of these categories. For an easier consultation of the document, for each category a list of concerned API items is provided.

1.4.1 Data formats

Parameters and return values of the API functions, other than the data types introduced in the following sections, are expressed in the machine native data types. The only assumption is that int's are at least 32 bits and shorts are at least 16 bits. If, in the future, a specific size or endianness is needed, the Shared-Data Formats shall be used.

1.4.2 Descriptors

Working with remote shared memories, DMA transfers and remote interrupts, the major communication features that this API offers, requires the use of logical entities like devices, memory segments, DMA queues. Each of these entities is characterized by a set of properties that should be managed as a unique object in order to avoid inconsistencies. To hide the details of the internal representation and management of such properties to an API user, a number of descriptors have been defined and made opaque: their contents can be referenced with a handle and can be modified only through the functions provided by the API.

The descriptors and their meaning are the following:

sci_desc

It represents an SISI 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`.

sci_local_segment

It represents a local memory segment and it is initialized when the segment is allocated by calling the function `SCICreateSegment()`.

sci_remote_segment

It represents a segment residing on a remote node. It is initialized by calling either the function `SCICreateRemoteSegment()`.

sci_map

It represents a memory segment mapped in the process address space. It is initialized by calling either the `SCICreateMapRemoteSegment()` or `SCICreateMapLocalSegment()` function.

sci_sequence

It 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()`.

sci_dma_queue

It represents a chain of specifications of data transfers to be performed using the DMA engine available on the adapter. The descriptor is initialized when the chain is created by calling the function `SCICreateDMAQueue()`.

sci_local_interrupt

It 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\(\)](#).

sci_remote_interrupt

It represents an interrupt that can be triggered on a remote node. It is initialized by calling the function [SCIConnectInterrupt\(\)](#).

sci_local_data_interrupt

It represents an instance of a data interrupt that an application has made available to remote nodes. It is initialized when the interrupt is created by calling the function [SCICreateDataInterrupt\(\)](#).

sci_remote_data_interrupt

It represents a data interrupt that can be triggered on a remote node. It is initialized by calling the function [SCIConnectDataInterrupt\(\)](#).

Each of the above descriptors is an opaque data type and can be referenced only via a handle. The name of the handle type is given by the name of the descriptor type with a trailing `_t`.

No automatic cleanup of the resources represented by the above descriptors is performed, rather it should be provided by the API client*. Resources cannot be released (and the corresponding descriptors deallocated) until all the dependent resources have been previously released. The dependencies between resource classes can be derived by the function specifications.

1.4.3 Flags

Nearly all functions included in this API accept a flags parameter in input. It is used to obtain from a function invocation an effect that slightly differs from its default semantics (e.g. choosing between a blocking and a non-blocking version of an operation).

Each SISI API function specification is followed by a list of accepted flags. Only the flags that change the default behavior are defined. Several flags can be OR'ed together to specify a combined effect. The flags parameter, represented with an unsigned int, has then to be considered a bit mask.

Some of the functions do not currently accept any flag. The parameter is nonetheless left in the specification, because it could become useful in view of future extensions, and the implementation shall check it to be 0.

A flag value starts with the prefix `SCI_FLAG_`.

1.4.4 Errors

Most of the API functions return an error code as an output parameter to indicate if the execution succeeded or failed. The error codes are collected in an enumeration type called `sci_error_t`. Each value starts with the prefix `SCI_ERR_`. The code denoting success is `SCI_ERR_OK` and an application should check that each function call returns this value.

In Chapter 3 each function specification is followed by a list of possible errors that are typical for that function. There are however common or very generic errors that are not repeated every time, unless they do not have a particular meaning for that function:

SCI_ERR_NOT_IMPLEMENTED

- the function is not implemented

SCI_ERR_ILLEGAL_FLAG

- the flags value passed to the function contains an illegal component. The check is done even if the function does not accept any flag (i.e. it accepts only the default value 0)

SCI_ERR_FLAG_NOT_IMPLEMENTED

- the flags value passed to the function is legal but the operation corresponding to one of its components is not implemented

SCI_ERR_ILLEGAL_PARAMETER

- one of the parameters passed to the function is illegal

SCI_ERR_NOSPC

- the function is unable to allocate some needed operating system resources

SCI_ERR_API_NOSPC

- the function is unable to allocate some needed API resources

SCI_ERR_HW_NOSPC

- the function is unable to allocate some hardware resources

SCI_ERR_SYSTEM

- the function has encountered a system error; errno should be checked

Each function requiring a local adapter number can generate the following errors:

SCI_ERR_ILLEGAL_ADAPTERNO

- the adapter number is out of the legal range

SCI_ERR_NO_SUCH_ADAPTERNO

- the adapter number is legal but it does not exist.

Each function requiring a node identifier can generate the following errors:

SCI_ERR_NO_SUCH_NODEID

- a node with the specified identifier does not exist

SCI_ERR_ILLEGAL_NODEID

- the node identifier is illegal

1.4.5 Other data types

Besides the data types specified in the previous sections others are used:

- sci_address_t
- sci_segment_cb_reason_t
- sci_dma_queue_state_t
- sci_sequence_status_t

1.5 General functions

In order to use correctly the network, an application is required to execute some operations like opening or closing a communication channel with the SISI driver. For using effectively the network an application may also need some information about the local or a remote node.

- [SCIOpen\(\)](#)
- [SCIClose\(\)](#)
- [SCIQuery\(\)](#)
- [SCIProbeNode\(\)](#)

1.6 Remote Shared Memory

The PCI Express technology implements a remote memory access approach that can be used to transfer data between systems and IO devices. An application can map into its own address space a memory segment actually residing on another node; then read and write operations from or to this memory segment are automatically and transparently converted by the hardware in remote operations. This API provides full support for creating and exporting local memory segments, for connecting to and mapping remote memory segments, for checking whether errors have occurred during a data transfer.

The functions included in this category actually concern three different aspects:

Memory management:

- [SCICreateSegment\(\)](#)
- [SCIRegisterSegmentMemory\(\)](#)
- [SCIRemoveSegment\(\)](#)
- [SCIPrepareSegment\(\)](#)

Connection management:

- [SCISetSegmentAvailable\(\)](#)
- [SCISetSegmentUnavailable\(\)](#)
- [SCIConnectSegment\(\)](#)
- [SCIDisconnectSegment\(\)](#)

Segment events

- [SCIWaitForLocalSegmentEvent\(\)](#)
- [SCIWaitForRemoteSegmentEvent\(\)](#)

Shared memory operations:

- [SCIMapLocalSegment\(\)](#)

- `SCIMapRemoteSegment()`
- `SCIUnmapSegment()`
- `SCIMemCpy()`
- `SCICreateMapSequence()`
- `SCIRemoveSequence()`
- `SCIStartSequence()`
- `SCICheckSequence()`
- `SCIStoreBarrier()`
- `SCIFlush()`

Memory and connection management functions affect the state of a local segment, whose state diagram is shown in the figure below.

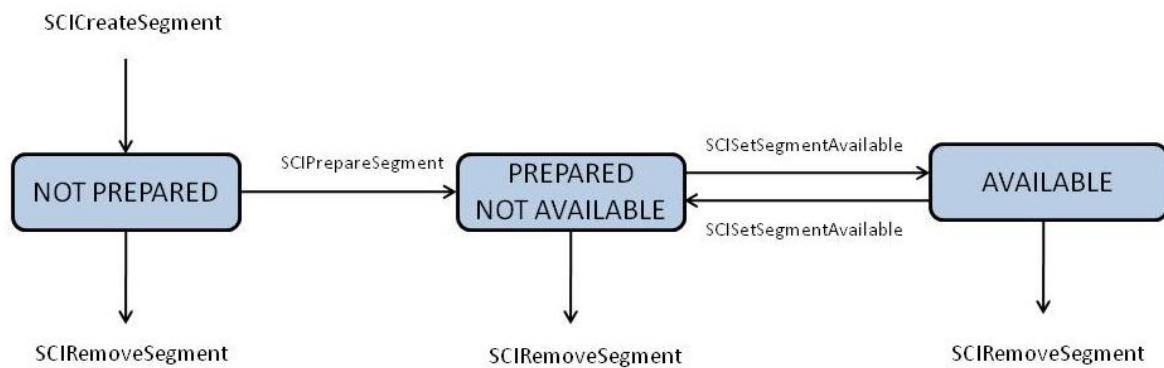


Figure 1: State diagram for a local segment

The state of a remote segment, shown in figure below, depends on what happens on the network or on the node where the segment physically resides. The transitions `sci_segment_cb_reason_t` are marked with callback reasons

between the remote segment states.

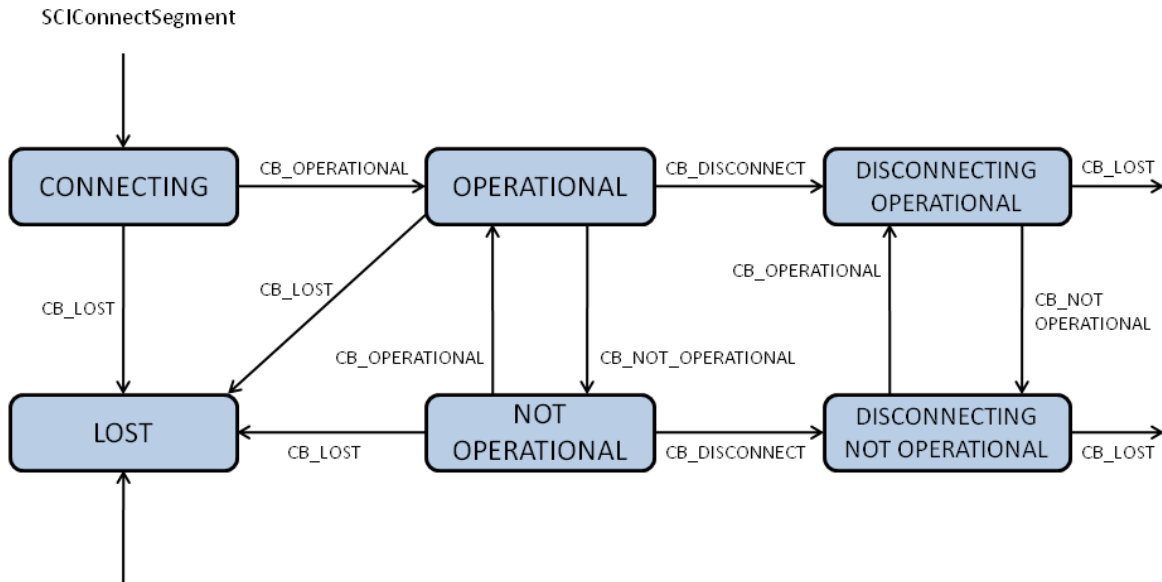


Figure 2: State diagram for a remote segment

The transitions are marked with callback reasons. SCIDisconnectSegment can be called from each state to exit the state diagram.

1.7 Direct Memory Access (DMA)

PIO (Programmed IO) has the lowest overhead and lowest latency accessing remote memory. The drawback of the PIO data transfers is that the CPU is busy reading or writing data from or to remote memory. An alternative is to use a DMA engine if this is available. The application specifies a queue of data transfers and passes it to the DMA engine. Then the CPU is free either to wait for the completion of the transfer or to do something else. In the latter case it is possible to specify a callback function that is invoked when the transfer has finished. DMA has high start up cost compared to using PIO and is normally only recommended for larger transfers.

DMA is an optional feature implemented with some PCIe chipsets only. Some systems also have a "System DMA" engine. The SISI API DMA functionality are available on all platforms supporting PCIe DMA or System DMA that is integrated and supported with the SISI driver stack. SISI DMA transfer functions will fail if there is no supported DMA engine available.

- [SCICreateDMAQueue\(\)](#)
- [SCIStartDmaTransfer\(\)](#)
- [SCIStartDmaTransferVec\(\)](#)
- [SCIStartDmaTransferMem\(\)](#)

- [SCIRemoveDMAQueue\(\)](#)
- [SCIWaitForDMAQueue\(\)](#)
- [SCIAbortDMAQueue\(\)](#)
- [SCIDMAQueueState\(\)](#)

1.8 Interrupts

Triggering an interrupt on a remote node should be considered a fast way to notify an application running remotely that something has happened. An interrupt is identified by a unique number and this is practically the only information an application gets when it is interrupted, either synchronously or asynchronously. The SISI API contains two types of interrupt, with and without data.

Interrupts with no data:

- [SCICreateInterrupt\(\)](#)
- [SCIRemoveInterrupt\(\)](#)
- [SCIConnectInterrupt\(\)](#)
- [SCIDisconnectInterrupt\(\)](#)
- [SCITriggerInterrupt\(\)](#)
- [SCIWaitForInterrupt\(\)](#)

Interrupts with data:

- [SCICreateDataInterrupt\(\)](#)
- [SCIRemoveDataInterrupt\(\)](#)
- [SCIConnectDataInterrupt\(\)](#)
- [SCIDisconnectDataInterrupt\(\)](#)
- [SCITriggerDataInterrupt\(\)](#)
- [SCIWaitForDataInterrupt\(\)](#)

1.9 Device to Device transfers

The SISI API supports setting up general IO devices to communicate directly, device to device - peer to peer communication. The alternative model, using the main memory as the intermediate buffer has significant overhead. The devices communicating can be placed in the same host or in different hosts interconnected by the shared memory fabric.

Peer to peer functionality is an optional PCI Express feature, please ensure your computer supports peer to peer transfers (Ask your system vendor).

- [SCIAttachPhysicalMemory\(\)](#)

- [SCIRegisterPCleRequester\(\)](#)
- [SCIUnregisterPCleRequester\(\)](#)

Please consult the `rpcia.c` example program found in the software distribution for more information on how to set up Device to Device transfers or access remote physical memory devices.

1.10 Reflective Memory / Multicast

The SISI API supports setting up Reflective Memory / Multicast transfers.

The Dolphin PCI Express IX and PX product families supports multicast operations as defined by the PCI Express Base Specification 2.1. Dolphin has integrated support for this functionality into the SISI API specification to make it easily available to application programmers. The multicast functionality is also available with some OEM hardware configurations. (Please check with you hardware vendor if multicast is available for your configuration). SISI functions will typically return an error if the multicast functionality is not available.

There are no special functions to to set up the Reflective Memory, programmers just need to use the flag `SCI_FL_↔AG_BROADCAST` when [SCICreateSegment\(\)](#) and [SCIConnectSegment\(\)](#) is called.

For reflective memory operations, the `Nodeld` parameter to [SCIConnectSegment\(\)](#) should be set to `DIS_BROAD_↔CAST_NODEID_GROUP_ALL`

Please consult the `reflective.c` example program found in the software distribution for more information on how to use the reflective memory functionality.

1.11 Support Functions

The SISI API provide support / helper functions that may be useful.

- [SCIGetErrorString\(\)](#)

1.12 SmartIO

NOTE: The SmartIO SISI API extension is currently not finalized and may change in future releases without warning!

The SmartIO functionality was introduced with eXpressWare 5.5.0 and is currently only available with Linux, but may be ported to other operating systems in the future. Please contact Dolpin if you interested in the SmartIO functionality.

The SISI API SmartIO extension exposes features from the SmartIO module, if present. These functions are appropriate to use in order to gain low level access to devices. They can be used to write user space drivers using SISI. The functions will work on devices anywhere in the cluster and multiple nodes can access the same device at the same time. In general, the SmartIO API will reuse 'normal' SISI features where possible. For instance, device BAR registers are exposed as segments. The main SmartIO module must be configured in order for this API to function.

SmartIO devices are represented in SISI as `sci_device_t` handles. This handle is initialized by borrowing the device, and released when the device is returned.

- [SCIBorrowDevice\(\)](#)
- [SCIReturnDevice\(\)](#)

Once a device has been borrowed, it's BARs can be accessed through a remote segment. To connect to these segments, the following function must be used. After connecting the segment is to be treated as any other remote segment.

- [SCISConnectDeviceMemory\(\)](#)

The device may also perform DMA to both local and remote segments. This must be set up by functions that map the given segment and provide the caller with the address to be used by the device to reach the given segment.

- [SCIMapLocalSegmentForDevice\(\)](#)
- [SCIMapRemoteSegmentForDevice\(\)](#)
- [SCIUnmapLocalSegmentForDevice\(\)](#)
- [SCIUnmapRemoteSegmentForDevice\(\)](#)

1.13 Special Functions

Some special functions exist, these are available for all platforms and operating systems, but may only be needed for special environments or purposes.

The following function can be used to synchronize the CPU memory cache on systems without coherent IO (Nvidia Jetson Tegra TK1, TX1, TX2).

- [SCICacheSync\(\)](#)

2 File Documentation

2.1 sisci_api.h File Reference

Low-level SISI software functional specification.

Functions

- SISI_API_EXPORT void [SCISInitialize](#) (unsigned int flags, [sisci_error_t](#) *error)
Initializes the SISI library.
- SISI_API_EXPORT void [SCISTerminate](#) (void)
Terminates and releases resources associated with the SISI library.
- SISI_API_EXPORT void [SCISOpen](#) ([sisci_desc_t](#) *sd, unsigned int flags, [sisci_error_t](#) *error)
Opens an SISI virtual device.
- SISI_API_EXPORT void [SCISClose](#) ([sisci_desc_t](#) sd, unsigned int flags, [sisci_error_t](#) *error)
Closes an open SISI virtual device.
- SISI_API_EXPORT void [SCISConnectSegment](#) ([sisci_desc_t](#) sd, [sisci_remote_segment_t](#) *segment, unsigned int nodeId, unsigned int segmentId, unsigned int localAdapterNo, [sisci_cb_remote_segment_t](#) callback, void *callbackArg, unsigned int timeout, unsigned int flags, [sisci_error_t](#) *error)
Connects an application to a memory segment.
- SISI_API_EXPORT void [SCISDisconnectSegment](#) ([sisci_remote_segment_t](#) segment, unsigned int flags, [sisci_error_t](#) *error)
SCISDisconnectSegment() disconnects from the given mapped shared memory segment.
- SISI_API_EXPORT size_t [SCISGetRemoteSegmentSize](#) ([sisci_remote_segment_t](#) segment)
SCISGetRemoteSegmentSize() returns the size in bytes of a remote segment after it has been connected with SCISConnectSegment().
- SISI_API_EXPORT [sisci_segment_cb_reason_t](#) [SCISWaitForRemoteSegmentEvent](#) ([sisci_remote_segment_t](#) segment, [sisci_error_t](#) *status, unsigned int timeout, unsigned int flags, [sisci_error_t](#) *error)
SCISWaitForRemoteSegmentEvent() blocks a program until an event concerning the remote segment has arrived.

- SISI_API_EXPORT volatile void * [SCIMapRemoteSegment](#) ([sci_remote_segment_t](#) segment, [sci_map_t](#) *map, [size_t](#) offset, [size_t](#) size, void *addr, unsigned int flags, [sci_error_t](#) *error)
SCIMapRemoteSegment() maps an area of a remote segment connected with [SCIConnectSegment\(\)](#) into the addressable space of the program and returns a pointer to the beginning of the mapped area.
- SISI_API_EXPORT void * [SCIMapLocalSegment](#) ([sci_local_segment_t](#) segment, [sci_map_t](#) *map, [size_t](#) offset, [size_t](#) 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.
- 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\(\)](#).
- SISI_API_EXPORT void [SCICreateSegment](#) ([sci_desc_t](#) sd, [sci_local_segment_t](#) *segment, unsigned int segmentId, [size_t](#) 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.
- SISI_API_EXPORT [sci_segment_cb_reason_t](#) [SCIWaitForLocalSegmentEvent](#) ([sci_local_segment_t](#) segment, unsigned int *sourcenodeId, 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.
- SISI_API_EXPORT void [SCIPrepareSegment](#) ([sci_local_segment_t](#) segment, unsigned int localAdapterNo, unsigned int flags, [sci_error_t](#) *error)
SCIPrepareSegment() enables a local segment to be accessible from the specified network adapter.
- 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.
- 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.
- 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.
- SISI_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.
- SISI_API_EXPORT void [SCIRemoveSequence](#) ([sci_sequence_t](#) sequence, unsigned int flags, [sci_error_t](#) *error)
SCIRemoveSequence() destroys a sequence descriptor.
- 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 network adapter before starting a sequence of read and write operations on the concerned mapped segment.
- 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.
- SISI_API_EXPORT void [SCIStoreBarrier](#) ([sci_sequence_t](#) sequence, unsigned int flags)
SCIStoreBarrier() synchronizes all PIO accesses to a mapped segment.
- 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.
- SISI_API_EXPORT void [SCIAttachPhysicalMemory](#) ([sci_ioaddr_t](#) ioaddress, void *address, unsigned int busNo, [size_t](#) size, [sci_local_segment_t](#) segment, unsigned int flags, [sci_error_t](#) *error)

SISCI Privileged function [SCIAttachPhysicalMemory\(\)](#) enables usage of physical devices and memory regions where the Physical PCI/PCIe 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 an interface to request various information from the system, settings and interconnect status.
- 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)

The function [SCIGetNodeIdByAdapterName\(\)](#) provides an interface to query the nodeId and adapter type for an adapter in the cluster specified by its name.
- void [SCIGetNodeInfoByAdapterName](#) (char *adaptername, unsigned int *adapterNo, [dis_nodeId_list_t](#) *nodeIdList, [dis_adapter_type_t](#) *type, unsigned int flags, [sci_error_t](#) *error)

Function description missing.
- SISCI_API_EXPORT const char * [SCIGetErrorString](#) ([sci_error_t](#) error)

Get error description from a SISCI error code.
- 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.
- SISCI_API_EXPORT void [SCIAbortDMAQueue](#) ([sci_dma_queue_t](#) dq, unsigned int flags, [sci_error_t](#) *error)

[SCIAbortDMAQueue\(\)](#) aborts a DMA transfer initiated with [SCIStartDmaTransfer\(\)](#) or [SCIStartDmaTransferVec\(\)](#).
- SISCI_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](#)).
- SISCI_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.
- SISCI_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 makes it available to remote nodes and initializes a descriptor for the interrupt.
- SISCI_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.
- SISCI_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.
- SISCI_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\(\)](#)).
- SISCI_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.
- SISCI_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 **SCICreateDataInterrupt** (**sci_desc_t** sd, **sci_local_data_interrupt_t** *interrupt, unsigned int localAdapterNo, unsigned int *interruptNo, **sci_cb_data_interrupt_t** callback, void *callbackArg, unsigned int flags, **sci_error_t** *error)

SCICreateDataInterrupt() creates a data interrupt resource and makes it available to remote nodes and initializes a descriptor for the interrupt.
- SISI_API_EXPORT void **SCIRemoveDataInterrupt** (**sci_local_data_interrupt_t** interrupt, unsigned int flags, **sci_error_t** *error)

SCIRemoveDataInterrupt() deallocates a data interrupt resource and destroys the corresponding descriptor.
- SISI_API_EXPORT void **SCIWaitForDataInterrupt** (**sci_local_data_interrupt_t** interrupt, void *data, unsigned int *length, unsigned int timeout, unsigned int flags, **sci_error_t** *error)

SCIWaitForDataInterrupt() blocks a program until a data interrupt is received.
- SISI_API_EXPORT void **SCIConnectDataInterrupt** (**sci_desc_t** sd, **sci_remote_data_interrupt_t** *interrupt, unsigned int nodeId, unsigned int localAdapterNo, unsigned int interruptNo, unsigned int timeout, unsigned int flags, **sci_error_t** *error)

SCIConnectDataInterrupt() connects the caller to a data interrupt resource available on a remote node (see *SCICreateDataInterrupt()*).
- SISI_API_EXPORT void **SCIDisconnectDataInterrupt** (**sci_remote_data_interrupt_t** interrupt, unsigned int flags, **sci_error_t** *error)

SCIDisconnectDataInterrupt() disconnects an application from a remote data interrupt resource and deallocates the corresponding descriptor.
- SISI_API_EXPORT void **SCITriggerDataInterrupt** (**sci_remote_data_interrupt_t** interrupt, void *data, unsigned int length, unsigned int flags, **sci_error_t** *error)

SCITriggerDataInterrupt() sends an interrupt message to a remote node, after having connected to it with *SCIConnectDataInterrupt()*.
- SISI_API_EXPORT void **SCIMemWrite** (void *memAddr, volatile void *remoteAddr, **size_t** size, unsigned int flags, **sci_error_t** *error)

SCIMemWrite() transfers efficiently a block of data from local memory to a mapped segment using the shared memory mode.
- SISI_API_EXPORT void **SCIMemCpy** (**sci_sequence_t** sequence, void *memAddr, **sci_map_t** remoteMap, **size_t** remoteOffset, **size_t** 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 **SCIRegisterSegmentMemory** (void *address, **size_t** size, **sci_local_segment_t** segment, unsigned int flags, **sci_error_t** *error)

SCIRegisterSegmentMemory() associates an area memory allocated by the program (eg using malloc) with a local segment.
- SISI_API_EXPORT void **SCIAttachLocalSegment** (**sci_desc_t** sd, **sci_local_segment_t** *segment, unsigned int segmentId, **size_t** *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.
- 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 write combining buffers of the local system.
- SISI_API_EXPORT void **SCIStartDmaTransfer** (**sci_dma_queue_t** dq, **sci_local_segment_t** localSegment, **sci_remote_segment_t** remoteSegment, **size_t** localOffset, **size_t** size, **size_t** remoteOffset, **sci_cb_dma_t** callback, void *callbackArg, unsigned int flags, **sci_error_t** *error)

SCIStartDmaTransfer() starts the execution of a DMA queue.
- SISI_API_EXPORT void **SCIStartDmaTransferMem** (**sci_dma_queue_t** dq, void *localAddress, **sci_remote_segment_t** remoteSegment, **size_t** size, **size_t** remoteOffset, **sci_cb_dma_t** callback, void *callbackArg, unsigned int flags, **sci_error_t** *error)

- SCIStartDmaTransferMem* starts the execution of a DMA queue based on a user-allocated memory buffer.
- SISI_API_EXPORT void [SCIStartDmaTransferVec](#) ([sci_dma_queue_t](#) dq, [sci_local_segment_t](#) localSegment, [sci_remote_segment_t](#) remoteSegment, [size_t](#) vecLength, [dis_dma_vec_t](#) *disDmaVec, [sci_callback_dma_t](#) callback, void *callbackArg, unsigned int flags, [sci_error_t](#) *error)

SCIStartDmaTransferVec() starts the execution of a DMA queue.
 - SISI_API_EXPORT void [SCICacheSync](#) ([sci_map_t](#) map, void *addr, [size_t](#) length, unsigned int flags, [sci_error_t](#) *error)

SCICacheSync() is used to control the CPU cache.
 - SISI_API_EXPORT void [SCIRegisterPCleRequester](#) ([sci_desc_t](#) sd, unsigned int localAdapterNo, unsigned int bus, unsigned int devfn, unsigned int flags, [sci_error_t](#) *error)

SCIRegisterPCleRequester() registers a local PCIe requester with the NT function so that it can send traffic through the NTB.
 - SISI_API_EXPORT void [SCIUnregisterPCleRequester](#) ([sci_desc_t](#) sd, unsigned int localAdapterNo, unsigned int bus, unsigned int devfn, unsigned int flags, [sci_error_t](#) *error)

SCIUnregisterPCleRequester() unregisters a local PCIe requester from the NT function.
 - SISI_API_EXPORT void [SCIBorrowDevice](#) ([sci_desc_t](#) sd, [sci_device_t](#) *device, unsigned int deviceId, unsigned int flags, [sci_error_t](#) *error)

NOTE: This function is not yet finalized and may change without notice!
 - SISI_API_EXPORT void [SCIReturnDevice](#) ([sci_device_t](#) device, unsigned int flags, [sci_error_t](#) *error)

NOTE: This function is not yet finalized and may change without notice!
 - SISI_API_EXPORT void [SCIConnectDeviceMemory](#) ([sci_desc_t](#) sd, [sci_remote_segment_t](#) *segment, unsigned int localAdapterNo, [sci_device_t](#) device, unsigned int segmentNo, unsigned int timeout, unsigned int flags, [sci_error_t](#) *error)

NOTE: This function is not yet finalized and may change without notice!
 - SISI_API_EXPORT void [SCIMapLocalSegmentForDevice](#) ([sci_local_segment_t](#) segment, unsigned int localAdapterNo, [sci_device_t](#) device, [sci_ioaddr_t](#) *remoteAddr, unsigned int userData, unsigned int flags, [sci_error_t](#) *error)

NOTE: This function is not yet finalized and may change without notice!
 - SISI_API_EXPORT void [SCIUnmapLocalSegmentForDevice](#) ([sci_local_segment_t](#) segment, unsigned int localAdapterNo, [sci_device_t](#) device, unsigned int flags, [sci_error_t](#) *error)

NOTE: This function is not yet finalized and may change without notice!
 - SISI_API_EXPORT void [SCIMapRemoteSegmentForDevice](#) ([sci_remote_segment_t](#) segment, [sci_device_t](#) device, [sci_ioaddr_t](#) *remoteAddr, unsigned int userData, unsigned int flags, [sci_error_t](#) *error)

NOTE: This function is not yet finalized and may change without notice!
 - SISI_API_EXPORT void [SCIUnmapRemoteSegmentForDevice](#) ([sci_remote_segment_t](#) segment, [sci_device_t](#) device, unsigned int flags, [sci_error_t](#) *error)

NOTE: This function is not yet finalized and may change without notice!

2.1.1 Detailed Description

Low-level SISI software functional specification.

Remarks

The SISI API implementation from Dolphin is available with Dolphins IX, PX, MX and Intel NTB (INX) enabled PCI Express products and for various 3rd party OEM hardware solutions licensing the software.

Some extensions, for example Reflective Memory is only available for some hardware configurations. Please consult your hardware vendor and software release notes for details.

Please read the manual carefully and consult the available SISI example code found in the software distribution / SISI Devel package as well as the SISI Users guide available for download from <http://www.dolphinics.com>

2.1.2 Function Documentation

SISCI_API_EXPORT void SCIInitialize (unsigned int *flags*, sci_error_t * *error*)

Initializes the SISCI library.

[SCIInitialize\(\)](#) must be called before [SCIOpen\(\)](#).

Parameters

<i>flags</i>	see below
<i>error</i>	error information

Error codes:

- SCI_ERR_OK
Successful completion.
- No specific error codes for this function.

SISCI_API_EXPORT void SCITerminate (void)

Terminates and releases resources associated with the SISCI library.

[SCITerminate\(\)](#) must be called after [SCIClose\(\)](#).

Error codes:

- SCI_ERR_OK
Successful completion.
- No specific error codes for this function.

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

Opens an SISCI virtual device.

[SCIOpen\(\)](#) opens an SISCI virtual device, that is a channel to the driver. It creates and initializes a new descriptor for an SISCI virtual device, to be used in subsequent calls to API functions. A single virtual device can be used for all API functions, but only one of each resource type. E.g. if you want to do multiple connections, multiple virtual devices needs to be opened.

Parameters

<i>sd</i>	handle to the new SISCI virtual device descriptor
<i>flags</i>	not used
<i>error</i>	error information

Error codes:

- SCI_ERR_OK
Successful completion.
- No specific error codes for this function.

Note

- The virtual device handle can handle only one instance of each kind of SISCI descriptors. I.e., one virtual device can manage descriptors for one local segment, one remote segment, one sequence, one interrupt, one interrupt with data etc.

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

Closes an open SISCI virtual device.

[SCIClose\(\)](#) closes an open SISCI 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

<i>sd</i>	handle to an open SISCI virtual device descriptor
<i>flags</i>	not used
<i>error</i>	error information

Error codes:

- `SCI_ERR_OK`
Successful completion
- `SCI_ERR_BUSY`
Some resources depending on this virtual device are still in use

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.

[SCIConnectSegment\(\)](#) connects an application to a memory segment made available on a local or 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 [State diagram for remote segments](#). If a timeout different from `SCI_INFINITE_TIMEOUT` is passed to the function, the attempt to connect will stop after the specified number of milliseconds. Implementation notice: Dolphin has not implemented support for setting a timeout different from `SCI_INFINITE_TIMEOUT`.

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\(\)](#). Implementation notice: The `SCI_FLAG_ASYNCHRONOUS_CONNECT` is not implemented by Dolphin.

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

<i>sd</i>	handle to an open SISCI virtual device descriptor
-----------	---

Parameters

<i>segment</i>	handle to the new connected segment descriptor
<i>nodeId</i>	identifier of the node where the segment is allocated
<i>segmentId</i>	identifier of the segment to connect
<i>localAdapterNo</i>	number of the local adapter used for the connection
<i>callback</i>	function called when an asynchronous event affecting the segment occurs
<i>callbackArg</i>	user-defined parameter passed to the callback function
<i>timeout</i>	time in milliseconds to wait for the connection to complete. Currently not implemented, parameter should always be set to SCI_INFINITE_TIMEOUT
<i>flags</i>	see below
<i>error</i>	error information

Flags:

- SCI_FLAG_USE_CALLBACK
The specified callback is active
- SCI_FLAG_ASYNCHRONOUS_CONNECT
The connection is asynchronous. Not implemented.
- SCI_FLAG_BROADCAST
This flag must be set to enable the use of multicast and use the reflected memory mechanism. This function connects to all available remote broadcast segments with the same segmentId. The remote segments must be created with the function [SCICreateSegment\(\)](#) and with the SCI_FLAG_BROADCAST flag specified. `SCICreateSegment(...,SCI_FLAG_BROADCAST)`. This flag is only available for configurations supporting multicast.

Error codes:

- SCI_ERR_OK
Successful completion
- SCI_ERR_NO_SUCH_SEGMENT
The remote 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
Not possible to communicate via a remote switch port
- SCI_ERR_SYSTEM
The callback thread could not be created

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 [SCICreateSegment\(\)](#) 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 [SCIConnectSegment\(\)](#) the execution of `SCIDisconnectSegment` also generates an `SCI_CB_DISCONNECT` event directed to the application that created the segment (see [SCICreateSegment\(\)](#) and `sci_cb_local_segment_t`).

Parameters

<i>segment</i>	handle to the connected segment descriptor
<i>flags</i>	not used
<i>error</i>	error information

Error codes:

- `SCI_ERR_OK`
Successful completion
- `SCI_ERR_BUSY`
The segment is currently mapped or in use

`SISCI_API_EXPORT size_t SCIGetRemoteSegmentSize (sci_remote_segment_t segment)`

[SCIGetRemoteSegmentSize\(\)](#) returns the size in bytes of a remote segment after it has been connected with [SCIConnectSegment\(\)](#).

Parameters

<i>segment</i>	handle to the connected segment descriptor
----------------	--

Returns

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

Error codes:

- `SCI_ERR_OK`
Successful completion.
- No specific error codes for this function.

`SISCI_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

<i>segment</i>	handle to the connected segment descriptor
<i>status</i>	status information
<i>timeout</i>	time in milliseconds to wait before giving up
<i>flags</i>	not used
<i>error</i>	error information

Returns

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

Error codes:

- `SCI_ERR_OK`
Successful completion.
- `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

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

`SCIMapRemoteSegment()` maps an area of a remote segment connected with `SCIConnectSegment()` 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 read-only.

Flags:

- `SCI_FLAG_EXACT_MAP_RESOURCE`
The low level physical map will be occupying exact number of att entries. With this flag, only those att pages that are requested by the size parameter are marked as occupied.
- `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_IO_MAP_IOSPACE`
Mapping using non-prefetch space (iospace) No prefetching, or speculative hold enabled.
- `SCI_FLAG_BROADCAST`
This flag must be set to enable the use of multicast and use the reflected memory mechanism. Maps the remote segment into the addressable space of the program and returns a pointer to the beginning of the mapped remote broadcast segments. All remote segments are accessible through this pointer. This flag is only available for configurations supporting multicast.

Error codes:

- `SCI_ERR_OK`
Successful completion.
- `SCI_ERR_NOT_CONNECTED`
The links between local and remote node are not active.

- `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.
- `SCI_ERR_BUSY`
`SCI_FLAG_EXACT_MAP_RESOURCE` was called on the first or the second `SCIMapRemoteSegment` for the same `remote_segment_t` pointer.

`SISCI_API_EXPORT void* SCIMapLocalSegment (sci_local_segment_t segment, sci_map_t * map, size_t offset, size_t 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

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

Error codes:

- `SCI_ERR_OK`
Successful completion.
- `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.

`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()`.

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

Parameters

<i>map</i>	handle to the mapped segment descriptor
<i>flags</i>	not used
<i>error</i>	error information

Error codes:

- `SCI_ERR_OK`
Successful completion.
- `SCI_ERR_BUSY`
The map is currently in use.

`SISCI_API_EXPORT void SCICreateSegment (sci_desc_t sd, sci_local_segment_t * segment, unsigned int segmentId, size_t 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 [Local segment state diagram](#).

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. 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()`).

`SCICreateSegment()` will fail with the error code `SCI_ERR_NOSPC` if the driver / system did not manage to allocate the required amount of memory. Please consult the eXpressWare installation guide, section "Managing PCIe and eXpressWare Resources" for additional information on how to tune resources.

Parameters

<i>sd</i>	handle to an open SISCI virtual device descriptor
<i>segment</i>	handle to the new local segment descriptor
<i>segmentId</i>	segment identifier
<i>size</i>	segment size; if <code>SCI_FLAG_EMPTY</code> is specified, size means the maximum size of the memory area that can be associated with this local segment
<i>callback</i>	callback function called when an asynchronous event affecting the local segment occurs
<i>callbackArg</i>	user-defined argument passed to the callback function
<i>flags</i>	see below

Parameters

<i>error</i>	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.
- **SCI_FLAG_DMA_GLOBAL**
Setting this flag creates a segment that can only be used for global DMA. It cannot be mapped for PIO access or mapped DMA access.
- **SCI_FLAG_BROADCAST**
This flag must be set to enable use of the multicast and the reflected memory mechanism. Creates a segment for multicast/reflected memory capabilities. All segments in a broadcast group must have the same segmentId. This flag is only available for configurations supporting multicast.
- **SCI_FLAG_ALLOW_UNICAST**
This flag may be used in conjunction with **SCI_FLAG_BROADCAST** in order to allow regular unicast connections to this local segment. For Dolphin PCI Express PX this flag can only be used with group 0, no other regular multicast segment can be used simultaneously. Segments allocated with this flag are allocated from the general segment memory pool and so compete with regular segments.

Error codes:

- **SCI_ERR_OK**
Successful completion.
- **SCI_ERR_NOSPC**
Not able to allocate local memory resources. More details above.
- **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.

SISCI_API_EXPORT sci_segment_cb_reason_t SCIWaitForLocalSegmentEvent (**sci_local_segment_t** *segment*, unsigned int * *sourcenodeId*, 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.

If a timeout different from **SCI_INFINITE_TIMEOUT** is specified the function gives up when the timeout expires. [SCIWaitForLocalSegmentEvent\(\)](#) cannot be used if a callback associated with the local segment is active (see [SCICreateSegment\(\)](#)).

Parameters

<i>segment</i>	handle to local segment descriptor
<i>sourcenodeId</i>	identifier of the node that have generated the event

Parameters

<i>localAdapterNo</i>	number of the local adapter that receive the event
<i>timeout</i>	time in milliseconds to wait before giving up
<i>flags</i>	not used
<i>error</i>	error information

Returns

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

Error codes:

- **SCI_ERR_OK**
Successful completion.
- **SCI_ERR_TIMEOUT**
The function timed out after specified timeout value.
- **SCI_ERR_CANCELLED**
The wait operation has been cancelled due to a [SCIRemoveSegment\(\)](#) on the same handle. The handle is invalid when this error is returned.

SISCI_API_EXPORT void SCIPrepareSegment (sci_local_segment_t segment, unsigned int localAdapterNo, unsigned int flags, sci_error_t * error)

[SCIPrepareSegment\(\)](#) enables a local segment to be accessible from the specified network adapter.

Parameters

<i>segment</i>	handle to the local segment descriptor
<i>localAdapterNo</i>	Adapter number for which the segment is made available
<i>flags</i>	see below
<i>error</i>	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 SISCI driver to use performance improving features.
- **SCI_FLAG_BROADCAST**
This flag must be set to enable use of the multicast and the reflected memory mechanism. Creates a segment for multicast/reflected memory capabilities. All segments in a broadcast group must have the same segmentId and size. This flag is only available for configurations supporting multicast.

Error codes:

- **SCI_ERR_OK**
Successful completion.
- No specific error codes for this function.

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.

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. Before calling this function, the program should consider the use of [SCISetSegmentUnavailable\(\)](#) with the flags NOTIFY or FORCE_DISCONNECT.

Parameters

<i>segment</i>	handle to local segment descriptor
<i>flags</i>	see below
<i>error</i>	error information

Flags:

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

Error codes:

- `SCI_ERR_OK`
Successful completion.
- `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 support. 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.

`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.

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

<i>segment</i>	handle to local segment descriptor
<i>localAdapterNo</i>	number of the local adapter where the local segment is made available for connections
<i>flags</i>	not used.
<i>error</i>	error information

Error codes:

- `SCI_ERR_OK`
Successful completion.
- `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`.

`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.

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

<i>segment</i>	handle to the local segment descriptor
<i>localAdapterNo</i>	number of the local adapter where the local segment was made available
<i>flags</i>	see below
<i>error</i>	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

Error codes:

- `SCI_ERR_OK`
Successful completion.
- `SCI_ERR_ILLEGAL_OPERATION`
The operation is illegal in the current state of the 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.

Parameters

<i>map</i>	handle to a valid mapped segment descriptor
<i>sequence</i>	handle to the new sequence descriptor
<i>flags</i>	not used
<i>error</i>	error information

Error codes:

- `SCI_ERR_OK`
Successful completion.
- No specific errors for this function.

`SISCI_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

<i>sequence</i>	handle to the sequence descriptor
<i>flags</i>	not used
<i>error</i>	error information

Error codes:

- `SCI_ERR_OK`
Successful completion.
- No specific errors for this function.

`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 network 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

<i>sequence</i>	handle to the sequence descriptor
<i>flags</i>	not used
<i>error</i>	error information

Returns

- The function returns the status of the sequence.

Error codes:

- `SCI_ERR_OK`
Successful completion.
- No specific errors for this function.

`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.

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 any write buffers and to wait for all the outstanding write requests to be completed. To prevent this actions the caller has to use specific flags.

Parameters

<i>sequence</i>	handle to a sequence descriptor
<i>flags</i>	see below
<i>error</i>	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.

Error codes:

- SCI_ERR_OK
Successful completion.
- No specific error values for this function.

SISCI_API_EXPORT void SCIStoreBarrier (*sci_sequence_t sequence*, unsigned int *flags*)

[SCIStoreBarrier\(\)](#) synchronizes all PIO accesses to a mapped segment.

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

Parameters

<i>sequence</i>	handle to the mapped segment descriptor
<i>flags</i>	not used

Error codes:

- SCI_ERR_OK
Successful completion.
- No specific errors for this function.

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.

Parameters

<i>sd</i>	handle to an open SISI virtual device descriptor
<i>localAdapterNo</i>	number of the local adapter used for the check
<i>nodeId</i>	identifier of the remote node
<i>flags</i>	not used
<i>error</i>	error information

Returns

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

Error codes:

- `SCI_ERR_OK`
Successful completion, the node is currently reachable through the specified network.
- Errors if the function returns 1
 - `SCI_ERR_NO_LINK_ACCESS`
It was not possible to reach the node via the specified local adapter.
 - `SCI_ERR_NO_REMOTE_LINK_ACCESS`
It was not possible to communicate via a remote switch port.

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

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

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

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

Parameters

<i>ioaddress</i>	this is the address on the PCI bus that a PCI bus master has to use to write to the specified memory
<i>address</i>	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.
<i>busNo</i>	bus number where the device is located. Only required for SPARC system. Should be set to 0 for all other systems
<i>size</i>	size of the memory regions
<i>segment</i>	buffer
<i>flags</i>	see below
<i>error</i>	error information

Flags:

- `SCI_FLAG_CUDA_BUFFER`
Support CUDA managed GPU buffer - NVIDIA
- `SCI_FLAG_SCIF_BUFFER`

Support SCIF buffer - INTEL

Error codes:

- SCI_ERR_OK
Successful completion.
- No specific errors for this function.

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

SCIQuery() provides an interface to request various information from the system, settings and interconnect status.

The information can be vendor dependent, but some requests are specified in the API and must be satisfied: the vendor identifier, the version of the API implemented, and 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.

A query consist of a major-command and a sub-command.

Parameters

<i>command</i>	type of information required
<i>data</i>	generic data structure for possible sub-commands and output information
<i>flags</i>	specified per command / sub-command below
<i>error</i>	error information

Commands

- Major-commands are listed below:
- SCI_Q_ADAPTER
Major command for adapter queries. This query returns adapter specific information depending on the sub-command. The information is returned in the data structure.
- SCI_Q_SYSTEM
Major command for system queries. This query returns system specific information depending on the sub-command. The information is returned in the data structure.
- SCI_Q_DMA
Major command for DMA queries. This query return DMA specific information depending on the sub-command. The information is returned in the data structure.
- SCI_Q_LOCAL_SEGMENT
Major command for local segment queries. This query returns local segment specific information depending on the sub-command. The information is returned in the data structure.
- SCI_Q_REMOTE_SEGMENT
Major command for remote segment queries. This query returns remote segment specific information depending on the sub-command. The information is returned in the data structure.
- SCI_Q_MAP
Major command for local segment map queries. This query returns local segment map specific information depending on the sub-command. The information is returned in the data structure.
- SCI_Q_VENDORID
Major command for vendor id queries. The vendor identifier is returned in a data structure of type sci_query_string.
- SCI_Q_API
Major command for vendor id queries. The version of the API implemented is returned in a data structure of type sci_query_string.

- **SCI_Q_ADAPTER** sub-commands:

- **SCI_Q_ADAPTER_SERIAL_NUMBER:**
Returns the serial number of the local adapter
- **SCI_Q_ADAPTER_CARD_TYPE:**
Returns the card type of local adapter.
- **SCI_Q_ADAPTER_NODEID:**
Returns the node id of local adapter.
- **SCI_Q_ADAPTER_LINK_OPERATIONAL:**
Returns true if the local link is operational.
- **SCI_Q_ADAPTER_CONFIGURED:**
Returns true if the local adapter is configured.
- **SCI_Q_ADAPTER_LINK_WIDTH:**
Returns the PCIe link width for the specified link port.
- **SCI_Q_ADAPTER_LINK_SPEED:**
Returns the PCIe link speed for the specified link port.
- **SCI_Q_ADAPTER_LINK_UPTIME:**
Returns the seconds of link uptime for the specified link port.
- **SCI_Q_ADAPTER_LINK_DOWNTIME:**
Returns the seconds of link downtime for the specified link port.
- **SCI_Q_ADAPTER_LINK_OPERATIONAL:**
Returns status for the specified link port.
- **SCI_Q_ADAPTER_LINK_CABLE_INSERTED:**
Returns if the cable is inserted for the specified link port.
- **SCI_Q_ADAPTER_LINK_ENABLED:**
Returns if the link is enabled for the specified link port.
- **SCI_Q_ADAPTER_LINK_PARTNER_PORT_NO:**
Returns the partner (remote) link port number for the specified link port.
- **SCI_Q_ADAPTER_NUMBER_OF_LINKS:**
Returns the number of adapter link ports that are enabled.
- **SCI_Q_ADAPTER_DMA_MTU:**
Returns the max transfer unit (MTU) of the DMA engine of the adapter.
Flags: **SCI_FLAG_DMA_SYSDMA** - return the MTU of the system DMA.
- **SCI_Q_ADAPTER_MCAST_MAX_GROUPS:**
Returns the number of available multicast groups.

- **SCI_Q_DMA** sub-commands:

- **SCI_Q_DMA_AVAILABLE:**
Returns whether the DMA mode specified in "flags" is available.
- **SCI_Q_DMA_CAPABILITIES:**
Returns the capabilities of the DMA mode specified in "flags".
- **Flags:** **SCI_FLAG_DMA_ADAPTER**, **SCI_FLAG_DMA_GLOBAL**, **SCI_FLAG_DMA_SYSDMA**

- **SCI_Q_LOCAL_SEGMENT** sub-commands:

- **SCI_Q_LOCAL_SEGMENT_IOADDR:**
Returns the local io-address of the local segment.
- **SCI_Q_LOCAL_SEGMENT_VIRTUAL_KERNEL_ADDR:**
Returns the local virtual kernel address of the local segment.
- **SCI_Q_LOCAL_SEGMENT_PHYS_ADDR :**
Returns the local physical address of the local segment.

- SCI_Q_REMOTE_SEGMENT sub-commands:

- SCI_Q_REMOTE_SEGMENT_IOADDR:
Returns the local io-address of the local segment.

Error codes:

- SCI_ERR_OK
Successful completion.
- SCI_ERR_ILLEGAL_QUERY
Unrecognized command.

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

Get local node id.

Parameters

<i>adapterNo</i>	number of the local adapter to get node id
<i>nodeId</i>	identifier of the local node
<i>flags</i>	not used
<i>error</i>	error information

Error codes:

- SCI_ERR_OK
Successful completion.
- No specific errors for this function.

SISCI_API_EXPORT void SCIGetNodeidByAdapterName (char * *adaptername*, dis_nodeid_list_t * *nodeId*, dis_adapter_type_t * *type*, unsigned int *flags*, sci_error_t * *error*)

The function SCIGetNodeidByAdapterName() provides an interface to query the nodeId and adapter type for an adapter in the cluster specified by its name.

The local dishosts.conf file specifies the adapter name to nodeId map.

Parameters

<i>adaptername</i>	name of the adapter to query node id
<i>nodeId</i>	nodeid associated with the specified adapter name
<i>type</i>	adapter type
<i>flags</i>	not used
<i>error</i>	error information

Error codes:

- SCI_ERR_OK
Successful completion.
- No specific errors for this function.

```
void SCIGetNodeInfoByAdapterName ( char * adaptername, unsigned int * adapterNo, dis_nodeld_list_t * nodeldlist,
dis_adapter_type_t * type, unsigned int flags, sci_error_t * error )
```

Function description missing.

Parameters

<i>adaptername</i>	names of the adapters to query node ids
<i>adapterNo</i>	number of local adapter with the given adapter name
<i>nodeldlist</i>	
<i>type</i>	
<i>flags</i>	not used
<i>error</i>	error information

Error codes:

- SCI_ERR_OK
Successful completion.
- No specific errors for this function.

```
SISCI_API_EXPORT const char* SCIGetErrorString ( sci_error_t error )
```

Get error description from a SISCO error code.

[SCIGetErrorString\(\)](#) was introduced in DIS release 5.5.0

Parameters

<i>error</i>	SISCO error code
--------------	------------------

Returns

- The function returns a string describing the error.

```
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.

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

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

Error codes:

- `SCI_ERR_OK`
Successful completion.
- No specific errors for this function.

`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.

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

<i>dq</i>	handle to the DMA queue descriptor
<i>flags</i>	not used
<i>error</i>	error information

Error codes:

- `SCI_ERR_OK`
Successful completion.
- `SCI_ERR_ILLEGAL_OPERATION`
Not allowed in this queue state.

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

`SCIAbortDMAQueue()` aborts a DMA transfer initiated with `SCIStartDmaTransfer()` or `SCIStartDmaTransferVec()`.

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

<i>dq</i>	handle to the DMA queue descriptor
<i>flags</i>	not used
<i>error</i>	error information

Error codes:

- `SCI_ERR_OK`
Successful completion.
- `SCI_ERR_ILLEGAL_OPERATION`
Illegal operation

SISCI_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

<i>dq</i>	handle to the DMA queue descriptor
-----------	------------------------------------

Returns

- The function returns the state of the DMA queue.

Error codes:

- SCI_ERR_OK
Successful completion.
- No specific errors for this function.

SISCI_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

<i>dq</i>	handle to a DMA queue descriptor
<i>timeout</i>	timeout in milliseconds to wait before giving up
<i>flags</i>	not used
<i>error</i>	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.

Error codes:

- SCI_ERR_OK
Successful completion.
- 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.

SISCI_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 makes it available to remote nodes and initializes a descriptor for the interrupt.

An interrupt is associated by the driver with a unique number.

There is normally not a one to one relation between triggered and received interrupts. Several interrupts may be collapsed into one remote interrupt if interrupts are sent faster than the remote system can handle. The SISCI driver will ensure that at least one interrupt event is seen by the remote system, i.e. the callback will be invoked or [SCIWaitForInterrupt\(\)](#) will wake up at least once, respectively.

If the flag SCI_FLAG_FIXED_INTNO is specified, the function tries to use the number passed by the caller.

Parameters

<i>sd</i>	handle to an open SISCI virtual device descriptor
<i>interrupt</i>	handle to the new interrupt descriptor
<i>localAdapterNo</i>	number of the local adapter used to make the interrupt
<i>interruptNo</i>	number assigned to the interrupt
<i>callback</i>	function called when the interrupt is triggered
<i>callbackArg</i>	user-defined parameter passed to the callback function
<i>flags</i>	see below
<i>error</i>	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
The interrupt can be shared by several, everybody will receive the interrupt notification.
- 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. Interrupts are costly, it is recommended to not use this flag, but use a shared memory location to count the number of invocations.

Error codes:

- SCI_ERR_OK
Successful completion.
- SCI_ERR_INTNO_USED
This interrupt number is already used
- SCI_ERR_SYSTEM
The callback thread could not be created

SISCI_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

<i>interrupt</i>	handle to the local interrupt descriptor
<i>flags</i>	not used
<i>error</i>	error information

Error codes:

- **SCI_ERR_OK**
Successful completion.
- **SCI_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.

SISCI_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

<i>interrupt</i>	handle to the local interrupt descriptor
<i>timeout</i>	time in milliseconds to wait before giving up
<i>flags</i>	not used
<i>error</i>	error information

Error codes:

- **SCI_ERR_OK**
Successful completion.
- **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.

SISCI_API_EXPORT void [SCIConnectInterrupt](#) (**sci_desc_t** *sd*, **sci_remote_interrupt_t** * *interrupt*, unsigned int *nodeld*, 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

<i>sd</i>	handle to an open SISCI virtual device descriptor
<i>interrupt</i>	handle to a new remote interrupt descriptor
<i>nodeld</i>	identifier of the remote node where the interrupt has been created
<i>localAdapterNo</i>	number of the local adapter used for the connection

Parameters

<i>interruptNo</i>	number assigned to the interrupt
<i>timeout</i>	time in milliseconds to wait before giving up. Not implemented, should always be set to SCI_INFINITE_TIMEOUT
<i>flags</i>	see below
<i>error</i>	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 a remote interrupt generated after the first and after the last trigger. If an interrupt is triggered faster than the remote interrupt handler can handle, interrupts may be lost.

Error codes:

- SCI_ERR_OK
Successful completion.
- 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.

SISCI_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

<i>interrupt</i>	handle to the remote interrupt descriptor
<i>flags</i>	not used
<i>error</i>	error information

Error codes:

- SCI_ERR_OK
Successful completion.
- No specific errors for this function.

SISCI_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 [SCICheckInterrupt\(\)](#).

What happens to the remote application that made the interrupt resource available depends on what it specified at the time it called [SCICreateInterrupt\(\)](#).

Parameters

<i>interrupt</i>	handle to the remote interrupt descriptor
<i>flags</i>	not used
<i>error</i>	error information

Error codes:

- SCI_ERR_OK
Successful completion.
- No specific errors for this function.

SISCI_API_EXPORT void SCICreateDataInterrupt (sci_desc_t *sd*, sci_local_data_interrupt_t * *interrupt*, unsigned int *localAdapterNo*, unsigned int * *interruptNo*, sci_cb_data_interrupt_t *callback*, void * *callbackArg*, unsigned int *flags*, sci_error_t * *error*)

[SCICreateDataInterrupt\(\)](#) creates a data interrupt resource and makes it available to remote nodes and initializes a descriptor for the interrupt.

A data 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

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

Error codes:

- SCI_ERR_OK \n
Successful completion.
- SCI_ERR_INTNO_USED \n
This interrupt number is already used.
- SCI_ERR_SYSTEM \n
The callback thread could not be created.

SISCI_API_EXPORT void SCIRemoveDataInterrupt (sci_local_data_interrupt_t *interrupt*, unsigned int *flags*, sci_error_t * *error*)

[SCIRemoveDataInterrupt\(\)](#) deallocates a data interrupt resource and destroys the corresponding descriptor.

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

Parameters

<i>interrupt</i>	handle to the local data interrupt descriptor
<i>flags</i>	not used
<i>error</i>	error information

Error codes:

- **SCI_ERR_OK**
Successful completion.
- **SCI_ERR_BUSY**
The resource is used by a remote node. [SCIRemoveDataInterrupt\(\)](#) 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.

SISCI_API_EXPORT void SCIWaitForDataInterrupt (sci_local_data_interrupt_t interrupt, void * data, unsigned int * length, unsigned int timeout, unsigned int flags, sci_error_t * error)

[SCIWaitForDataInterrupt\(\)](#) blocks a program until a data interrupt is received.

If a timeout different from **SCI_INFINITE_TIMEOUT** is specified the function gives up when the timeout expires. [SCIWaitForDataInterrupt\(\)](#) cannot be used if a callback associated with the data interrupt is active (see [SCICreateInterrupt\(\)](#)).

Parameters

<i>interrupt</i>	handle to the local data interrupt descriptor
<i>data</i>	pointer to data buffer
<i>length</i>	[in] length of data buffer; [out] actual data size received
<i>timeout</i>	time in milliseconds to wait before giving up
<i>flags</i>	not used
<i>error</i>	error information

Error codes:

- **SCI_ERR_OK**
Successful completion.
- **SCI_ERR_TIMEOUT**
The function timed out after specified timeout value.
- **SCI_ERR_CANCELLED**
The wait was interrupted by a call to [SCIRemoveDataInterrupt\(\)](#) or by the arrival of a signal. The handle is invalid when this error code is returned.
- **SCI_ERR_OUT_OF_RANGE**
The actual data size exceeded the data buffer length.

SISCI_API_EXPORT void SCIConnectDataInterrupt (sci_desc_t sd, sci_remote_data_interrupt_t * interrupt, unsigned int nodeld, unsigned int localAdapterNo, unsigned int interruptNo, unsigned int timeout, unsigned int flags, sci_error_t * error)

[SCIConnectDataInterrupt\(\)](#) connects the caller to a data interrupt resource available on a remote node (see [SCICreateDataInterrupt\(\)](#)).

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

Parameters

<i>sd</i>	handle to an open SISI virtual device descriptor
<i>interrupt</i>	handle to a new remote data interrupt descriptor
<i>nodeId</i>	identifier of the remote node where the interrupt has been created
<i>localAdapterNo</i>	number of the local adapter used for the connection
<i>interruptNo</i>	number assigned to the interrupt
<i>timeout</i>	time in milliseconds to wait before giving up. Not implemented, should always be set to <code>SCI_INFINITE_TIMEOUT</code>
<i>flags</i>	see below
<i>error</i>	error information

Flags:

- `SCI_FLAG_BROADCAST`
The data interrupt will be forwarded to all nodes.

Error codes:

- `SCI_ERR_OK`
Successful completion.
- `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.

`SISCI_API_EXPORT void SCIDisconnectDataInterrupt (sci_remote_data_interrupt_t interrupt, unsigned int flags, sci_error_t * error)`

[SCIDisconnectDataInterrupt\(\)](#) disconnects an application from a remote data interrupt resource and deallocates the corresponding descriptor.

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

Parameters

<i>interrupt</i>	handle to the remote data interrupt descriptor
<i>flags</i>	not used
<i>error</i>	error information

Error codes:

- `SCI_ERR_OK`
Successful completion.
- No specific errors for this function.

SISCI_API_EXPORT void SCITriggerDataInterrupt (sci_remote_data_interrupt_t *interrupt*, void * *data*, unsigned int *length*, unsigned int *flags*, sci_error_t * *error*)

SCITriggerDataInterrupt() sends an interrupt message to a remote node, after having connected to it with SCIConnectDataInterrupt().

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

Parameters

<i>interrupt</i>	handle to the remote data interrupt descriptor
<i>data</i>	pointer to data buffer
<i>length</i>	length of data
<i>flags</i>	not used
<i>error</i>	error information

Error codes:

- SCI_ERR_OK
Successful completion.
- No specific errors for this function.

SISCI_API_EXPORT void SCIMemWrite (void * *memAddr*, volatile void * *remoteAddr*, size_t *size*, unsigned int *flags*, sci_error_t * *error*)

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

Parameters

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

Flags:

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

Error codes:

- SCI_ERR_OK
Successful completion.
- 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

SISCI_API_EXPORT void SCIMemCpy (sci_sequence_t sequence, void * memAddr, sci_map_t remoteMap, size_t remoteOffset, size_t 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

<i>sequence</i>	handle to the sequence descriptor
<i>memAddr</i>	base address in virtual memory of the source
<i>remoteMap</i>	handle to the descriptor of the mapped segment that is the destination of the transfer
<i>remoteOffset</i>	offset inside the mapped segment where the transfer should start
<i>size</i>	size of the transfer
<i>flags</i>	see below
<i>error</i>	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

Error codes:

- SCI_ERR_OK
Successful completion.
- 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.

SISCI_API_EXPORT void SCiRegisterSegmentMemory (void * address, size_t size, sci_local_segment_t segment, unsigned int flags, sci_error_t * error)

[SCiRegisterSegmentMemory\(\)](#) associates an area memory allocated by the program (eg using malloc) with a local segment.

The segment must have been 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.

[SCiRegisterSegmentMemory\(\)](#) was implemented in DIS release 5.5.0 for Linux running on Intel systems.

Please note that this function requires the Input/Output Memory Management Unit (IOMMU, Intel VT-d) to be enabled in the BIOS and Linux booted with the IOMMU on (boot param intel_iommu=on).

Parameters

<i>address</i>	base address of the user-allocated memory in the programs virtual address space
<i>size</i>	size of the user-allocated memory to be associated with the local segment
<i>segment</i>	handle to local segment descriptor
<i>flags</i>	see below
<i>error</i>	error information

Flags: *

- **SCI_FLAG_LOCK_USER_MEM**
Pin the pages of the user-allocated buffer in physical memory. This may offer a performance advantage when the buffer is repeatedly used for DMA operations. [SCIPrepareSegment\(\)](#) will pin the pages if this flag is not specified and the segment will be used for remote PIO.

Error codes:

- **SCI_ERR_OK**
Successful completion.
- **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.
- **SCI_ERR_NOT_SUPPORTED**
Platform or operating system does not support [SCIRegisterSegmentMemory\(\)](#)

SISCI_API_EXPORT void SCIAttachLocalSegment (sci_desc_t sd, sci_local_segment_t * segment, unsigned int segmentId, size_t * 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.

Error codes:

- **SCI_ERR_OK**
Successful completion.
- **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()`.

`SISCI_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

<i>segment</i>	handle to the descriptor of local segment.
<i>flags</i>	not used
<i>error</i>	error information

Error codes:

- `SCI_ERR_OK`
Successful completion.
- No specific error information provided.

`SISCI_API_EXPORT void SCIFlush (sci_sequence_t sequence, unsigned int flags)`

`SCIFlush()` flushes the CPU write combining buffers of the local system.

This function will make sure all data previously written to a remote segment, that may be residing in a local CPU cache etc, will be flushed out of the local system. The data may still be on its way through the interconnect when the function returns.

`SCICheckSequence()` should be used if the application wants to verify data has reached the destination memory.

Parameters

<i>sequence</i>	handle to the sequence descriptor
<i>flags</i>	see below

Flags:

- `SCI_FLAG_FLUSH_CPU_BUFFERS_ONLY`
Do not flush Dolphin SCI Write combining buffers. The flag is supported on all architectures but ignored for all except SCI.

Error codes:

- No error information provided by this function.

```
SISCI_API_EXPORT void SCIStartDmaTransfer ( sci_dma_queue_t dq, sci_local_segment_t localSegment,
sci_remote_segment_t remoteSegment, size_t localOffset, size_t size, size_t remoteOffset, sci_cb_dma_t callback,
void * callbackArg, unsigned int flags, sci_error_t * error )
```

[SCIStartDmaTransfer\(\)](#) starts the execution of a DMA queue.

The contents of the local segment is transferred into the remote segment using localOffset and remoteOffset. The local segment must be prepared before this function is called. The remote segment must be mapped with [SCIMapRemoteSegment\(\)](#) prior to starting the transfer depending on the chosen DMA mode (see Flags below).

The function returns as soon as the transfer specifications contained in the queue are passed to the DMA engine. 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 [SCIWaitForDMAQueue\(\)](#). An application is allowed to start another transfer on a queue only after the previous transfer for that queue has completed.

Note that that DMA support and available modes may vary according to your specific adapter card and system. Please consult the technical specifications for more information.

Parameters

<i>dq</i>	handle to the DMA queue descriptor
<i>localSegment</i>	handle to the local segment descriptor
<i>remoteSegment</i>	handle to the remote segment descriptor
<i>localOffset</i>	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)
<i>size</i>	size of the data to be transferred
<i>remoteOffset</i>	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)
<i>callback</i>	callback function to be invoked when all the DMA transfers have completed or in case an error occurs during a transfer
<i>callbackArg</i>	user-defined parameter passed to the callback function
<i>flags</i>	see below
<i>error</i>	error information

Flags:

- **SCI_FLAG_USE_CALLBACK**
The end of the transfer will cause the callback function to be invoked.
- **SCI_FLAG_DMA_READ**
Reverse the transfer direction and make the DMA engine read from the remote segment into the local segment. NOTE: Read operations may achieve lower bandwidth than write operations. It may be beneficial to make the remote node perform the transfer rather than to pass this flag.
- **SCI_FLAG_BROADCAST**
This flag must be set to enable the use of multicast and use the reflected memory mechanism. This function connects to all available remote broadcast segments with the same segmentId. The remote segments must be created with the function [SCICreateSegment\(\)](#) and with the SCI_FLAG_BROADCAST flag specified. [SCICreateSegment\(...,SCI_FLAG_BROADCAST\)](#). This flag is only available for configurations supporting multicast.
- **SCI_FLAG_DMA_GLOBAL**
Use global DMA which does not require the remote segment to be mapped with [SCIMapRemoteSegment](#).
- **SCI_FLAG_DMA_SYSDMA**
Use the DMA engine provided by the host platform and OS instead of the DMA engine on the adapter. Currently Linux is supported. Cannot be combined with [SCI_FLAG_DMA_GLOBAL](#).

Error codes:

- `SCI_ERR_OK`
Successful completion.
- `SCI_ERR_ILLEGAL_OPERATION`
Illegal operation
- `SCI_ERR_SYSTEM`
The callback thread could not be created
- `SCI_ERR_DMA_NOT_AVAILABLE`
The requested or required DMA mode is not available
- `SCI_ERR_DMA_DISABLED`
The requested or required DMA mode is disabled

```
SISCI_API_EXPORT void SCIStartDmaTransferMem ( sci_dma_queue_t dq, void * localAddress, sci_remote_segment_t
remoteSegment, size_t size, size_t remoteOffset, sci_cb_dma_t callback, void * callbackArg, unsigned int flags,
sci_error_t * error )
```

`SCIStartDmaTransferMem` starts the execution of a DMA queue based on a user-allocated memory buffer.

The contents of the buffer pointed to by `localAddress` is transferred into the remote segment using `remoteOffset`. The remote segment must be mapped with `SCIMapRemoteSegment()` prior to starting the transfer depending on the chosen DMA mode (see Flags below).

The function returns as soon as the transfer specifications contained in the queue are passed to the DMA engine. 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 `SCIWaitForDMAQueue()`. An application is allowed to start another transfer on a queue only after the previous transfer for that queue has completed.

Note that that DMA support and available modes may vary according to your specific adapter card and system. Please consult the technical specifications for more information.

`SCIStartDmaTransferMem()` was introduced in DIS release 5.5.0 for Linux and Windows.

Parameters

<i>dq</i>	handle to the DMA queue descriptor
<i>localAddress</i>	pointer to a user-allocated memory buffer
<i>remoteSegment</i>	handle to the remote segment descriptor
<i>size</i>	size of the data to be transferred
<i>remoteOffset</i>	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)
<i>callback</i>	callback function to be invoked when all the DMA transfers have completed or in case an error occurs during a transfer
<i>callbackArg</i>	user-defined parameter passed to the callback function
<i>flags</i>	see below
<i>error</i>	error information

Flags:

- `SCI_FLAG_USE_CALLBACK`
The end of the transfer will cause the callback function to be invoked.
- `SCI_FLAG_DMA_READ`
Reverse the transfer direction and make the DMA engine read from the remote segment into the local seg-

ment. NOTE: Read operations may achieve lower bandwidth than write operations. It may be beneficial to make the remote node perform the transfer rather than to pass this flag.

- **SCI_FLAG_BROADCAST**
This flag must be set to enable the use of multicast and use the reflected memory mechanism. This function connects to all available remote broadcast segments with the same segmentId. The remote segments must be created with the function [SCICreateSegment\(\)](#) and with the SCI_FLAG_BROADCAST flag specified. `SCICreateSegment(...,SCI_FLAG_BROADCAST)`. This flag is only available for configurations supporting multicast.
- **SCI_FLAG_DMA_GLOBAL**
Use global DMA which does not require the remote segment to be mapped with [SCIMapRemoteSegment](#).
- **SCI_FLAG_DMA_SYSDMA**
Use the DMA engine provided by the host platform and OS instead of the DMA engine on the adapter. Currently Linux is supported. Cannot be combined with `SCI_FLAG_DMA_GLOBAL`.

Error codes:

- **SCI_ERR_OK**
Successful completion.
- **SCI_ERR_ILLEGAL_OPERATION**
Illegal operation
- **SCI_ERR_SYSTEM**
The callback thread could not be created
- **SCI_ERR_DMA_NOT_AVAILABLE**
The requested or required DMA mode is not available
- **SCI_ERR_DMA_DISABLED**
The requested or required DMA mode is disabled

```
SISCI_API_EXPORT void SCIStartDmaTransferVec ( sci_dma_queue_t dq, sci_local_segment_t localSegment,
sci_remote_segment_t remoteSegment, size_t vecLength, dis_dma_vec_t * disDmaVec, sci_cb_dma_t callback, void *
callbackArg, unsigned int flags, sci_error_t * error )
```

[SCIStartDmaTransferVec\(\)](#) starts the execution of a DMA queue.

The contents of the local segment is transferred into the remote segment using the offsets in the vector. Each vector element contains a size, local and remote offset. The local segment must be prepared before this function is called. The remote segment must be mapped with [SCIMapRemoteSegment\(\)](#) prior to starting the transfer depending on the chosen DMA mode (see Flags below).

The function returns as soon as the transfer specifications contained in the queue are passed to the DMA engine. 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 [SCIWaitForDMAQueue\(\)](#). An application is allowed to start another transfer on a queue only after the previous transfer for that queue has completed.

Note that that DMA support and available modes may vary according to your specific adapter card and system. Please consult the technical specifications for more information.

Parameters

<i>dq</i>	handle to the DMA queue descriptor
<i>localSegment</i>	handle to the local segment descriptor
<i>remoteSegment</i>	handle to the remote segment descriptor
<i>vecLength</i>	length of the DMA vector queue.
<i>disDmaVec</i>	handle to the DMA vector queue.

Parameters

<i>callback</i>	callback function to be invoked when all the DMA transfers have completed or in case an error occurs during a transfer
<i>callbackArg</i>	user-defined parameter passed to the callback function
<i>flags</i>	see below
<i>error</i>	error information

Flags:

- **SCI_FLAG_USE_CALLBACK**
The end of the transfer will cause the callback function to be invoked.
- **SCI_FLAG_DMA_READ**
Reverse the transfer direction and make the DMA engine read from the remote segment into the local segment. NOTE: Read operations may achieve lower bandwidth than write operations. It may be beneficial to make the remote node perform the transfer rather than to pass this flag.
- **SCI_FLAG_DMA_WAIT**
The call to this function will block until the transfer has completed.
- **SCI_FLAG_BROADCAST**
This flag must be set to enable the use of multicast and use the reflected memory mechanism. This function connects to all available remote broadcast segments with the same segmentId. The remote segments must be created with the function [SCICreateSegment\(\)](#) and with the SCI_FLAG_BROADCAST flag specified. See [SCICreateSegment\(...,SCI_FLAG_BROADCAST\)](#). This flag is only available for configurations supporting multicast.
- **SCI_FLAG_DMA_GLOBAL**
Use global DMA which does not require the remote segment to be mapped with [SCIMapRemoteSegment](#).
- **SCI_FLAG_DMA_SYSDMA**
Use the DMA engine provided by the host platform and OS instead of the DMA engine on the adapter. Currently Linux is supported. Cannot be combined with **SCI_FLAG_DMA_GLOBAL**.

Error codes:

- **SCI_ERR_OK**
Successful completion
- **SCI_ERR_ILLEGAL_OPERATION**
Illegal operation
- **SCI_ERR_SYSTEM**
The callback thread could not be created
- **SCI_ERR_DMA_NOT_AVAILABLE**
The requested or required DMA mode is not available
- **SCI_ERR_DMA_DISABLED**
The requested or required DMA mode is disabled

```
SISCI_API_EXPORT void SCICacheSync ( sci_map_t map, void * addr, size_t length, unsigned int flags, sci_error_t * error )
```

[SCICacheSync\(\)](#) is used to control the CPU cache.

This function is only needed on platforms where the hardware does NOT provide a coherent IO (cache) system. This function is currently only needed for the Tegra K1 and X1. Users of all other platforms can ignore this function.

Platforms without IO cache coherency requires special care when IO devices and the CPU is operating on the same region in memory. This applies to local segments that are exported or used for DMA.

On cache incoherent platforms, writes by the CPU to local segments may not be visible by remote nodes and DMA until the CPU cache is flushed. To avoid remote nodes seeing stale data, programs may call `SCICacheSync` with the `SCI_FLAG_CACHE_FLUSH` flag. In the reverse direction, the CPU may not see changes by a remote node, or DMA engine until the CPU has invalidated its CPU cache. Programs may invalidate the CPU cache by calling `SCICacheSync` with the `SCI_FLAG_CACHE_INVALIDATE` flag.

The function will always sync at least the range specified, but may operate on bytes preceding and following if the address and/or length is not aligned to the CPU cache line size.

The function is available and can be called on all platforms but will immediately return with no side effects unless used on a system where the CPU cache must be managed.

Parameters

<i>map</i>	The local map handle.
<i>addr</i>	The virtual address pointing to the first byte to be synced. This address will be aligned down to the nearest CPU cache line.
<i>length</i>	The number of bytes to be synced. Will be rounded up to the CPU cache line size.
<i>flags</i>	Controls the cache operation. Both <code>FLUSH</code> and <code>INVALIDATE</code> may be given at the same time as a logical OR of the flags. In this case the range will first be flushed and then invalidated. Some platforms may not support all combinations.
<i>error</i>	Error information.

Flags:

- `SCI_FLAG_CACHE_FLUSH`
Flush any dirty cache lines in the range specified from the CPU cache all the way to main memory, overwriting the range in main memory.
- `SCI_FLAG_CACHE_INVALIDATE`
Invalidate all cache lines specified from the CPU cache discarding any changes by the CPU not flushed to main memory.

Error codes:

- `SCI_ERR_OK`
Successful completion.
- `SCI_ERR_ILLEGAL_ADDRESS`
Illegal address.
- `SCI_ERR_ILLEGAL_PARAMETER`
Map is not valid.
- `SCI_ERR_NOT_SUPPORTED`
The flags specifies an unsupported combination (for this platform).
- `SCI_ERR_ILLEGAL_OPERATION`
Invalid flags.
- `SCI_ERR_OUT_OF_RANGE`
Range specified is outside the local segment.

```
SISCI_API_EXPORT void SCIRegisterPCleRequester ( sci_desc_t sd, unsigned int localAdapterNo, unsigned int bus,
unsigned int devfn, unsigned int flags, sci_error_t * error )
```

[SCIRegisterPCleRequester\(\)](#) registers a local PCIe requester with the NT function so that it can send traffic through the NTB.

The corresponding [SCIUnregisterPCleRequester\(\)](#) should be called before the application terminates.

Parameters

<i>sd</i>	Handle to an open SISI virtual device descriptor.
<i>localAdapterNo</i>	Number of the local adapter used for the check.
<i>bus</i>	Bus number of the device.
<i>devfn</i>	Device and function number of the device.
<i>flags</i>	<ul style="list-style-type: none"> • SCI_FLAG_BROADCAST - allow device to generate only broadcast (multicast) traffic. • SCI_FLAG_PCIE_REQUESTER_GLOBAL - allow device to access remote memory. (SCI_FLAG_PCIE_REQUESTER_GLOBAL implies SCI_FLAG_BROADCAST)
<i>error</i>	Error information.

Error codes:

- SCI_ERR_OK
Successful completion.
- SCI_ERR_NOSPC
It was not possible to register the device.

SISCI_API_EXPORT void SCIUnregisterPCleRequester (*sci_desc_t sd*, unsigned int *localAdapterNo*, unsigned int *bus*, unsigned int *devfn*, unsigned int *flags*, *sci_error_t * error*)

[SCIUnregisterPCleRequester\(\)](#) unregisters a local PCIe requester from the NT function.

The PCIe requester must previously have been registered using the [SCIRegisterPCleRequester\(\)](#) function.

Parameters

<i>sd</i>	Handle to an open SISI virtual device descriptor.
<i>localAdapterNo</i>	Number of the local adapter used for the check.
<i>bus</i>	Bus number of the device.
<i>devfn</i>	Device and function number of the device.
<i>flags</i>	Not used.
<i>error</i>	Error information.

Error codes:

- SCI_ERR_OK
Successful completion.
- SCI_ERR_NOSPC
It was not possible to register the device.

SISCI_API_EXPORT void SCIBorrowDevice (*sci_desc_t sd*, *sci_device_t * device*, unsigned int *deviceld*, unsigned int *flags*, *sci_error_t * error*)

NOTE: This function is not yet finalized and may change without notice!

Borrows a SmartIO device. By default, multiple programs can access the device simultaneously. It's the users responsibility to synchronize access between multiple users.

When done, the device must be returned by using [SCIReturnDevice\(\)](#)

Introduced in DIS release 5.5.0.

Parameters

<i>sd</i>	handle to an open SISI virtual device descriptor
<i>device</i>	handle to the new borrowed device descriptor
<i>deviceId</i>	identifier of the device to be borrowed
<i>flags</i>	see below
<i>error</i>	error information ...

Flags:

- `SCI_FLAG_EXCLUSIVE`
Request exclusive access to the device, preventing others from borrowing the device. Can be denied by system configuration.

Error codes:

- `SCI_ERR_OK`
Successful completion.
- `SCI_ERR_ILLEGAL_OPERATION`
Illegal operation
- `SCI_ERR_NOT_IMPLEMENTED`
SMARTIO not supported on this system or installed driver.
- `SCI_ERR_NOT_AVAILABLE`
Device is not currently available
- `SCI_ERR_NO_SUCH_SEGMENT`
No device with the given deviceId exists

SISI_API_EXPORT void `SCIReturnDevice (sci_device_t device, unsigned int flags, sci_error_t * error)`

NOTE: This function is not yet finalized and may change without notice!

Undo `SCIBorrowDevice` by releasing the borrowed device. Any resources (segments, interrupts, etc) associated with the device should be released before returning the device.

Introduced in DIS release 5.5.0.

Parameters

<i>device</i>	handle to the descriptor of the device that is being returned
<i>flags</i>	not used
<i>error</i>	error information ...

Error codes:

- `SCI_ERR_OK`
Successful completion.
- `SCI_ERR_ILLEGAL_OPERATION`
Illegal operation

```
SISCI_API_EXPORT void SCIDeconnectDeviceMemory ( sci_desc_t sd, sci_remote_segment_t * segment, unsigned int localAdapterNo, sci_device_t device, unsigned int segmentNo, unsigned int timeout, unsigned int flags, sci_error_t * error )
```

NOTE: This function is not yet finalized and may change without notice!

Connects an application to a device memory segment. There are three different types of device memory segments: BAR, PRIVATE and SHARED. The type is specified with the flag parameter. This function will connect to such segments for a borrowed device (See [SCIBorrowDevice\(\)](#)). These segments must be connected to with this function, but are otherwise identical other remote segments. The behaviour of this function is identical to [SCIDeconnectSegment\(\)](#) with the exception of the function parameters and any notes, flags or error codes documented here.

Notes:

- Device memory segments must be disconnected (see [SCIDisconnectSegment\(\)](#)) before the associated device is returned (see [SCIReturnDevice\(\)](#)). Otherwise the state of the segment is undefined after returning the device.
- A remote segment callback reason `SCI_CB_DISCONNECT` for a device segment signals that the segment has been disconnected and any mapping to it is invalid.

Introduced in DIS release 5.5.0.

Parameters

<i>sd</i>	handle to an open SISCI virtual device descriptor
<i>segment</i>	handle to the new connected segment descriptor
<i>localAdapterNo</i>	number of the local adapter through which the device segment should be accessed.
<i>device</i>	handle to the device that owns the BAR
<i>segmentNo</i>	number of the segment of the given type that should be connected to
<i>timeout</i>	time in milliseconds to wait for the connection to complete
<i>flags</i>	See flags below and flags for SCIDeconnectSegment()
<i>error</i>	See below and error codes for SCIDeconnectSegment()

Flags:

- `SCI_FLAG_PRIVATE`
The caller is given a private device memory segment only accessible by the device and the caller. The `segmentNo` parameter is ignored.
- `SCI_FLAG_SHARED`
Connect to a shared segment that different callers and the device can access.
- `SCI_FLAG_BAR`
Connects to the device's BAR segment given by the `segmentNo` parameter.

Error codes:

- `SCI_ERR_OK`
Successful completion

```
SISCI_API_EXPORT void SCIMapLocalSegmentForDevice ( sci_local_segment_t segment, unsigned int localAdapterNo, sci_device_t device, sci_ioaddr_t * remoteAddr, unsigned int userData, unsigned int flags, sci_error_t * error )
```

NOTE: This function is not yet finalized and may change without notice!

Sets up access to a local segment for device that is being borrowed from a remote node, allowing the device to DMA to the segment using `remoteAddr`.

Semantically similar to the device performing [SCIConnectSegment\(\)](#) and [SCIMapRemoteSegment\(\)](#). After successful completion, the device can access the segment using the address in `remoteAddr`.

Introduced in DIS release 5.5.0.

Parameters

<i>segment</i>	handle to the local segment descriptor
<i>localAdapterNo</i>	number of the local adapter through which the local segment can be mapped
<i>device</i>	handle to the descriptor of the device that the segment should be mapped for.
<i>remoteAddr</i>	resulting remote address that device can access segment through
<i>flags</i>	not used
<i>userData</i>	optional user defined value for identifying the map. Currently ignored.
<i>error</i>	error information

Error codes:

- `SCI_ERR_OK`
Successful completion.
- `SCI_ERR_ILLEGAL_OPERATION`
Illegal operation

`SISCI_API_EXPORT void SCIUnmapLocalSegmentForDevice (sci_local_segment_t segment, unsigned int localAdapterNo, sci_device_t device, unsigned int flags, sci_error_t * error)`

NOTE: This function is not yet finalized and may change without notice!

Undo [SCIMapLocalSegmentForDevice\(\)](#). Semantically similar to the device performing [SCIUnmapSegment\(\)](#) and [SCIDisconnectSegment\(\)](#).

Introduced in DIS release 5.5.0.

Parameters

<i>segment</i>	handle to the local segment descriptor
<i>localAdapterNo</i>	number of the local adapter through which the local segment has been mapped
<i>device</i>	handle to the descriptor of the device that the segment should be mapped for.
<i>flags</i>	not used
<i>error</i>	error information

Error codes:

- `SCI_ERR_OK`
Successful completion.
- `SCI_ERR_ILLEGAL_OPERATION`
Illegal operation

`SISCI_API_EXPORT void SCIMapRemoteSegmentForDevice (sci_remote_segment_t segment, sci_device_t device, sci_ioaddr_t * remoteAddr, unsigned int userData, unsigned int flags, sci_error_t * error)`

NOTE: This function is not yet finalized and may change without notice!

Sets up access to a remote segment for device that is being borrowed from a remote node, allowing the device to DMA to the segment using `remoteAddr`.

Semantically similar to the device performing [SCIConnectSegment\(\)](#) and [SCIMapRemoteSegment\(\)](#). After successful completion, the device can access the segment using the address in `remoteAddr`.

Introduced in DIS release 5.5.0.

Parameters

<i>segment</i>	handle to the remote segment descriptor
<i>device</i>	handle to the descriptor of the device that the segment should be mapped for.
<i>remoteAddr</i>	resulting remote address that device can access segment through
<i>flags</i>	not used
<i>userData</i>	optional user defined value for identifying the map. Currently ignored.
<i>error</i>	error information

Error codes:

- `SCI_ERR_OK`
Successful completion.
- `SCI_ERR_ILLEGAL_OPERATION`
Illegal operation

`SISCI_API_EXPORT void SCIUnmapRemoteSegmentForDevice (sci_remote_segment_t segment, sci_device_t device, unsigned int flags, sci_error_t * error)`

NOTE: This function is not yet finalized and may change without notice!

Undo [SCIMapRemoteSegmentForDevice\(\)](#). Semantically similar to the device performing [SCIUnmapSegment\(\)](#) and [SCIDisconnectSegment\(\)](#).

Introduced in DIS release 5.5.0.

Parameters

<i>segment</i>	handle to the local segment descriptor
<i>device</i>	handle to the descriptor of the device that the segment should be mapped for.
<i>flags</i>	not used
<i>error</i>	error information

Error codes:

- `SCI_ERR_OK`
Successful completion.
- `SCI_ERR_ILLEGAL_OPERATION`
Illegal operation

2.2 sisci_api_introduction.txt File Reference

2.3 sisci_error.h File Reference

Data types.

Enumerations

2.3.1 Detailed Description

Data types.

2.3.2 Enumeration Type Documentation

enum `sisci_error_t`

The type `sisci_error_t` represents the error code.

Enumerator

- `SCI_ERR_OK`** 0x00000000 - The error code represents successful operation
- `SCI_ERR_BUSY`** 0x40000900 - Some resources are busy
- `SCI_ERR_FLAG_NOT_IMPLEMENTED`** 0x40000901 - This flag option is not implemented
- `SCI_ERR_ILLEGAL_FLAG`** 0x40000902 - Illegal flag option
- `SCI_ERR_NOSPC`** 0x40000904 - Out of local resources
- `SCI_ERR_API_NOSPC`** 0x40000905 - Out of local API resources
- `SCI_ERR_HW_NOSPC`** 0x40000906 - Out of hardware resources
- `SCI_ERR_NOT_IMPLEMENTED`** 0x40000907 - The functionality is currently not supported
- `SCI_ERR_ILLEGAL_ADAPTERNO`** 0x40000908 - Illegal adapter number
- `SCI_ERR_NO_SUCH_ADAPTERNO`** 0x40000909 - The specified adapter number can not be found - Check the configuration
- `SCI_ERR_TIMEOUT`** 0x4000090A - The operation timed out
- `SCI_ERR_OUT_OF_RANGE`** 0x4000090B - The specified variable is not within the legal range
- `SCI_ERR_NO_SUCH_SEGMENT`** 0x4000090C - The specified segmentId is not found
- `SCI_ERR_NO_SUCH_INTNO`** 0x4000090C - The specified interrupt number is not found
- `SCI_ERR_ILLEGAL_NODEID`** 0x4000090D - Illegal nodeId - Check the configuration
- `SCI_ERR_CONNECTION_REFUSED`** 0x4000090E - Connection to remote node is refused
- `SCI_ERR_SEGMENT_NOT_CONNECTED`** 0x4000090F - No connection to the segment
- `SCI_ERR_SIZE_ALIGNMENT`** 0x40000910 - The specified size is not aligned
- `SCI_ERR_OFFSET_ALIGNMENT`** 0x40000911 - The specified offset is not aligned
- `SCI_ERR_ILLEGAL_PARAMETER`** 0x40000912 - Illegal function parameter
- `SCI_ERR_MAX_ENTRIES`** 0x40000913 - Maximum possible physical mapping is exceeded - Check the configuration
- `SCI_ERR_SEGMENT_NOT_PREPARED`** 0x40000914 - The segment is not prepared - Check documentation for [SCIPrepareSegment\(\)](#)
- `SCI_ERR_ILLEGAL_ADDRESS`** 0x40000915 - Illegal address
- `SCI_ERR_ILLEGAL_OPERATION`** 0x40000916 - Illegal operation
- `SCI_ERR_ILLEGAL_QUERY`** 0x40000917 - Illegal query operation - Check the documentation for function [SCIQuery\(\)](#)
- `SCI_ERR_SEGMENTID_USED`** 0x40000918 - This segmentId is already used - The segmentId must be unique
- `SCI_ERR_SYSTEM`** 0x40000919 - Could not get requested resources from the system (OS dependent)
- `SCI_ERR_CANCELLED`** 0x4000091A - The operation was cancelled
- `SCI_ERR_NOT_CONNECTED`** 0x4000091B - The host is not connected to the remote host
- `SCI_ERR_NOT_AVAILABLE`** 0x4000091C - The requested operation is not available
- `SCI_ERR_INCONSISTENT_VERSIONS`** 0x4000091D - Inconsistent driver versions on local host or remote host

SCI_ERR_OVERFLOW 0x4000091F - Out of local resources

SCI_ERR_NOT_INITIALIZED 0x40000920 - The host is not initialized - Check the configuration

SCI_ERR_ACCESS 0x40000921 - No local or remote access for the requested operation

SCI_ERR_NOT_SUPPORTED 0x40000922 - The request is not supported

SCI_ERR_DEPRECATED 0x40000923 - This function or functionality is deprecated

SCI_ERR_DMA_NOT_AVAILABLE 0x40000924 - The requested DMA mode, or the required DMA mode for the operation, is not available

SCI_ERR_DMA_DISABLED 0x40000925 - The requested DMA mode, or the required DMA mode for the operation, is disabled

SCI_ERR_NO_SUCH_NODEID 0x40000A00 - The specified nodeid could not be found

SCI_ERR_NODE_NOT_RESPONDING 0x40000A02 - The specified node does not respond to the request

SCI_ERR_NO_REMOTE_LINK_ACCESS 0x40000A04 - The remote link is not operational

SCI_ERR_NO_LINK_ACCESS 0x40000A05 - The local link is not operational

SCI_ERR_TRANSFER_FAILED 0x40000A06 - The transfer failed

SCI_ERR_IRQ_ILLEGAL 0x40000B02 - Illegal interrupt line

SCI_ERR_REMOTE_BUSY 0x40000B00 - The remote host is busy

SCI_ERR_LOCAL_BUSY 0x40000B03 - The local host is busy

SCI_ERR_ALL_BUSY 0x40000B04 - The system is busy

2.4 sisci_types.h File Reference

Data types.

Macros

- #define **DIS_DMA_MAX_VECLEN** 256
DMA queue vector interface for function *SCIStartDmaTransferVec()*.

Typedefs

- typedef struct sci_desc * **sci_desc_t**
A variable of type *sci_desc_t* represents an SISCI 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 adapter or in the system.
- 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_remote_data_interrupt * [sci_remote_data_interrupt_t](#)
A variable of type `sci_remote_data_interrupt_t` represents a data interrupt that can be triggered on a remote node.
- typedef struct sci_local_data_interrupt * [sci_local_data_interrupt_t](#)
A variable of type `sci_local_data_interrupt_t` represents an instance of a data interrupt that an application has made available to remote nodes.
- typedef struct sci_device * [sci_device_t](#)
A variable of type `sci_device_t` represents an instance of a device that an application has borrowed from a remote node.
- typedef [sci_callback_action_t](#)(* [sci_cb_local_segment_t](#)) (void *arg, [sci_local_segment_t](#) segment, [sci_local_segment_cb_reason_t](#) reason, unsigned int nodeld, 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_remote_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 [sci_callback_action_t](#)(* [sci_cb_interrupt_t](#)) (void *arg, [sci_local_interrupt_t](#) interrupt, [sci_error_t](#) status)
Interrupt callback.
- typedef [sci_callback_action_t](#)(* [sci_cb_data_interrupt_t](#)) (void *arg, [sci_local_data_interrupt_t](#) interrupt, void *data, unsigned int length, [sci_error_t](#) status)
Data Interrupt callback.

Enumerations

2.4.1 Detailed Description

Data types.

2.4.2 Macro Definition Documentation

```
#define DIS_DMA_MAX_VECLEN 256
```

DMA queue vector interface for function [SCIStartDmaTransferVec\(\)](#).

2.4.3 Typedef Documentation

```
typedef struct sci_desc* sci\_desc\_t
```

A variable of type `sci_desc_t` represents an SISI 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\(\)](#).

```
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.

It is initialized by calling the function [SCIConnectSegment\(\)](#)

```
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\(\)](#).

```
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\(\)](#).

```
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 adapter or in the system.

The descriptor is initialized when the chain is created by calling the function [SCICreateDMAQueue\(\)](#).

```
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\(\)](#).

```
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\(\)](#).

```
typedef struct sci_remote_data_interrupt* sci_remote_data_interrupt_t
```

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

It is initialized by calling the function [SCIConnectDataInterrupt\(\)](#).

```
typedef struct sci_local_data_interrupt* sci_local_data_interrupt_t
```

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

It is initialized when the interrupt is created by calling the function [SCICreateDataInterrupt\(\)](#).

```
typedef struct sci_device* sci_device_t
```

A variable of type `sci_device_t` represents an instance of a device that an application has borrowed from a remote node.

It is initialized by calling the function [SCIBorrowDevice\(\)](#).

```
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

<i>arg</i>	user-defined argument passed to the callback function.
------------	--

Parameters

<i>segment</i>	handle to the local segment descriptor affected by the asynchronous event.
<i>reason</i>	reason why the callback function has been invoked.
<i>nodeId</i>	identifier of the remote node that has provoked, directly or indirectly, the asynchronous event.
<i>localAdapterNo</i>	number of the local adapter that received the asynchronous event.

Returns

SCI_CALLBACK_CANCEL or SCI_CALLBACK_CONTINUE.

```
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

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

Returns

SCI_CALLBACK_CANCEL or SCI_CALLBACK_CONTINUE.

```
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 for a DMA operation and will be invoked when the transfer has completed, either successfully or with an error.

Parameters

<i>arg</i>	user-defined argument passed to the callback function.
<i>queue</i>	handle to the DMA queue descriptor.
<i>status</i>	status information.

Returns

SCI_CALLBACK_CANCEL or SCI_CALLBACK_CONTINUE.

```
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

<i>arg</i>	user-defined argument passed to the callback function.
<i>interrupt</i>	handle to the triggered interrupt descriptor.
<i>status</i>	status information

Returns

SCI_CALLBACK_CANCEL or SCI_CALLBACK_CONTINUE.

```
typedef sci_callback_action_t(* sci_cb_data_interrupt_t)(void *arg, sci_local_data_interrupt_t interrupt, void *data,
unsigned int length, sci_error_t status)
```

Data Interrupt callback.

A function of type `sci_cb_data_interrupt_t` can be specified when a data interrupt is created with [SCICreateDataInterrupt\(\)](#) and it will be invoked asynchronously when the data interrupt will be triggered from a remote node.

Parameters

<i>arg</i>	user-defined argument passed to the callback function.
<i>interrupt</i>	handle to the triggered data interrupt descriptor.
<i>data</i>	pointer to the interrupt message data
<i>length</i>	length of the interrupt message
<i>status</i>	status information

Returns

SCI_CALLBACK_CANCEL or SCI_CALLBACK_CONTINUE.

2.4.4 Enumeration Type Documentation

```
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 Indicates that the PCI Express link is no longer operational. Typical problems: cable unplugged, remote node or switch power cycled etc. Normally followed by a [SCI_CB_OPERATIONAL](#) or [SCI_CB_LOST](#).

SCI_CB_OPERATIONAL Indicates that the PCI Express link is operational again.

SCI_CB_LOST Indicates the session to a remote node has been lost - normally caused by a remote node reboot or reloading of the driver. All mappings, connections to a remote node should be removed and a full reconnect/remmap of remote resources should be attempted.

SCI_CB_FATAL Indicates an uncorrectable error on the adapter PCIe slot (edge connector).

enum `sci_dma_queue_state_t`

DMA queue status.

Precise description needed.

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`.

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

DIS_DMA_MAX_VECLEN
sisci_types.h, [59](#)

SCI_CALLBACK_CANCEL
sisci_types.h, [63](#)

SCI_CALLBACK_CONTINUE
sisci_types.h, [63](#)

SCI_CB_CONNECT
sisci_types.h, [62](#)

SCI_CB_DISCONNECT
sisci_types.h, [62](#)

SCI_CB_FATAL
sisci_types.h, [62](#)

SCI_CB_LOST
sisci_types.h, [62](#)

SCI_CB_NOT_OPERATIONAL
sisci_types.h, [62](#)

SCI_CB_OPERATIONAL
sisci_types.h, [62](#)

SCI_ERR_ACCESS
sisci_error.h, [58](#)

SCI_ERR_ALL_BUSY
sisci_error.h, [58](#)

SCI_ERR_API_NOSPC
sisci_error.h, [57](#)

SCI_ERR_BUSY
sisci_error.h, [57](#)

SCI_ERR_CANCELLED
sisci_error.h, [57](#)

SCI_ERR_CONNECTION_REFUSED
sisci_error.h, [57](#)

SCI_ERR_DEPRECATED
sisci_error.h, [58](#)

SCI_ERR_DMA_DISABLED
sisci_error.h, [58](#)

SCI_ERR_DMA_NOT_AVAILABLE
sisci_error.h, [58](#)

SCI_ERR_FLAG_NOT_IMPLEMENTED
sisci_error.h, [57](#)

SCI_ERR_HW_NOSPC
sisci_error.h, [57](#)

SCI_ERR_ILLEGAL_ADAPTERNO
sisci_error.h, [57](#)

SCI_ERR_ILLEGAL_ADDRESS
sisci_error.h, [57](#)

SCI_ERR_ILLEGAL_FLAG
sisci_error.h, [57](#)

SCI_ERR_ILLEGAL_NODEID
sisci_error.h, [57](#)

SCI_ERR_ILLEGAL_OPERATION
sisci_error.h, [57](#)

SCI_ERR_ILLEGAL_PARAMETER
sisci_error.h, [57](#)

SCI_ERR_ILLEGAL_QUERY
sisci_error.h, [57](#)

SCI_ERR_INCONSISTENT_VERSIONS
sisci_error.h, [57](#)

SCI_ERR_IRQ_ILLEGAL
sisci_error.h, [58](#)

SCI_ERR_LOCAL_BUSY
sisci_error.h, [58](#)

SCI_ERR_MAX_ENTRIES
sisci_error.h, [57](#)

SCI_ERR_NO_LINK_ACCESS
sisci_error.h, [58](#)

SCI_ERR_NO_REMOTE_LINK_ACCESS
sisci_error.h, [58](#)

SCI_ERR_NO_SUCH_ADAPTERNO
sisci_error.h, [57](#)

SCI_ERR_NO_SUCH_INTNO
sisci_error.h, [57](#)

SCI_ERR_NO_SUCH_NODEID
sisci_error.h, [58](#)

SCI_ERR_NO_SUCH_SEGMENT
sisci_error.h, [57](#)

SCI_ERR_NODE_NOT_RESPONDING
sisci_error.h, [58](#)

SCI_ERR_NOSPC
sisci_error.h, [57](#)

SCI_ERR_NOT_AVAILABLE
sisci_error.h, [57](#)

SCI_ERR_NOT_CONNECTED
sisci_error.h, [57](#)

SCI_ERR_NOT_IMPLEMENTED
sisci_error.h, [57](#)

SCI_ERR_NOT_INITIALIZED
sisci_error.h, [58](#)

SCI_ERR_NOT_SUPPORTED
sisci_error.h, [58](#)

SCI_ERR_OFFSET_ALIGNMENT
sisci_error.h, [57](#)

SCI_ERR_OUT_OF_RANGE
sisci_error.h, [57](#)

SCI_ERR_OVERFLOW
sisci_error.h, [57](#)

SCI_ERR_OK
sisci_error.h, [57](#)

SCI_ERR_REMOTE_BUSY
sisci_error.h, [58](#)

SCI_ERR_SEGMENT_NOT_CONNECTED
sisci_error.h, [57](#)

SCI_ERR_SEGMENT_NOT_PREPARED
sisci_error.h, [57](#)

SCI_ERR_SEGMENTID_USED
sisci_error.h, [57](#)

SCI_ERR_SIZE_ALIGNMENT
sisci_error.h, [57](#)

SCI_ERR_SYSTEM
sisci_error.h, [57](#)

SCI_ERR_TIMEOUT

sisci_error.h, [57](#)
 SCI_ERR_TRANSFER_FAILED
 sisci_error.h, [58](#)
 SCI_SEQ_NOT_RETRIABLE
 sisci_types.h, [63](#)
 SCI_SEQ_OK
 sisci_types.h, [63](#)
 SCI_SEQ_PENDING
 sisci_types.h, [63](#)
 SCI_SEQ_RETRIABLE
 sisci_types.h, [63](#)
 SCIAbortDMAQueue
 sisci_api.h, [34](#)
 SCIAttachLocalSegment
 sisci_api.h, [44](#)
 SCIAttachPhysicalMemory
 sisci_api.h, [29](#)
 SCIBorrowDevice
 sisci_api.h, [52](#)
 SCICacheSync
 sisci_api.h, [49](#)
 SCICheckSequence
 sisci_api.h, [27](#)
 SCIClose
 sisci_api.h, [17](#)
 SCIConnectDataInterrupt
 sisci_api.h, [40](#)
 SCIConnectDeviceMemory
 sisci_api.h, [53](#)
 SCIConnectInterrupt
 sisci_api.h, [37](#)
 SCIConnectSegment
 sisci_api.h, [17](#)
 SCICreateDMAQueue
 sisci_api.h, [33](#)
 SCICreateDataInterrupt
 sisci_api.h, [39](#)
 SCICreateInterrupt
 sisci_api.h, [35](#)
 SCICreateMapSequence
 sisci_api.h, [26](#)
 SCICreateSegment
 sisci_api.h, [22](#)
 SCIDMAQueueState
 sisci_api.h, [34](#)
 SCIDisconnectDataInterrupt
 sisci_api.h, [41](#)
 SCIDisconnectInterrupt
 sisci_api.h, [38](#)
 SCIDisconnectSegment
 sisci_api.h, [18](#)
 SCIFlush
 sisci_api.h, [45](#)
 SCIGetErrorString
 sisci_api.h, [33](#)
 SCIGetLocalNodeId
 sisci_api.h, [32](#)
 SCIGetNodeIdByAdapterName
 sisci_api.h, [32](#)
 SCIGetNodeInfoByAdapterName
 sisci_api.h, [32](#)
 SCIGetRemoteSegmentSize
 sisci_api.h, [19](#)
 SCIInitialize
 sisci_api.h, [16](#)
 SCIMapLocalSegment
 sisci_api.h, [21](#)
 SCIMapLocalSegmentForDevice
 sisci_api.h, [54](#)
 SCIMapRemoteSegment
 sisci_api.h, [20](#)
 SCIMapRemoteSegmentForDevice
 sisci_api.h, [55](#)
 SCIMemCpy
 sisci_api.h, [42](#)
 SCIMemWrite
 sisci_api.h, [42](#)
 SCIOpen
 sisci_api.h, [16](#)
 SCIPrepareSegment
 sisci_api.h, [24](#)
 SCIProbeNode
 sisci_api.h, [28](#)
 SCIQuery
 sisci_api.h, [30](#)
 SCIRegisterPCIeRequester
 sisci_api.h, [50](#)
 SCIRegisterSegmentMemory
 sisci_api.h, [43](#)
 SCIRemoveDMAQueue
 sisci_api.h, [34](#)
 SCIRemoveDataInterrupt
 sisci_api.h, [39](#)
 SCIRemoveInterrupt
 sisci_api.h, [36](#)
 SCIRemoveSegment
 sisci_api.h, [24](#)
 SCIRemoveSequence
 sisci_api.h, [26](#)
 SCIReturnDevice
 sisci_api.h, [53](#)
 SCISetSegmentAvailable
 sisci_api.h, [25](#)
 SCISetSegmentUnavailable
 sisci_api.h, [26](#)
 SCIShareSegment
 sisci_api.h, [45](#)
 SCIStartDmaTransfer
 sisci_api.h, [45](#)
 SCIStartDmaTransferMem
 sisci_api.h, [47](#)
 SCIStartDmaTransferVec
 sisci_api.h, [48](#)
 SCIStartSequence
 sisci_api.h, [27](#)
 SCIStoreBarrier

- sisci_api.h, 28
- SCITerminate
 - sisci_api.h, 16
- SCITriggerDataInterrupt
 - sisci_api.h, 41
- SCITriggerInterrupt
 - sisci_api.h, 38
- SCIUnmapLocalSegmentForDevice
 - sisci_api.h, 55
- SCIUnmapRemoteSegmentForDevice
 - sisci_api.h, 56
- SCIUnmapSegment
 - sisci_api.h, 22
- SCIUnregisterPCIeRequester
 - sisci_api.h, 52
- SCIWaitForDMAQueue
 - sisci_api.h, 35
- SCIWaitForDataInterrupt
 - sisci_api.h, 40
- SCIWaitForInterrupt
 - sisci_api.h, 37
- SCIWaitForLocalSegmentEvent
 - sisci_api.h, 23
- SCIWaitForRemoteSegmentEvent
 - sisci_api.h, 19
- sci_callback_action_t
 - sisci_types.h, 63
- sci_cb_data_interrupt_t
 - sisci_types.h, 62
- sci_cb_dma_t
 - sisci_types.h, 61
- sci_cb_interrupt_t
 - sisci_types.h, 61
- sci_cb_local_segment_t
 - sisci_types.h, 60
- sci_cb_remote_segment_t
 - sisci_types.h, 61
- sci_desc_t
 - sisci_types.h, 59
- sci_device_t
 - sisci_types.h, 60
- sci_dma_queue_state_t
 - sisci_types.h, 62
- sci_dma_queue_t
 - sisci_types.h, 60
- sci_error_t
 - sisci_error.h, 57
- sci_local_data_interrupt_t
 - sisci_types.h, 60
- sci_local_interrupt_t
 - sisci_types.h, 60
- sci_local_segment_t
 - sisci_types.h, 59
- sci_map_t
 - sisci_types.h, 59
- sci_remote_data_interrupt_t
 - sisci_types.h, 60
- sci_remote_interrupt_t
 - sisci_types.h, 60
- sisci_types.h, 60
- sci_remote_segment_t
 - sisci_types.h, 59
- sci_segment_cb_reason_t
 - sisci_types.h, 62
- sci_sequence_status_t
 - sisci_types.h, 63
- sci_sequence_t
 - sisci_types.h, 60
- sisci_api.h, 11
 - SCIAbortDMAQueue, 34
 - SCIAttachLocalSegment, 44
 - SCIAttachPhysicalMemory, 29
 - SCIBorrowDevice, 52
 - SCICacheSync, 49
 - SCICheckSequence, 27
 - SCIClose, 17
 - SCIConnectDataInterrupt, 40
 - SCIConnectDeviceMemory, 53
 - SCIConnectInterrupt, 37
 - SCIConnectSegment, 17
 - SCICreateDMAQueue, 33
 - SCICreateDataInterrupt, 39
 - SCICreateInterrupt, 35
 - SCICreateMapSequence, 26
 - SCICreateSegment, 22
 - SCIDMAQueueState, 34
 - SCIDisconnectDataInterrupt, 41
 - SCIDisconnectInterrupt, 38
 - SCIDisconnectSegment, 18
 - SCIFlush, 45
 - SCIGetErrorString, 33
 - SCIGetLocalNodeId, 32
 - SCIGetNodeIdByAdapterName, 32
 - SCIGetNodeInfoByAdapterName, 32
 - SCIGetRemoteSegmentSize, 19
 - SCIInitialize, 16
 - SCIMapLocalSegment, 21
 - SCIMapLocalSegmentForDevice, 54
 - SCIMapRemoteSegment, 20
 - SCIMapRemoteSegmentForDevice, 55
 - SCIMemCpy, 42
 - SCIMemWrite, 42
 - SCIOpen, 16
 - SCIPrepareSegment, 24
 - SCIProbeNode, 28
 - SCIQuery, 30
 - SCIRegisterPCIeRequester, 50
 - SCIRegisterSegmentMemory, 43
 - SCIRemoveDMAQueue, 34
 - SCIRemoveDataInterrupt, 39
 - SCIRemoveInterrupt, 36
 - SCIRemoveSegment, 24
 - SCIRemoveSequence, 26
 - SCIReturnDevice, 53
 - SCISetSegmentAvailable, 25
 - SCISetSegmentUnavailable, 26
 - SCIShareSegment, 45

- SCIStartDmaTransfer, [45](#)
- SCIStartDmaTransferMem, [47](#)
- SCIStartDmaTransferVec, [48](#)
- SCIStartSequence, [27](#)
- SCIStoreBarrier, [28](#)
- SCITerminate, [16](#)
- SCITriggerDataInterrupt, [41](#)
- SCITriggerInterrupt, [38](#)
- SCIUnmapLocalSegmentForDevice, [55](#)
- SCIUnmapRemoteSegmentForDevice, [56](#)
- SCIUnmapSegment, [22](#)
- SCIUnregisterPCIeRequester, [52](#)
- SCIWaitForDMAQueue, [35](#)
- SCIWaitForDataInterrupt, [40](#)
- SCIWaitForInterrupt, [37](#)
- SCIWaitForLocalSegmentEvent, [23](#)
- SCIWaitForRemoteSegmentEvent, [19](#)
- sisci_api_introduction.txt, [56](#)
- sisci_error.h, [56](#)
 - SCI_ERR_ACCESS, [58](#)
 - SCI_ERR_ALL_BUSY, [58](#)
 - SCI_ERR_API_NOSPC, [57](#)
 - SCI_ERR_BUSY, [57](#)
 - SCI_ERR_CANCELLED, [57](#)
 - SCI_ERR_CONNECTION_REFUSED, [57](#)
 - SCI_ERR_DEPRECATED, [58](#)
 - SCI_ERR_DMA_DISABLED, [58](#)
 - SCI_ERR_DMA_NOT_AVAILABLE, [58](#)
 - SCI_ERR_FLAG_NOT_IMPLEMENTED, [57](#)
 - SCI_ERR_HW_NOSPC, [57](#)
 - SCI_ERR_ILLEGAL_ADAPTERNO, [57](#)
 - SCI_ERR_ILLEGAL_ADDRESS, [57](#)
 - SCI_ERR_ILLEGAL_FLAG, [57](#)
 - SCI_ERR_ILLEGAL_NODEID, [57](#)
 - SCI_ERR_ILLEGAL_OPERATION, [57](#)
 - SCI_ERR_ILLEGAL_PARAMETER, [57](#)
 - SCI_ERR_ILLEGAL_QUERY, [57](#)
 - SCI_ERR_INCONSISTENT_VERSIONS, [57](#)
 - SCI_ERR_IRQL_ILLEGAL, [58](#)
 - SCI_ERR_LOCAL_BUSY, [58](#)
 - SCI_ERR_MAX_ENTRIES, [57](#)
 - SCI_ERR_NO_LINK_ACCESS, [58](#)
 - SCI_ERR_NO_REMOTE_LINK_ACCESS, [58](#)
 - SCI_ERR_NO_SUCH_ADAPTERNO, [57](#)
 - SCI_ERR_NO_SUCH_INTNO, [57](#)
 - SCI_ERR_NO_SUCH_NODEID, [58](#)
 - SCI_ERR_NO_SUCH_SEGMENT, [57](#)
 - SCI_ERR_NODE_NOT_RESPONDING, [58](#)
 - SCI_ERR_NOSPC, [57](#)
 - SCI_ERR_NOT_AVAILABLE, [57](#)
 - SCI_ERR_NOT_CONNECTED, [57](#)
 - SCI_ERR_NOT_IMPLEMENTED, [57](#)
 - SCI_ERR_NOT_INITIALIZED, [58](#)
 - SCI_ERR_NOT_SUPPORTED, [58](#)
 - SCI_ERR_OFFSET_ALIGNMENT, [57](#)
 - SCI_ERR_OUT_OF_RANGE, [57](#)
 - SCI_ERR_OVERFLOW, [57](#)
 - SCI_ERR_OK, [57](#)
 - SCI_ERR_REMOTE_BUSY, [58](#)
 - SCI_ERR_SEGMENT_NOT_CONNECTED, [57](#)
 - SCI_ERR_SEGMENT_NOT_PREPARED, [57](#)
 - SCI_ERR_SEGMENTID_USED, [57](#)
 - SCI_ERR_SIZE_ALIGNMENT, [57](#)
 - SCI_ERR_SYSTEM, [57](#)
 - SCI_ERR_TIMEOUT, [57](#)
 - SCI_ERR_TRANSFER_FAILED, [58](#)
 - sci_error_t, [57](#)
- sisci_types.h, [58](#)
 - DIS_DMA_MAX_VECLEN, [59](#)
 - SCI_CALLBACK_CANCEL, [63](#)
 - SCI_CALLBACK_CONTINUE, [63](#)
 - SCI_CB_CONNECT, [62](#)
 - SCI_CB_DISCONNECT, [62](#)
 - SCI_CB_FATAL, [62](#)
 - SCI_CB_LOST, [62](#)
 - SCI_CB_NOT_OPERATIONAL, [62](#)
 - SCI_CB_OPERATIONAL, [62](#)
 - SCI_SEQ_NOT_RETRIABLE, [63](#)
 - SCI_SEQ_OK, [63](#)
 - SCI_SEQ_PENDING, [63](#)
 - SCI_SEQ_RETRIABLE, [63](#)
 - sci_callback_action_t, [63](#)
 - sci_cb_data_interrupt_t, [62](#)
 - sci_cb_dma_t, [61](#)
 - sci_cb_interrupt_t, [61](#)
 - sci_cb_local_segment_t, [60](#)
 - sci_cb_remote_segment_t, [61](#)
 - sci_desc_t, [59](#)
 - sci_device_t, [60](#)
 - sci_dma_queue_state_t, [62](#)
 - sci_dma_queue_t, [60](#)
 - sci_local_data_interrupt_t, [60](#)
 - sci_local_interrupt_t, [60](#)
 - sci_local_segment_t, [59](#)
 - sci_map_t, [59](#)
 - sci_remote_data_interrupt_t, [60](#)
 - sci_remote_interrupt_t, [60](#)
 - sci_remote_segment_t, [59](#)
 - sci_segment_cb_reason_t, [62](#)
 - sci_sequence_status_t, [63](#)
 - sci_sequence_t, [60](#)