Uniform Driver Interface

UDI SCSI Driver Specification Version 1.0

http://www.project-UDI.org/specs.html

UDI SCSI Driver Specification

Abstract

The UDI SCSI Driver Specification defines the required interfaces and semantics for UDI environments that support SCSI Drivers. This is an optional extension to the UDI Core Specification, which is defined in a separate book. See the Document Organization chapter in the UDI Core Specification for a description of the other books in the UDI Specification, as well as references to additional tutorial materials. The intended audience for this book includes driver writers, environment implementors, and metalanguage implementors.

Status of This Document

This document has been reviewed by Project UDI Members and other interested parties and has been endorsed as a Final Specification. It is a stable document and may be used as reference material or cited as a normative reference from another document. This version of the specification is intended to be ready for use in product design and implementation. Every attempt has been made to ensure a consistent and implementable specification. Implementations should ensure compliance with this version.

Copyright Notice

Acknowledgements

The authors would like to thank everyone who reviewed working drafts of the specification and submitted suggestions and corrections.

The authors would especially like to thank their significant others for putting up with the many hours of overtime put into the development of this specification over long periods.

Thanks to the following folks who contributed significant amounts of time, ideas, or authoring in support of the development of this specification:

> Richard Arndt (IBM), Bob Barned (Lockheed Martin), Mark Bradley (Adaptec), Darren Busing (Adaptec), Thomas Clark (Sun), Mark Evenson (HP), Barry Feild (SCO), Kurt Gollhardt (SCO), Jim Heidbrink (Lockheed Martin), Chris Ilnicki (HP), Bret Indrelee (SBS Technologies), David Kahn (Sun), John Lee (Sun), Robert Lipe (SCO), Lynne McCue (IBM), Mark Parenti (DEC), James Partridge (IBM), Kevin Quick (Interphase), Larry Robinson (Adaptec), James Smart (Compaq), Pete Smoot (HP), David Stoft (HP), Wolfgang Thaler (Sun), Linda Wang (Sun), Mike Wenzel (HP).

In addition, thanks to everyone who worked on the prototype implementations that helped us validate the specification, including:

> Betty Dall (HP), Don Dugger (Intel), Bob Goudreau (Data General), James Hall (SCO), Matt Kaufman (SCO), Mike Lyons (IBM), Alex Malone (DEC), Guru Pangal (Starcom), Scott Popp (SCO), Richard Schall (Intel), Sam Shteingart (HP), Ajmer Singh (SCO), Ramaswamy Tummala (Starcom).

Finally, thanks to David Roberts (Certek Software Designs) for designing the Project UDI logo.

Table of Contents

List of Reference Pages by Chapter

Chapter 3 SCSI Metalanguage Interfaces

Alphabetical List of Symbols

SCSI Driver Introduction 1

1.1 Introduction

A UDI SCSI Driver is a conformant UDI driver which uses the SCSI Metalanguage, either in the SCSI Peripheral Driver (PD) role or the SCSI HBA Driver (HD) role. This document, the UDI SCSI Driver Specification, defines interfaces for communicating between SCSI PDs and HDs.

1.2 Scope

The UDI SCSI Driver Specification defines the complete set of interfaces available for communication between SCSI Peripheral Drivers (PDs) and SCSI HBA Drivers (HDs). These interfaces cover drivers which control SCSI-1, SCSI-2, and SCSI-3 compliant devices, and can be used in any configuration where the SCSI protocol is used or desired – including encapsulated SCSI across serial links such as Fibre Channel.

The UDI SCSI Driver Specification also defines the responsibilities and requirements on PDs and HDs, and specifies SCSI-specific bindings to the UDI Core Specification.

1.3 Normative References

The UDI SCSI Driver Specification references the following non-UDI standards, listed below. These standards contain provisions that, through reference in this document, constitute provisions of the UDI SCSI Driver Specification.

1. ANSI X3.131-1986 (SCSI-1).
2. ANSI X3.131-1994 (SCSI-2).
3. ANSI X3.270-1996 (SCSI-3 Architectural Model).
4. SCSI-3 Architectural Model - 2 (SAM-2), Revision 11, 16 July 1999

The UDI SCSI Driver Specification also references and depends upon the UDI Core Specification.

1.4 Conformance

A conforming UDI SCSI PD implementation attaches to the child end of a SCSI metalanguage channel, and uses only the interfaces specified in this document for communication across that channel to its parent HD. Similarly, a SCSI HD attaches to the parent end of a SCSI Metalanguage channel, and uses only the interfaces specified in this document for communication across that channel to its child PD.

Furthermore, conforming SCSI PDs and HDs follow all of the rules and requirements specified in this document, as well as any other relevant UDI specifications, including the use of header files and versioning, and the responsibilities shared or owned by the PD versus the HD.

1.5 Terminology

This section defines common terminology and acronyms with specific usage or meaning in the UDI SCSI Driver Specification.

adapter See definition in the *"Terminology"* chapter of the UDI Core Specification.

Bus Number A value representing a SCSI bus/link interface on an HBA. Typically, this value is zero unless an HBA is multi-ported and has interdependent SCSI interfaces (e.g. only a single hardware entity is enumerated that manages the multiple SCSI bus/link interfaces). This is the first level of SCSI I/O addressing for the SCSI Metalanguage. Refer to Section 2.3, "SCSI I/O Addressing" for further details.

HBA Host Bus Adapter. Refers to the hardware or software entity that the HBA Driver (see **HD**) controls. For parallel SCSI HDs this typically refers to the hardware adapter between the host and the parallel SCSI bus.

HD SCSI HBA Driver. UDI driver that receives and processes SCSI Metalangauge requests from a SCSI Peripheral Driver; i.e., the driver which provides the parent role on the SCSI Metalanguage channel.

LUN Logical Unit Number. An externally addressable entity within a target (see **target**) that implements the functions of a device module (e.g., part of a node on a SCSI bus). The LUN is the third level of SCSI I/O addressing for the SCSI Metalanguage. The LUN addresses one of many peripheral devices (like a SCSI disk) on the target interface which connects to the SCSI interconnect. Refer to Section 2.3, "SCSI I/O Addressing" for further details.

PD SCSI Peripheral Driver. UDI driver that controls a specific type of peripheral device or class of devices attached to a SCSI interconnect; i.e., the driver which provides the child role on a SCSI Metalanguage channel.

SAM SCSI-3 Architectural Model.

SCSI Small Computer System Interface. SCSI refers to the ANSI standards SCSI-1 (X3.131-1986), SCSI-2 (X3.131-1994), and the set of working drafts which comprise the in-progress SCSI-3 standard. SCSI defines a protocol for interconnecting computers and peripheral devices. A primary objective of SCSI has been to provide host computers with device independence within a set of defined device models. Thus, SCSI defines a set of device models for various classes of devices, each with their own device model specific command set.

SCSI Interconnect A SCSI interconnect is either the SCSI parallel bus defined in SCSI-1 or SCSI-2, or one of the supported I/O interconnects defined for SCSI-3 (parallel bus, Fibre Channel, SSA, 1394, etc.).

tag The fourth level of SCSI I/O addressing for the SCSI Metalanguage, The "tag" designates a specific I/O request. The use of tags is typical in "tagged command queueing" on SCSI-2 and "tagged tasks" on SCSI-3. Refer to Section 2.3, "SCSI I/O Addressing" for further details.

Target ID (Target) The second level of SCSI I/O addressing for the SCSI Metalanguage. The Target ID corresponds to the hardware entity (“target”) that directly connects to the SCSI interconnect. The target typically is an access interface behind which multiple SCSI devices are presented via the third level of addressing - the LUN. Although historically set to the physical address of the target on the SCSI interconnect, the Target ID value is considered a logical handle exported by the HD (and potentially different from other HD’s on the same interconnect) . Refer to Section 2.3, “SCSI I/O Addressing” for further details.

SCSI Driver Requirements & Bindings 2

2.1 General Requirements

2.1.1 Versioning

All functions and structures defined in the UDI SCSI Driver Specification are part of the "`udi_scsi`" interface, currently at version "`0x100`". A driver that conforms to and uses the UDI SCSI Driver Specification, Version 1.0, must include the following declaration in its `udiprops.txt` file (see Chapter 31, *"Static Driver Properties"*, of the UDI Core Specification):

```
requires udi_scsi 0x100
```

In each UDI SCSI driver source file, before including any UDI header files, the driver must define the preprocessor symbol, `UDI_SCSI_VERSION`, to indicate the version of the UDI SCSI Driver Specification to which it conforms, which must be the same as the interface version defined above:

```
#define UDI_SCSI_VERSION   0x100
```

A portable implementation of the SCSI Metalanguage Library must include a corresponding "provides" declaration in its `udiprops.txt` file and must also define `UDI_SCSI_VERSION`.

As defined in Section 31.4.6, "Requires Declaration," on page 31-6 of the UDI Core Specification, the two least-significant hexadecimal digits of the interface version represent the minor number; the rest of the hex digits represent the major number. Versions that have the same "major version number" as an earlier version shall be backward compatible with that earlier version (i.e. a strict superset).

2.1.2 Header Files

Each UDI SCSI driver source file must include the file "`udi_scsi.h`" after it includes "`udi.h`", as follows:

```
#include <udi.h>
#include <udi_scsi.h>
```

The "`udi_scsi.h`" header file contains function prototypes and other definitions needed to use the UDI SCSI interfaces.

2.2 SCSI Metalanguage Model

The SCSI Metalanguage is designed to allow for exactly one HD instance attached to a given HBA, but multiple instances of a PD may be attached to a given LUN. A PD is either a multi-lun PD (see the definition of the "`scsi_multi_lun`" attribute in Section 2.5.3.1, "Enumeration Attributes" below), or a single-lun PD which controls a single LUN. PDs are generally single-lun in nature, but special PDs can be configured which control multiple LUNs.

While there is a 1-to-1 correspondence between an HD and an HBA (as defined in Section 1.5, "Terminology"), there isn't necessarily a 1-to-1 correspondence between HDs and SCSI buses or interconnects. For example, a non-independent multi-port HBA (an HBA with multiple SCSI buses that can't be controlled independently) will have a single instance of an HD which controls multiple SCSI buses/links. This is the reason for the "`scsi_bus`" enumeration attribute.

For purposes of exclusive access (see the exclusive bind flags in the `udi_scsi_bind_cb_t` below) and SCSI event notification, unless stated otherwise in the SCSI interfaces, a multi-lun PD is treated as if there is a single-lun PD attached to each possible LUN. Thus, a multi-lun PD which is exclusively bound prohibits all other PD bindings to the HD, effectively giving the multi-lun PD exclusive access to the HD. A multi-lun PD which is non-exclusively bound prevents any other PD from binding exclusively (although another PD could in this case still do a "temporary exclusive bind"). If a SCSI event occurs which affects the PDs on a given LUN, any multi-lun PDs which have the event enabled will be notified along with the single-lun PDs attached to the LUN.

2.3 SCSI I/O Addressing

The SCSI Metalanguage recognizes 4 levels of SCSI I/O addressing: Bus Number, Target ID, LUN, and Tag. The Tag is not strictly an addressing component (it is a request identifier within a LUN from a given initiator) and is therefore not described in detail in this section. Tag use and assignment is typically HD and/or SCSI interconnect dependent. However, the PD is allowed to specify Tag types (Head of Queue, Ordered, etc) for each SCSI I/O issued to the HD.

The SCSI Metalanguage is designed to set up bindings between the PDs and HDs such that I/O request packets passed between the PD and HD need not specify Bus Number, Target ID, or LUN values (with the single exception of the multi-lun PD). The addressing values are a property of the enumerated PD instance and are inherent in the channel used for communication between the PD and the HD. For the multi-lun PD, the Bus Number is inherent in the channel used to communicate with the HD, and the Target ID and LUN values are encoded into the SCSI I/O control block.

2.3.1 Bus Number

The Bus Number represents a SCSI bus/link interface on an HBA. Typical HBA's are single ported (containing a single SCSI interface) or if multi-ported, enumerate their hardware such that separate and independent hardware instances are created for each SCSI interface. In these cases, the Bus Number is zero. For the multi-ported HBA's that contain interdependent SCSI interfaces (e.g. only one HD instance is enumerated for multiple SCSI interfaces), the Bus Number represents which of the SCSI interfaces on the HBA the I/O should be issued on. The Bus Number values are relative to the HBA's HD. Multiple HD's whose hardware is connected to the same SCSI interconnect, may export different Bus Numbers for the SCSI interconnect.

The Bus Number addressing component is specified by the HD as an enumeration attribute when it enumerates the PD instances. SCSI I/O requests never specify a Bus Number, as it is inherent in the channel used to make the I/O request.

2.3.2 Target ID

The Target ID represents a hardware entity attached to the SCSI interconnect. The value is assigned by the HD and, depending on the HD implementation, may or may not be persistent across driver restarts. As the Target ID value is HD dependent, multiple HD's attached to the same SCSI interconnect may export different Target ID values for the same hardware entity.

Although the Target ID value has historically been set to the physical value of the hardware entity on the interconnect, the SCSI Metalanguage recognizes that there are inherent problems in doing so, especially with new SCSI-3 interconnects where physical addresses are temporal, much larger, sparsely populated, and may be changed while I/O is in flight. Historical mappings also suffer from additional issues regarding delays and interconnect utilization when I/O is sent to Target IDs that are non-existent.

In general, the SCSI Metalanguage expects an HD to support a maximum number (N) of Target IDs, whose values range from 0 to N-1. During enumeration, an HD may enumerate devices that it detects are present at some subset of the Target IDs. All N Target IDs can be addressed by the multi-lun PD (even if not enumerated), with the value of N being passed to the multi-lun PD during the binding sequence (see ***max_targets*** in the `udi_scsi_bind_ack` operation).

The Target ID addressing component is specified by the HD as an enumeration attribute on all non-multi-lun PD instances that it enumerates. On a per-request basis, the Target ID component is specified only in SCSI I/O requests made by a multi-lun PD. I/O requests from all other PDs have the Target ID inherent in the channel that the I/O request is issued on.

2.3.3 LUN - Logical Unit Number

As per the SCSI Architecture Model - 2 (SAM-2) specification: the LUN represents a target-resident entity which implements a device model and executes SCSI commands sent by an application client. The LUN value is an encoded 64 bit (8 byte) identifier for a SCSI logical unit. A detailed definition of a logical unit number may be found in the SAM-2 specification.

Logical Unit values are typically obtained from the Target ID (e.g. the physical device) via the use of the SCSI-3 REPORT LUNS SCSI command issued to LUN value zero. However, some hardware and interconnects do not fully support the SCSI-3 concept of an 8-byte LUN value as yet or do not support the use of the SCSI-3 REPORT LUNS command. In such cases, it is expected that the HD will support 256 logical units or less, and report it's maximum LUN value during the binding process. All LUN values between 0 and the maximum can be addressed by the multi-lun PD (even if not enumerated). When specifying or interpreting the 8-byte LUN value for these non-SCSI-3 cases, the LUN value shall conform to the Single Level LUN Structure as per SAM-2 (i.e, byte 0 is zero, byte 1 contains the LUN value, and the remaining 6 bytes are zero).

The LUN addressing component is specified by the HD as an enumeration attribute on all non-multi-lun PD instances that it enumerates. On a per-request basis, the LUN component is specified only in SCSI I/O requests made by a multi-lun PD. I/O requests from all other PDs have the LUN value inherent in the channel the I/O request is issued on.

2.4 Peripheral Driver & HBA Driver Responsibilities

The SCSI Metalanguage divides operation and decision-making between PDs and HDs. In general, PDs make decisions specific to a particular SCSI device, and HDs make decisions that apply to an entire SCSI bus. This section provides some clarifications on each driver's responsibilities.

2.4.1 Retries

Both the PD and the HD may retry I/Os. However, any request that can affect device state must not be retried by the HD. The HD guarantees that it is immediately ready to accept a retry attempt from the PD, but, as with other operations, it may not be committed to the hardware instantly.

2.4.2 Timeouts

Both the PD and the HD may time I/Os or other requests. However, the HD will time I/O requests received via a `udi_scsi_io_req` operation if the PD specifies a nonzero timeout value in the request, so PDs should normally not time such requests. Other requests must be timed by the PD if a time limit is desired.

The timeout value specified by the PD in the `udi_scsi_io_req` should be much longer than the longest time that the specified request is expected to physically take in the device. To reduce variability, the HD must time the request only from the time it starts the request on the SCSI interconnect (not from the time it first receives the request from the PD); the HD must guarantee forward progress of internally queued requests. If a condition exists that prohibits forward progress, the HD must abort the request, returning it to the PD with the appropriate error indication. Even with the HD guaranteeing forward progress, there can still be significant variability in the delivery time across the interconnect to the device, so the PD must provide significant padding over and above the maximum expectation of time that the request can take in the device.

2.4.3 Aborts

The PD aborts individual SCSI I/O requests or control requests by `udi_channel_op_abort`, passing the control control block pointer for the original request. If the original request is still in the HD, the HD will receive a `udi_channel_event_ind` of type `UDI_CHANNEL_OP_ABORTED`, with the pointer to the original request. The HD is to then abort and then ack (or nak as appropriate) the original request to the PD. If the original request is no longer pending in the HD, the `udi_channel_op_abort` request will be discarded.

Control requests, which affect multiple SCSI I/O requests (Target Reset, etc), shall result in the desired function performed by the HD, followed by the the HD generating an ack (or nak as appropriate) of all affected I/O requests back to the PD. The HD will not respond with the corresponding ack for the control request until the function is performed and all affected requests have been returned to the PD.

2.4.4 Transfer Negotiation

The HD is responsible for maintaining the current state of transfer parameter negotiation (e.g., synchronous and wide parameters on parallel SCSI). It will renegotiate with the device whenever it believes that the negotiation has been lost, such as after a Unit Attention indicating SCSI bus reset. The

HD will always negotiate for the maximum transfer rate the device is capable of, unless the parent-visible instance attribute "`@scsi_max_xfer_rate`" has been set on any of the PDs associated with the device.

2.4.5 Task/Queue Management

The HD generates and maintains the tag values associated with tagged SCSI commands. Since multiple PDs can be bound to the same LUN, the HD must keep track of these tags on a per-LUN basis, not merely on a per-PD basis.

The HD is responsible for ensuring that the task ordering rules of SCSI are followed; this includes ensuring that tagged and untagged requests are not pending for the same device simultaneously.

The PD specifies its device's queue depth to the HD in the scsi bind request, and it is then the responsibility of the HD to guarantee that the queue depth to the device (i.e., the minimum of the queue depths specified by the PDs attached to a given LUN, with the exception of the multi-lun PDs which are excluded from this minimum calculation as noted below) is not exceeded. In this model the PD can send as many requests as it likes to the HD and the HD must manage per-LUN request queues to make sure that the number of requests outstanding on the device is not exceeded.

The PD can dynamically adjust its queue depth via the scsi_ctl request, in which case the queue depth change takes effect with respect to subsequent requests received from the PD.

It is the responsibility of the PD to handle QUEUE FULL conditions by reducing its queue depth as necessary to reduce the likelihood of QUEUE FULLs. This can occur for example in the presence of PDs on other initiators attached to the same LUN.

Note – multi-lun PDs are excluded from the minimum calculation of queue depth for a given LUN since a multi-LUN PD is effectively a generic SCSI pass-through driver which can address any target and LUN. Therefore, the HD must ignore the queue depth specified by multi-lun PDs (so as to not affect the queue depth requirements of more specific PDs).

2.4.6 SCSI Bus/Link Errors

The HD is responsible for detecting bus hangs or link errors, and responding appropriately to alleviate or recover from the condition where possible. Any affected I/O requests or control operations not recoverable at the link level will be returned to the requesting PD, with appropriate status, whether started at the device or not.

2.5 Bindings to the UDI Core Specification

2.5.1 Static Driver Properties Bindings

Some of the bindings for the static driver properties are defined in Section 2.1.1, "Versioning". This includes the definition of the relevant interface name(s) (i.e., the <interface_name> parameter on the "requires" and "provides" and other property declarations), and the definition of the interface version number for this version of this Specification.

The driver category to be used with the "category" declaration (see Section 31.5.3, "Category Declaration," on page 31-9 of the UDI Core Specification) by a portable implementation of the SCSI Metalanguage Library shall be "SCSI Host Bus Adapters".

The <attr_name> parameter values defined for use with the "custom" declaration (see Section 31.6.13, "Custom Declaration," on page 31-18 of the UDI Core Specification) are specified in Section 2.5.3.5, "Custom Parameter Attributes" below.

2.5.2 Transfer Constraints Bindings

The applicability and semantics of the UDI transfer constraints, defined in **udi_constraints_attr_t** on page 13-4 of the UDI Core Specification, are metalanguage-specific. For the SCSI Metalanguage, these apply only to the `udi_scsi_io_req` operation.

2.5.3 Instance Attributes Bindings

In each of the attribute tables below, the **ATTRIBUTE NAME** is a null-terminated string (see "Instance Attribute Names" on page 16-1 of the UDI Core Specification); the **TYPE** column specifies an attribute data type as defined in "udi_instance_attr_type_t" on page 16-7 of the UDI Core Specification; and the **SIZE** column specifies the valid sizes, in bytes, for each attribute.

2.5.3.1 Enumeration Attributes

The driver that enumerates SCSI peripheral devices (either the SCSI HD or an associated SCSI probe driver) must create the following enumeration attributes and pass them to the Management Agent in the ***`attr_list`*** parameter of the `udi_enumerate_ack` operation (see "Device Management Operations" on page 25-26 of the UDI Core Specification).

As the enumeration values describe the end device explicitly, including it's addressing components, if a

Table 2-1 SCSI Enumeration Attributes

ATTRIBUTE NAME	TYPE	SIZE	DESCRIPTION
scsi_bus	UDI_ATTR_UBIT32	4	SCSI bus number for this adapter, from 0
scsi_target	UDI_ATTR_UBIT32	4	SCSI Target ID
scsi_lun	UDI_ATTR_ARRAY8	1..8	8 bytes of SCSI LUN
scsi_inquiry	UDI_ATTR_ARRAY8	1..36	First 36 bytes of the SCSI INQUIRY data
scsi_dev_pqual	UDI_ATTR_UBIT32	4	SCSI Peripheral Qualifier (from INQUIRY)
scsi_dev_type	UDI_ATTR_UBIT32	4	SCSI Peripheral Device Type (from INQUIRY)
scsi_vendor_id	UDI_ATTR_STRING	1..9	SCSI Device Vendor ID (from INQUIRY)
scsi_product_id	UDI_ATTR_STRING	1..17	SCSI Device Product ID (from INQUIRY)
scsi_product_rev	UDI_ATTR_STRING	1..5	SCSI Device Product Revision (from INQUIRY)
scsi_multi_lun	UDI_ATTR_BOOLEAN	1	Multi-LUN Driver Instance
scsi_lun_wwid	UDI_ATTR_ARRAY8	8..16	LUN World-Wide ID
scsi_tgt_wwid	UDI_ATTR_ARRAY8	2..32	Target World-Wide ID
identifier	UDI_ATTR_STRING	1..45	Hex-encoding of LUN WWID or INQUIRY data
address_locator	UDI_ATTR_STRING	27	Hex-encoded concatenation of bus, target, LUN

hot plug event takes place, which is detected by the driver and results in the change of any enumeration parameters, the previous device is to be denumerated (potentially causing it to be unbound) and the newly detected device enumerated.

SCSI-3 architecturally allows for Target IDs and LUNs up to 64 bits in size. The LUN value follows the format as described in SAM-2. The Target ID value is a logical value, 32 bits in size, constructed by the HD. It is the HD's responsibility to map this to larger Target address spaces, where applicable. Refer to Section 2.3, "SCSI I/O Addressing" for additional details.

The "`scsi_bus`" attribute specifies the peripheral device's SCSI bus number, which is needed for multi-port SCSI HBAs (i.e., HBAs that have multiple SCSI buses) which, due to hardware inter-dependencies, must be controlled by a single SCSI HD instance. The range of this value is from 0 to 255.

The "`scsi_inquiry`" attribute provides the first 36 bytes of SCSI INQUIRY data associated with the peripheral device, or as many bytes as received from the device if less than 36 bytes were received.

The "`scsi_dev_pqual`" attribute is equivalent to the Peripheral Qualifier field in the high-order 3 bits of the first byte (byte 0) of the INQUIRY data. Note that it is a 4 byte integer attribute even though only the high order 3 bits are needed for the data.

The "`scsi_dev_type`" attribute is equivalent to the Peripheral Device Type field in the low-order 5 bits of the first byte (byte 0) of the INQUIRY data. Note that it is a 4 byte integer attribute even though only the low order 5 bits are needed for the data.

The "`scsi_vendor_id`" attribute provides an up to 9-byte vendor ID string (including a null terminator). The contents of this attribute are the 8 bytes of vendor ID in the SCSI INQUIRY data with trailing blanks and null characters removed and a null terminator appended. If the device does not provide a vendor ID string, this attribute shall be set to a null string.

The "`scsi_product_id`" attribute provides an up to 17-byte product ID string (including a null terminator). The contents of this attribute are the 16 bytes of product ID in the SCSI INQUIRY data with trailing blanks and null characters removed and a null terminator appended. If the device does not provide a product ID string, this attribute shall be set to a null string.

The "`scsi_product_rev`" attribute provides an up to 5-byte product revision string (including a null terminator). The contents of this attribute are the 4 bytes of product revision in the SCSI INQUIRY data with trailing blanks and null characters removed and a null terminator appended. If the device does not provide a product revision string, this attribute shall be set to a null string.

Note – The utility function, udi_strncpy_rtrim on page 21-6 of the UDI Core Specification, can be used to convert the corresponding character arrays in the SCSI INQUIRY data to the null-terminated strings required in the "scsi_vendor_id", "scsi_product_id", and "scsi_product_rev" attributes.

The "`scsi_multi_lun`" attribute indicates that the PD needs to access multiple LUNs on a single bind channel. If this type of binding is accepted, the PD will be able to pass the Target ID and LUN on each I/O request, in the CDB memory area. Note that PDs that require multi-lun access must include 12 additional bytes in ***`cdb_mem_size`*** when initializing their SCSI I/O control block properties. See the ***`cdb_ptr`*** definition in the `udi_scsi_io_cb_t` for additional details.

The "`scsi_lun_wwid`" attribute provides the world-wide ID string associated with the logical unit. Although there are several potential device identifiers, only two types of world-wide ID values shall be valid for use with this attribute. The first is the 8-byte FC-PH IEEE Registered value obtained via the Vital Product Data Device Identification page (page 0x83) of the SCSI INQUIRY command. The second is the 16-byte FC-PH IEEE Registered Extended value obtained via the Vital Product Data Device Identification page (page 0x83) of the SCSI INQUIRY command. If the device does not support either of these identifiers or the Vital Products Data page, or if the identifier cannot otherwise be obtained, the attribute must not be enumerated.

The "`scsi_tgt_wwid`" attribute provides an up to 32-byte world-wide ID string associated with the target device the logical unit is connected to. In many SCSI-3 interconnects, the target has a unique interconnect-specific world-wide identifier. For Fibre Channel, this attribute shall be set to the 16-byte N*_Port <PortName,NodeName> pair (in that order). For SIP (SCSI-3 Interlocked Protocol), this shall be the 32-byte SCAM (SCSI Configured Auto-Matically) identification string, as defined in Annex B of the SCSI-3 Parallel Interface specification (SPI). For SBP (the IEEE 1394 Serial Bus Protocol), this shall be the 2-byte Node ID, as defined by IEEE 1394-1995. If the interconnect does not support one of these world-wide identifiers, or if the identifier cannot be obtained, then this attribute must not be enumerated.

The remaining SCSI enumeration attributes are generic enumeration attributes, described below.

2.5.3.2 Generic Enumeration Attributes

2.5.3.2.1 identifier attribute

For SCSI peripheral devices, the "`identifier`" attribute encodes the device's LUN WWID value, if present, or its INQUIRY data, as a null-terminated string.

If the device has a "`scsi_lun_wwid`" value, then "`identifier`" encodes this value as a string of upper-case hexidecimal digits, two for each byte of the 8-byte value, with the first byte of data corresponding to the first two characters in the string, and so on.

If the device does not have a "`scsi_lun_wwid`" value, then "`identifier`" encodes up to 36 bytes of the SCSI INQUIRY data, with the first 8 bytes encoded as upper-case hexidecimal digits (two for each byte, starting with the first byte), followed by the next 28 bytes forced to be printable ASCII characters by replacing any byte outside the range 33..126 with the value 46 (ASCII for the period character, '.'). If there are fewer than 36 INQUIRY bytes, the "`identifier`" string is truncated accordingly.

Note – The utility function, **udi_scsi_inquiry_to_string** on page 3-36, can be used to convert SCSI INQUIRY to an "`identifier`" string.

2.5.3.2.2 address_locator attribute

For SCSI peripheral devices, the "`address_locator`" attribute encodes a concatentation of the bus, target, and lun attributes using the following syntax:

```
address_locator = BBTTTTTTTTLLLLLLLLLLLLLLLL
```

where `BB` is a two-digit upper-case hexidecimal-encoded ASCII representation of `scsi_bus`, TTTTTTTT is a four-digit upper-case hexidecimal-encoded ASCII representing `scsi_target` (first two digits encode the most significant byte, and so on), and LLLLLLLLLLLLLLLL is an 8-digit upper-case hexidecimal-encoded ASCII representation of `scsi_lun` (first two digits encode the first byte, and so on), respectively.

2.5.3.2.3 physical_locator attribute

No "`physical_locator`" attribute is defined for the SCSI metalanguage.

2.5.3.2.4 physical_label attribute

No "`physical_label`" attribute is defined generically for the SCSI metalanguage. Platforms that have access to such information may set `physical_label` attributes.

2.5.3.3 Filter Attributes

Of the above listed enumeration attributes, the following shall be supported as filter attributes for enumeration filtering:

```
scsi_bus
```

```
scsi_target
scsi_lun
```

Excepting `scsi_lun`, the **`attr_stride`** in the filter element structure (see **udi_filter_element_t** on page 25-15 of the UDI Core Specification) is interpreted linearly; that is, the stride value is simply added to the numeric value of these attributes. For `scsi_lun`, the ***`attr_stride`*** in the filter element structure will only apply to SCSI LUN values that conform to the Single Level LUN Structure as per SAM-2. In which case, the stride will be interpreted linearly on the encapsulated LUN value.

2.5.3.4 Parent-Visible Attributes

Parent-visible instance attributes, as defined in Chapter 16, *"Instance Attribute Management"*, of the UDI Core Specification, are attributes that the system sets on a PD instance, but are only visible to the HD. The following such attributes are currently defined:

Table 2-2 SCSI Parent-Visible Attributes

ATTRIBUTE NAME	TYPE	SIZE	DESCRIPTION
`@scsi_max_xfer_rate`	`UDI_ATTR_UBIT32`	4	Maximum transfer rate (in Mega-xfers/sec) for PD's device
`@scsi_pd_bus_reset_allowed`	`UDI_ATTR_BOOLEAN`	4	PD allowed to reset the bus?
`@scsi_max_temp_bind_excl`	`UDI_ATTR_UBIT32`	4	Maximum time (in milliseconds) of a TEMP_BIND_EXCLUSIVE bind

The "`@scsi_max_xfer_rate`" attribute must be requested by the HD during child binding. If the attribute exists for a given child, the HD must not attempt to negotiate with respect to that child's device for a transfer rate, in Mega-transfers per second, which is greater than the value of "`@scsi_max_xfer_rate`".

The "`@scsi_pd_bus_reset_allowed`" attribute must be requested by the HD during child binding. If the attribute doesn't exist for a given PD, the HD must fail any BUS_RESET control request from that child PD with UDI_STAT_NOT_SUPPORTED status. If the attribute exists and the HD can support BUS_RESET control requests, the HD must take the appropriate SCSI reset action. The value of this attribute is ignored.

The "`@scsi_max_temp_bind_excl`" attribute will be set by the environment on the PD instance before sending it a `udi_bind_to_parent_req`. The HD must obtain this attribute from the child PD's node when it receives a UDI_SCSI_TEMP_BIND_EXCLUSIVE bind. If the attribute doesn't exist or its value is zero the HD must reject the bind with a UDI_STAT_CANNOT_BIND_EXCLUSIVE status; otherwise, if not rejecting the bind for some other reason, the HD must return the value of this attribute in the `udi_scsi_bind_ack`. This allows environments to selectively disable the temporary-bind-exclusive capability.

2.5.3.5 Custom Parameter Attributes

The following <attr_name> parameter values are defined for use with the "custom" declaration (see Section 31.6.13, "Custom Declaration," on page 31-18 of the UDI Core Specification):

Table 2-3 SCSI Custom Parameter Attributes

ATTRIBUTE NAME	TYPE	SIZE	DESCRIPTION
%scsi_initiator_id	UDI_ATTR_UBIT32	4	HBA's own Target ID
%scsi_max_xfer_rate	UDI_ATTR_UBIT32	4	Maximum transfer rate (in Mega-xfers/sec) across the HBA

Each of these parameters applies to bus number 0 on the HBA. If the HBA has multiple buses, the above attribute names must be suffixed with an underscore followed by an ASCII-encoded decimal string representing the bus number, for bus numbers greater than zero.

2.5.4 Trace Event Bindings

The following defines the rules and conventions in the SCSI Metalanguage for the use of the metalanguage-selectable trace events (see the "Metalanguage-Selectable Trace Events" #defines in udi_trevent_t on page 18-3 of the UDI Core Specification).

- UDI_TREVENT_IO_SCHEDULED
  - The HD should trace at least ***data_buf->buf_size*** and the contents of the first byte of the CDB for udi_scsi_io_req operations, or the ***ctrl_func*** field for udi_scsi_ctl_req operations, as well as the corresponding control block pointer in either case.
- UDI_TREVENT_IO_COMPLETED
  - The HD should trace at least the contents of the first byte of the CDB and the udi_scsi_status_t structure (if non-zero) for udi_scsi_io_req operations, or the ***ctrl_func*** field for udi_scsi_ctl_req operations, as well as the corresponding control block pointer in either case. If an I/O request returns sense data, the trace should also include the sense key, additional sense code (ASC), and additional sense code qualifier (ASCQ) bytes. If an I/O request has an underrun, the number of bytes transferred should also be traced.
- UDI_TREVENT_META_SPECIFIC_1
  - The HD should trace the point at which a bus hang is detected, and should include at least the state of the SCSI bus lines if available. Both the PD and the HD should trace the sending or receiving of enabled SCSI events (from the events listed in the udi_scsi_bind_req), and should include at least the event code.
- UDI_TREVENT_META_SPECIFIC_2,
  UDI_TREVENT_META_SPECIFIC_3,
  UDI_TREVENT_META_SPECIFIC_4,
  UDI_TREVENT_META_SPECIFIC_5
  - Reserved for future use.

Note – All returned status values other than `UDI_OK` that indicate exceptional conditions must be logged and, when enabled, may also be traced, even if such events are expected.

2.6 Metalanguage State Diagram

See “Driver Instantiation” on page 25-2 of the UDI Core Specification for the general configuration sequence of UDI drivers. See “Management Metalanguage States” on page 25-36 of the UDI Core Specification for details on the Management Metalanguage states.

The following state diagram shows the SCSI metalanguage state diagram, which illustrates the set of states specific to use of the SCSI metalanguage. This same state diagram applies to both the PD and HD.

Figure 2-1 SCSI Metalanguage State Diagram

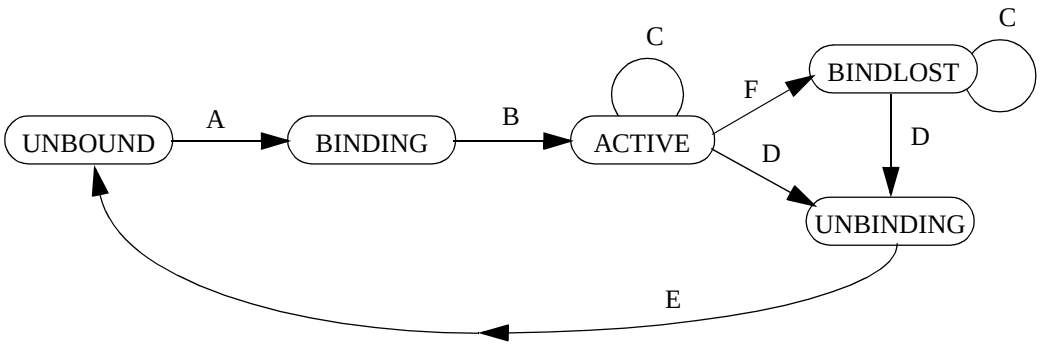

Table 2-4 SCSI Metalanguage Events

Event	Operation
A	`udi_scsi_bind_req`
B	`udi_scsi_bind_ack`
C	`udi_scsi_io_req, udi_scsi_io_ack, udi_scsi_io_nak, udi_scsi_ctl_req, udi_scsi_ctl_ack, udi_scsi_event_ind, udi_scsi_event_res`
D	`udi_scsi_unbind_req`
E	`udi_scsi_unbind_ack`
F	After binding `UDI_SCSI_TEMP_BIND_EXCLUSIVE`, the PD has remained bound longer than allowed by `@scsi_max_temp_bind_excl`.

2.6.1 SCSI Metalanguage States

UNBOUND — A SCSI channel in the unbound state has been established between the two regions but has not yet been initialized in those regions for general use. The PD side of the SCSI channel should initiate the SCSI bind operation when in this state.

BINDING — This state occurs when the PD side of the SCSI channel has initiated a bind operation and is waiting for the HD side of the SCSI channel to complete its initialization and acknowledge that bind request.

ACTIVE — This state occurs when the SCSI channel is fully bound between the two regions. The channel may be used for SCSI I/O and control operations or event indications.

BINDLOST — This state occurs when the PD has bound with the HD with the UDI_SCSI_TEMP_BIND_EXCL indicator set, and has not iniated an unbind request within the time frame allowable by the `@scsi_temp_bind_excl` attribute. When this state is encountered, the HD will reject all PD i/o and control requests with a UDI_STAT_INVALID_STATE status code, and will not post SCSI events to the PD.

UNBINDING — This indicates that the SCSI channel is being shut down. The PD can cause this state to be entered by issuing a `udi_scsi_unbind_req`. When the unbind operation is acknowledged, both the PD and the HD return to the UNBOUND state.

SCSI Metalanguage Interfaces 3

3.1 Introduction

This chapter specifies the channel operations and service calls of the SCSI Metalanguage. The SCSI Metalanguage defines the operations through which a SCSI peripheral driver (PD) communicates with a SCSI HBA driver (HD) and vice-versa. Subsections define the various channel operations' arguments, control block structures, constraints and guidelines for the use of each operation, along with any error conditions that can occur.

The SCSI Metalanguage operations are grouped into two roles: one for operations called by a PD to send an operation to the HD, and one for operations that are sent from the HD to the PD. Each role has a single channel ops vector. The PD-to-HD operations are called by a peripheral driver to request a service from the HBA driver, acknowledge an event from the HBA driver, or bind to the HBA driver. The HD-to-PD operations are called by an HBA driver to return completion information to the peripheral driver, to notify the peripheral driver of an asynchronous event, or to acknowledge a binding.

The only SCSI Metalanguage operation that is abortable with `udi_channel_op_aborted` is `udi_scsi_io_req`.

3.2 Overview of Interfaces and Data Structures

Channel operations in UDI are driver-callable functions that are used to communicate between driver modules. As alluded to above, there are two sets of channel operations in the SCSI Metalanguage which are distinguishable by the direction of communication: PD-to-HD operations are callable only by the PD, and HD-to-PD operations are callable only by the HD. Associated with each channel operation is a corresponding driver entry point in the target driver. Thus PD-to-HD operations are callable by PDs while HDs contain a corresponding driver entry point; and the reverse is true for HD-to-PD operations. Each driver registers its SCSI entry points by initializing a `udi_ops_init_t` in its `udi_init_info` to point to the driver's ops vector.

3.3 Control Blocks

The SCSI Metalanguage contains four types of control blocks for inter-driver communication: a **bind** control block for SCSI bind and unbind operations, an **io** control block for normal I/O requests on the SCSI interconnect, a **ctl** control block for control functions, and an **event** control block for asynchronous event notification and acknowledgment. These correspond to the four control block groups in the SCSI Metalanguage, one for each type of control block.

Control blocks contain space for an associated driver scratch area whose size requirement is specified in a corresponding `udi_cb_init_t` structure as part of `udi_init_info`. Access to the driver scratch area associated with a SCSI control block is provided via the ***`scratch`*** pointer in the ***`gcb`*** field at the front of the control block. This scratch area may be used for driver internal queuing, per-request state, or any other driver-specific purpose.

3.4 Status Codes

The following SCSI-specific status code values are defined in the SCSI Metalanguage. The semantic definitions of these statuses are given in the reference page where the status is used with a given operation or structure. The code values are consolidated here to simplify keeping the codes unique and to reduce clutter in the rest of the document. The value **`MS`** is replaced with **`UDI_STAT_META_SPECIFIC`** in each `#define` below.

```
/* SCSI I/O NAK status codes */
#define UDI_SCSI_STAT_NONZERO_STATUS_BYTE     (1|MS)
#define UDI_SCSI_STAT_ACA_PENDING             (2|MS)
#define UDI_SCSI_STAT_NOT_PRESENT             (3|MS)
#define UDI_SCSI_STAT_DEVICE_PHASE_ERROR      (4|MS)
#define UDI_SCSI_STAT_UNEXPECTED_BUS_FREE     (5|MS)
#define UDI_SCSI_STAT_DEVICE_PARITY_ERROR     (6|MS)
#define UDI_SCSI_STAT_ABORTED_HD_BUS_RESET    (7|MS)
#define UDI_SCSI_STAT_ABORTED_RMT_BUS_RESET   (8|MS)
#define UDI_SCSI_STAT_ABORTED_REQ_BUS_RESET   (9|MS)
#define UDI_SCSI_STAT_ABORTED_REQ_TGT_RESET   (10|MS)
#define UDI_SCSI_STAT_LINK_FAILURE            (11|MS)

/* SCSI Ctl Ack status codes */
#define UDI_SCSI_CTL_STAT_FAILED              (100|MS)
```

3.5 Channel Ops Vectors

This section defines the channel ops vector types for use with the SCSI Metalanguage. There are two ops vector types in the SCSI Metalanguage: one that a PD uses on its end of a SCSI channel (`udi_scsi_pd_ops_t`) and one that an HD uses on its end of a SCSI channel (`udi_scsi_hd_ops_t`).

NAME

udi_scsi_pd_ops_t *SCSI Peripheral Driver entry point ops vector*

SYNOPSIS

```
#include <udi.h>
#include <udi_scsi.h>

typedef struct {
     udi_channel_event_ind_op_t *channel_event_ind_op;
     udi_scsi_bind_ack_op_t *bind_ack_op;
     udi_scsi_unbind_ack_op_t *unbind_ack_op;
     udi_scsi_io_ack_op_t *io_ack_op;
     udi_scsi_io_nak_op_t *io_nak_op;
     udi_scsi_ctl_ack_op_t *ctl_ack_op;
     udi_scsi_event_ind_op_t *event_ind_op;
} udi_scsi_pd_ops_t;

/* Ops Vector Number */
#define UDI_SCSI_PD_OPS_NUM                1
```

DESCRIPTION

A SCSI Peripheral Driver uses the `udi_scsi_pd_ops_t` structure in a `udi_ops_init_t` as part of its `udi_init_info` in order to register its SCSI Metalanguage entry points.

EXAMPLE

The driver's `udi_init_info` might include the following:

```
#define MY_SCSI_OPS   1 /* Ops for my SCSI HBA parent */
#define MY_OTHER_OPS 2 /* Some other ops */
#define MY_SCSI_META 1 /* Meta index for the SCSI Metalanguage */
static const udi_scsi_pd_ops_t ddd_scsi_pd_ops = {
     ddd_scsi_channel_event_ind,
     ddd_scsi_bind_ack,
     ddd_scsi_unbind_ack,
     ddd_scsi_io_ack,
     ddd_scsi_io_nak,
     ddd_scsi_ctl_ack,
     ddd_scsi_event_ind
};
...
static const udi_ops_init_t ddd_ops_init_list[] = {
     {    MY_SCSI_OPS,
          MY_SCSI_META,
          UDI_SCSI_PD_OPS_NUM,
          0, /* chan_context_size */
          (udi_ops_vector_t *)&ddd_scsi_pd_ops },
     { 0 }
};
```

NAME

udi_scsi_hd_ops_t *SCSI HBA Driver entry point ops vector*

SYNOPSIS

```
#include <udi.h>
#include <udi_scsi.h>

typedef struct {
     udi_channel_event_ind_op_t *channel_event_ind_op;
     udi_scsi_bind_req_op_t *bind_req_op;
     udi_scsi_unbind_req_op_t *unbind_req_op;
     udi_scsi_io_req_op_t *io_req_op;
     udi_scsi_ctl_req_op_t *ctl_req_op;
     udi_scsi_event_res_op_t *event_res_op;
} udi_scsi_hd_ops_t;

/* Ops Vector Number */
#define UDI_SCSI_HD_OPS_NUM              2
```

DESCRIPTION

A SCSI HBA Driver uses the `udi_scsi_hd_ops_t` structure in a `udi_ops_init_t` as part of its `udi_init_info` in order to register its SCSI Metalanguage entry points.

EXAMPLE

The driver's `udi_init_info` might include the following:

```
#define MY_BRIDGE_OPS 1  /* Ops for my parent bridge */
#define MY_SCSI_OPS   2  /* Ops for my SCSI PD children */
#define MY_SCSI_META  1  /* Meta index for the SCSI Metalanguage */
#define MY_BUS_META   2  /* Meta index for Bus Bridge Metalanguage */
static const udi_scsi_hd_ops_t ddd_scsi_hd_ops = {
     ddd_scsi_channel_event_ind,
     ddd_scsi_bind_req,
     ddd_scsi_unbind_req,
     ddd_scsi_io_req,
     ddd_scsi_ctl_req,
     ddd_scsi_event_res
};
...
static const udi_ops_init_t ddd_ops_init_list[] = {
     {    MY_SCSI_OPS,
          MY_SCSI_META,
          UDI_SCSI_HD_OPS_NUM,
          0, /* chan_context_size */
          (udi_ops_vector_t *)&ddd_scsi_hd_ops },
     {    MY_BRIDGE_OPS,
          MY_BUS_META,
          UDI_BUS_BRIDGE_OPS_NUM,
          0, /* chan_context_size */
          (udi_ops_vector_t *)&ddd_bus_bridge_ops },
     ...
};
```

3.6 Binding Operations

Once the HD is bound to its parent, it can receive bindings for its PD children. As defined in “Driver Instantiation” on page 25-2 of the UDI Core Specification, this involves interaction with the Management Agent (MA) on the Management Channel and with the child PD directly on the newly created SCSI bind channel. The metalanguage-specific bind request and ack steps in that description in the Core Specification correspond to the `udi_scsi_bind_req` and `udi_scsi_bind_ack` operations. The `udi_scsi_bind_req`, sent by the PD to the HD, is the first operation passed on the SCSI bind channel. Once the HD has received and processed the `udi_scsi_bind_req`, if for a successful binding, the HD must then call `udi_constraints_propagate`, followed by `udi_scsi_bind_ack`.

At this point the SCSI channel is “open for business” and any of the other SCSI operations may be performed on it. Note that the PD must not do a `udi_scsi_bind_req` to the HD after this point without an intervening `udi_scsi_unbind_req`.

When it binds with the HD, the PD may request exclusive access to its device via the UDI_SCSI_BIND_EXCLUSIVE flag. If the HD finds that there is another PD already bound to the device then it will fail the bind with UDI_STAT_CANNOT_BIND_EXCLUSIVE status. A PD may also request “temporary exlusive access” via the UDI_SCSI_TEMP_BIND_EXCLUSIVE flag even while other PDs are bound to the device, as long as none of them are bound exclusively. While the HD has a PD bound exclusively (v ia either of the two exlusive bind flags), it will reject all other binds to that PD’s device with UDI_STAT_BOUND_EXCLUSIVELY.

NAME

udi_scsi_bind_cb_t *Control block for SCSI bind operations*

SYNOPSIS

```
#include <udi.h>
#include <udi_scsi.h>

typedef struct {
     udi_cb_t gcb;
     udi_ubit16_t events;
} udi_scsi_bind_cb_t;

/* SCSI Events */
#define UDI_SCSI_EVENT_AEN                    (1U<<0)
#define UDI_SCSI_EVENT_TGT_RESET              (1U<<1)
#define UDI_SCSI_EVENT_BUS_RESET              (1U<<2)
#define UDI_SCSI_EVENT_UNSOLICITED_RESELECT   (1U<<3)

/* Control Block Group Number */
#define UDI_SCSI_BIND_CB_NUM              1
```

MEMBERS

gcb is a generic control block header, which includes a pointer to the scratch space associated with this control block. The driver may use the scratch space while it owns the control block, but the values are not guaranteed to persist across channel operations.

events is a set of SCSI event types. On the `udi_scsi_bind_req` the PD sets the events for which it wants to be notified. On the corresponding ack the HD masks off any requested events that it doesn't support, and passes back that (potentially smaller) set of events to the PD. An HD for a parallel SCSI bus must support `UDI_SCSI_EVENT_TGT_RESET` and `UDI_SCSI_EVENT_BUS_RESET`. The ***events*** field is ignored on SCSI unbind operations.

Note that if a SCSI event occurs which affects the PDs on a given LUN, any multi-lun PDs which have the event enabled will be notified along with the single-lun PDs attached to the LUN.

The following events are defined:

UDI_SCSI_EVENT_AEN - Asynchronous Event Notification. The HD will send notification to the PD when its device sends a SCSI AEN, which is typically used to send notification of out-of-band events – i.e., device events that occur outside the context of a SCSI command from this HD initiator. If the PD finds that the HD doesn't support receiving AENs (by noting that the HD has cleared this flag), then the PD must poll for events that it cares about. The PD must also poll if it determines that its device does not support SCSI AEN.

UDI_SCSI_EVENT_TGT_RESET - A reset on the SCSI target ID has occured (via a BDR or Target Reset). The HD will only deliver this event for a locally requested Target Reset (as opposed to a reset due to another initiator), and in that case it will only deliver the event to other PDs attached to the LUN than the one that requested the reset (which includes any multi-lun PDs).

UDI_SCSI_EVENT_BUS_RESET - A SCSI bus reset occurred.

UDI_SCSI_EVENT_UNSOLICITED_RESELECT - The PD's LUN generated an unsolicited re-selection.

DESCRIPTION

The SCSI bind control block is used in `udi_scsi_bind_req`/`ack` and `udi_scsi_unbind_req`/`ack` operations.

In order to use this type of control block it must be associated with a control block index by including `UDI_SCSI_BIND_CB_NUM` in a `udi_cb_init_t` in the driver's `udi_init_info`.

REFERENCES

`udi_scsi_bind_req, udi_scsi_bind_ack, udi_scsi_unbind_req, udi_scsi_unbind_ack, udi_cb_alloc`

NAME

udi_scsi_bind_req *Request a SCSI binding (PD-to-HD)*

SYNOPSIS

```
#include <udi.h>
#include <udi_scsi.h>

void udi_scsi_bind_req (
     udi_scsi_bind_cb_t *cb,
     udi_ubit16_t bind_flags,
     udi_ubit16_t queue_depth,
     udi_time_t timeout_granularity,
     udi_ubit16_t max_sense_len,
     udi_ubit16_t aen_buf_size );

/* Bind Flags */
#define UDI_SCSI_BIND_EXCLUSIVE          (1U<<0)
#define UDI_SCSI_TEMP_BIND_EXCLUSIVE     (1U<<1)
```

ARGUMENTS

cb is a pointer to a SCSI bind control block.

bind_flags contains flags restricting or qualifying this binding. The following flags are defined. At most one of UDI_SCSI_BIND_EXCLUSIVE and UDI_SCSI_TEMP_BIND_EXCLUSIVE must be set.

UDI_SCSI_BIND_EXCLUSIVE - Indicates that the PD wants to get an exclusive bind to the SCSI device (target/LUN) associated with this bind channel. The HD will respond with a status of UDI_SCSI_CANNOT_BIND_EXCLUSIVE if another PD is currently bound to the target/LUN.

UDI_SCSI_TEMP_BIND_EXCLUSIVE - Indicates that the PD wants "temporary exclusive access" to the device. If no PD is currently exclusively bound to the device the HD will grant this bind as follows: the HD will quiesce any other PDs bound to the device (transparently to those PDs), and will then ack back to this PD with UDI_OK status. Any requests which come in from other PDs while this PD has exclusive access will be queued until this PD unbinds.

Drivers which bind with this flag must not remain bound any longer than the number of milliseconds specified by the ***max_temp_bind_excl*** parameter returned in the scsi_bind_ack; failure to do so is considered illegal driver behavior which can result in the PD instance being killed.

queue_depth is the number of SCSI commands that the HD is allowed to have pending to the device simultaneously. This is used to avoid excessive queue-full statuses. The PD may change this value later using the UDI_SCSI_CTL_SET_QUEUE_DEPTH control request. Note that the HD must manage queue depth on a per-LUN basis, guaranteeing that the queue depth to the device (i.e., the minimum of the queue depths specified by PDs attached to a given LUN) is not exceeded. See Section 2.4.5, "Task/Queue Management" for additional details.

timeout_granularity The HD is responsible for checking all pending I/O requests for timeout. The value of ***timeout_granularity*** is the maximum period of time that must elapse between such checks. The PD is responsible for ensuring that this value is consistent with the system's timer abilities reported via the ***min_timer_res*** field of udi_limits_t.

max_sense_len is the maximum sense data size required for the corresponding peripheral device. This sense data must be requested by the HD on behalf of the PD for all check conditions reported by the device. If ***max_sense_len*** is zero, the HD does not report the data to the PD.

Warning – While the HD is not required to request pending sense data from the device if ***max_sense_len*** is zero, it should do so, since not doing so could result in sense data building up on the device, particularly for multi-hosted devices.

---

aen_buf_size is the valid data size, in bytes, of AEN data buffers to be allocated by the HD and sent to the PD with each UDI_SCSI_EVENT_AEN event. If ***aen_buf_size*** is zero, no AEN buffers will be allocated.

aen_buf_size is ignored if the UDI_SCSI_EVENT_AEN event is either not supported by the HD or not enabled by the PD.

TARGET CHANNEL

The target channel for this operation is the bind channel connecting a SCSI PD to its parent HD.

DESCRIPTION

A SCSI PD uses this operation to bind to its parent HD.

The PD must prepare for the udi_scsi_bind_req operation by allocating a SCSI bind control block (calling udi_cb_alloc with a ***cb_idx*** that was previously associated with UDI_SCSI_BIND_CB_NUM). Next, the PD fills in the control block and sends it to the HD in a udi_scsi_bind_req operation.

The udi_scsi_bind_req operation must either be the first channel operation sent on the bind channel or the first operation since a SCSI unbind was done on the channel. The PD must not send any further operations on the bind channel until it receives the corresponding udi_scsi_bind_ack from the HD.

REFERENCES

udi_scsi_bind_cb_t, udi_scsi_bind_ack

NAME

udi_scsi_bind_ack *Acknowledge a SCSI bind request (HD-to-PD)*

SYNOPSIS

```
#include <udi.h>
#include <udi_scsi.h>

void udi_scsi_bind_ack (
     udi_scsi_bind_cb_t *cb,
     udi_ubit32_t max_targets,
     udi_ubit16_t max_luns,
     udi_ubit32_t max_temp_bind_excl,
     udi_status_t status );
```

ARGUMENTS

cb is a pointer to a SCSI bind control block.

max_targets is the maximum number of targets that the HD can enumerate. This may be an interconnect-specific limit or an HBA or driver-specific limit and must be non-zero. This parameter is only useful to a multi-LUN PD and should be ignored by other PDs.

max_luns is the maximum number of LUNs per target addressable by the HD and its associated hardware, if in the range 1..256; otherwise ***max_luns*** must be set to zero indicating that the HD and its associated hardware support full SCSI-3 LUN addressing per the SAM-2 Eight Byte LUN structure definition. This parameter is only useful to a multi-LUN PD and should be ignored by other PDs.

max_temp_bind_excl is the maximum time, in milliseconds, that a PD can remain bound via a UDI_SCSI_TEMP_BIND_EXCLUSIVE bind. The HD uses the "`@scsi_max_temp_bind_excl`" attribute to set this parameter. See Section 2.5.3.4, "Parent-Visible Attributes," on page 2-10 for details.

status is the status of this SCSI bind.

TARGET CHANNEL

The target channel for this operation is the bind channel connecting a SCSI HD to its child PD.

DESCRIPTION

The `udi_scsi_bind_ack` operation is used by an HD to acknowledge binding with a child PD (or failure to do so, as indicated by ***status***), as requested by a `udi_scsi_bind_req` operation.

STATUS VALUES

UDI_OK indicates that the SCSI bind succeeded.

UDI_STAT_BOUND_EXCLUSIVELY is returned by the HD to the PD to reject a bind because the device is bound exclusively by another PD and cannot therefore be bound by this PD.

UDI_STAT_CANNOT_BIND_EXCLUSIVE is returned by the HD to the PD to reject a UDI_SCSI_BIND_EXCLUSIVE bind when one or more PDs are already bound to this device and cannot therefore

be bound exclusively. This status is also used to reject a UDI_SCSI_TEMP_BIND_EXCLUSIVE bind when the "`@scsi_max_temp_bind_excl`" attribute doesn't exist or it's value is zero.

UDI_STAT_CANNOT_BIND indicates that the HD cannot bind to this PD for some reason other than exclusivity.

WARNINGS

The control block must be the same control block as passed to the HD in the corresponding `udi_scsi_bind_req` operation.

REFERENCES

`udi_scsi_bind_cb_t, udi_scsi_bind_req`

3.7 Unbinding Operations

NAME **udi_scsi_unbind_req** *Request a SCSI unbind (PD-to-HD)*

SYNOPSIS

```
#include <udi.h>
#include <udi_scsi.h>

void udi_scsi_unbind_req (
     udi_scsi_bind_cb_t *cb );
```

ARGUMENTS

cb is a pointer to a SCSI bind control block.

TARGET CHANNEL

The target channel for this operation is the bind channel connecting a SCSI PD to its parent HD.

DESCRIPTION

A SCSI PD uses this operation to unbind from its parent HD.

The PD must prepare for the `udi_scsi_unbind_req` operation by allocating a SCSI bind control block (calling `udi_cb_alloc` with a ***`cb_idx`*** that was previously associated with `UDI_SCSI_BIND_CB_NUM`). Next, the PD fills in the control block and sends it to the HD in a `udi_scsi_unbind_req` operation.

The PD may follow a SCSI unbind with another SCSI bind; the SCSI unbind in and of itself is not necessarily indicative of the PD instance going away.

REFERENCES

`udi_scsi_bind_cb_t, udi_scsi_unbind_ack`

NAME

udi_scsi_unbind_ack *Acknowledge a SCSI unbind (HD-to-PD)*

SYNOPSIS

```
#include <udi.h>
#include <udi_scsi.h>

void udi_scsi_unbind_ack (
     udi_scsi_bind_cb_t *cb );
```

ARGUMENTS

cb is a pointer to a SCSI bind control block.

TARGET CHANNEL

The target channel for this operation is the bind channel connecting a SCSI HD to its child PD.

DESCRIPTION

The `udi_scsi_unbind_ack` operation is used by a SCSI HD to acknowledge unbinding from a child PD, as requested by a `udi_scsi_unbind_req` operation.

There is no status parameter associated with this operation; the HD is expected to always be able to handle the unbind request and respond appropriately. If, for example, the HD were to receive an unbind from a PD without having first received a bind (or two unbinds in a row from the PD), the HD may log this condition but must always respond with this acknowledgment.

WARNINGS

The control block must be the same control block as passed to the HD in the corresponding `udi_scsi_unbind_req` operation.

REFERENCES

`udi_scsi_bind_cb_t, udi_scsi_unbind_req`

3.8 I/O Operations

NAME

udi_scsi_io_cb_t *Control block for SCSI I/O operations*

SYNOPSIS

```
#include <udi.h>
#include <udi_scsi.h>

typedef struct {
     udi_cb_t gcb;
     udi_buf_t *data_buf;
     udi_ubit32_t timeout;
     udi_ubit16_t flags;
     udi_ubit8_t attribute;
     udi_ubit8_t cdb_len;
     udi_ubit8_t *cdb_ptr;
} udi_scsi_io_cb_t;

/* I/O Request Flags */
#define UDI_SCSI_DATA_IN                    (1U<<0)
#define UDI_SCSI_DATA_OUT                   (1U<<1)
#define UDI_SCSI_NO_DISCONNECT              (1U<<2)

/* SCSI Task Attributes */
#define UDI_SCSI_SIMPLE_TASK                1
#define UDI_SCSI_ORDERED_TASK               2
#define UDI_SCSI_HEAD_OF_Q_TASK             3
#define UDI_SCSI_ACA_TASK                   4
#define UDI_SCSI_UNTAGGED_TASK              5

/* Control Block Group Number */
#define UDI_SCSI_IO_CB_NUM                  2
```

MEMBERS

gcb *is a* standard member at the front of SCSI control blocks, as defined in udi_scsi_bind_cb_t on page 3-7.

data_buf is a pointer to a buffer used to carry the data portion of a transfer. See `udi_scsi_io_req` and `udi_scsi_io_ack` for details on buffer usage.

timeout is an I/O timeout in milliseconds. A ***timeout*** value of zero specifies an infinite period, and it's up to the PD to time the I/O if it cares. The value of ***timeout*** will be rounded up by the HD to the nearest multiple of the operative timeout granularity in the HD (see `udi_scsi_bind_req`).

flags is a set of flags associated with this I/O request. The following flag bits are defined. At most one of **UDI_SCSI_DATA_IN** or **UDI_SCSI_DATA_OUT** must be specified. **UDI_SCSI_NO_DISCONNECT** may be optionally combined (ORed) with either of the IN/OUT flags.

UDI_SCSI_DATA_IN - Data-in from device to host.
UDI_SCSI_DATA_OUT - Data-out from host to device.

UDI_SCSI_NO_DISCONNECT - Disconnects on a parallel SCSI bus are disallowed during this I/O. On serial links this flag is ignored.

attribute is a SCSI-3 task attribute which specifies ordering and other constraints of requests (tasks) sent to the device. Exactly one of the following values may be associated with the I/O request by assigning it into the attribute field:

UDI_SCSI_SIMPLE_TASK
UDI_SCSI_ORDERED_TASK
UDI_SCSI_HEAD_OF_Q_TASK
UDI_SCSI_ACA_TASK
UDI_SCSI_UNTAGGED_TASK

All of the above attributes except **UDI_SCSI_ACA_TASK** are supported in the SCSI-2 architecture. If **UDI_SCSI_ACA_TASK** is passed to a SCSI-2 HD, it is the HD's responsibility to emulate SCSI-3 ACA behavior by freezing the queues in the HD that correspond to a given LUN and only allowing ACA_TASK'd requests through until a CLEAR_ACA control request is received from the PD.

cdb_len is the number of valid CDB bytes for this request. (This does not include any extra bytes used for multi-LUN addressing.)

cdb_ptr is a pointer to ***cdb_len*** bytes of SCSI CDB. If this is a multi-LUN binding the CDB bytes are followed by a 32-bit Target ID and a 64-bit LUN. The 32-bit Target ID shall be encoded as a little-endian quantity. The 64-bit LUN value shall be treated as an array of 8 bytes, formatted as per the SCSI Architecture Model - 2 (SAM-2) specification. Refer to Section 2.3 for further information on Target ID and LUN values.

This pointer is set by the environment when the control block is allocated and, like the scratch pointer, points to additional memory associated with this control block. The size of this memory area, and hence the maximum size of CDBs used by this driver, is set via the ***inline_size*** member of the relevant `udi_cb_init_t`, and must be incremented by 12 if this is a multi-LUN binding (i.e., maximum CDB size + 12) to provide room for the 32-bit Target ID and 64-bit LUN. The pointer itself must not be modified by the driver.

DESCRIPTION

The SCSI I/O control block is used between the PD and HD to process a SCSI I/O request.

In order to use this type of control block it must be associated with a control block index by including `UDI_SCSI_IO_CB_NUM` in a `udi_cb_init_t` in the driver's `udi_init_info`.

The size of the inline memory area pointed to by ***cdb_ptr*** must be specified using the ***inline_size*** member of that `udi_cb_init_t` structure (see Chapter 10, *"Initialization"*, of the UDI Core Specification). The memory is treated as an array of unstructured bytes. (I.e. ***cdb_ptr*** is a `UDI_DL_INLINE_UNTYPED` field.) Since the HD never allocates this type of control block, it must set ***inline_size*** to zero.

REFERENCES `udi_scsi_io_req, udi_scsi_io_ack, udi_scsi_io_nak, udi_cb_alloc`

NAME **udi_scsi_io_req** *Request a SCSI I/O operation (PD-to-HD)*

SYNOPSIS

```
#include <udi.h>
#include <udi_scsi.h>

void udi_scsi_io_req (
     udi_scsi_io_cb_t *cb );
```

ARGUMENTS

cb is a pointer to a SCSI IO control block.

TARGET CHANNEL

The target channel for this operation is the bind channel connecting a SCSI PD to its parent HD.

DESCRIPTION

A PD uses this operation to send a SCSI I/O request to its parent HD.

The PD must prepare for the `udi_scsi_io_req` operation by allocating a SCSI I/O control block (calling `udi_cb_alloc` with a ***cb_idx*** that was previously associated with `UDI_SCSI_IO_CB_NUM`) and filling in all of its members.

The PD indicates the desired transfer size by setting ***data_buf->buf_size*** to the desired number of bytes. If no bytes are to be transferred, the PD may set ***data_buf*** to `NULL`.

If ***flags*** in the control block do not include `UDI_SCSI_DATA_OUT`, any data in ***data_buf*** is not guaranteed to be preserved by this channel operation. That is, when the HD receives this operation, the contents (but not the size) of the buffer are unspecified unless `UDI_SCSI_DATA_OUT` is set.

This operation is abortable with `udi_channel_op_abort`.

REFERENCES

`udi_scsi_io_cb_t, udi_scsi_io_ack, udi_scsi_io_nak`

NAME

udi_scsi_io_ack *Acknowledge normal completion of SCSI I/O request*

SYNOPSIS

```
#include <udi.h>
#include <udi_scsi.h>

void udi_scsi_io_ack (
     udi_scsi_io_cb_t *cb );
```

ARGUMENTS

cb is a pointer to a SCSI IO control block.

TARGET CHANNEL

The target channel for this operation is the bind channel connecting a SCSI HD to its child PD.

DESCRIPTION

The `udi_scsi_io_ack` operation is used by an HD to acknowledge the normal completion of a SCSI I/O request back to a child PD, in response to a `udi_scsi_io_req` operation. This operation must be used to indicate to the PD a completion status of UDI_OK; otherwise, the `udi_scsi_io_nak` operation must be used.

If ***data_buf*** is not `NULL`, ***data_buf->buf_size*** must be the same as in the original request and must equal the number of bytes actually transferred. The ***data_buf*** pointer must either be the same as in the original request, or a direct "descendant" of the original buffer (i.e. results from a chain of one or more service calls such as `udi_buf_write` that replace the original buffer with a modified version).

If ***flags*** in the control block do not include `UDI_SCSI_DATA_IN`, any data in ***data_buf*** is not guaranteed to be preserved by this channel operation. That is, when the PD receives this operation, the contents (but not the size) of the buffer are unspecified unless `UDI_SCSI_DATA_IN` is set.

WARNINGS

The control block must be the same control block as passed to the HD in the corresponding `udi_scsi_io_req` operation.

REFERENCES

`udi_scsi_io_cb_t, udi_scsi_io_req, udi_scsi_io_nak`

NAME

udi_scsi_io_nak *Indicate abnormal completion of SCSI I/O request*

SYNOPSIS

```
#include <udi.h>
#include <udi_scsi.h>

void udi_scsi_io_nak (
     udi_scsi_io_cb_t *cb,
     udi_scsi_status_t status,
     udi_buf_t *sense_buf );
```

ARGUMENTS

cb is a pointer to a SCSI IO control block.

status is the status of the I/O request.

sense_buf is a pointer to the sense data buffer containing the details of a SCSI command failure if the values in ***status*** are set to indicate a CHECK CONDITION; otherwise, ***sense_buf*** must be NULL. If non-NULL, ***sense_buf->buf_size*** must equal the number of bytes of valid sense data. If there are no sense data bytes, ***sense_buf*** may be NULL.

TARGET CHANNEL

The target channel for this operation is the bind channel connecting a SCSI HD to its child PD.

DESCRIPTION

The udi_scsi_io_nak operation is used by an HD to indicate abnormal completion of a SCSI I/O request back to a child PD, in response to a udi_scsi_io_req operation. This operation must be used to indicate a status other than UDI_OK to the PD. The udi_scsi_io_ack operation must be used to indicate a UDI_OK status in which the exact amount of data requested was transferred.

If ***data_buf*** is not NULL, the HD must set ***data_buf->buf_size*** to the number of bytes actually transferred, which must be less than or equal to the requested size. The ***data_buf*** pointer must either be the same as in the original request, or a direct "descendant" of the original buffer (i.e. results from a chain of one or more service calls such as udi_buf_write that replace the original buffer with a modified version).

If ***flags*** in the control block include UDI_SCSI_DATA_OUT, the contents of the data buffer must be the same as in the original request. This allows the PD to retry failed operations if it so chooses.

Data in ***data_buf*** is always preserved by this channel operation.

After receiving and processing a udi_scsi_io_nak, the PD must free ***sense_buf***, if non-NULL, by calling udi_buf_free. The HD can reclaim the sense data buffer by copying it before sending it off in the nak; in many environment implementations this will be accomplished (via copy-on-write semantics) without any actual data copy.

WARNINGS

The control block must be the same control block as passed to the HD in the corresponding udi_scsi_io_req operation.

REFERENCES `udi_scsi_io_cb_t, udi_scsi_io_req, udi_scsi_io_ack`

NAME

udi_scsi_status_t *Status structure in SCSI I/O Acknowledgement*

SYNOPSIS

```
#include <udi.h>
#include <udi_scsi.h>

typedef struct {
     udi_status_t req_status;
     udi_ubit8_t scsi_status;
     udi_ubit8_t sense_status;
} udi_scsi_status_t;
```

MEMBERS

req_status is the main software status associated with this I/O request. See below for definitions of status values.

scsi_status is the status byte received over the SCSI bus as defined in the SCSI protocol.

sense_status is the status byte for the REQUEST SENSE command on the SCSI bus. This is valid only when ***scsi_status*** indicates a CHECK CONDITION.

DESCRIPTION

The SCSI status structure gives the status of the I/O request on I/O completion.

STATUS VALUES

The following status codes are defined for ***req_status***:

UDI_SCSI_STAT_ACA_PENDING – A contingent-allegiance condition remains pending at the SCSI device.

UDI_SCSI_STAT_SELECTION_TIMEOUT – SCSI device did not respond to selection.

UDI_SCSI_STAT_DEVICE_PHASE_ERROR – Adapter detected an illegal SCSI bus phase change on the part of the device.

UDI_SCSI_STAT_UNEXPECTED_BUS_FREE – Either the adapter or the device terminated the command prematurely by putting the SCSI bus in a free state.

UDI_SCSI_STAT_DEVICE_PARITY_ERROR – Adapter detected a parity error on the part of the device. (Device-detected parity errors are reported through sense data.)

UDI_SCSI_STAT_ABORTED_HD_BUS_RESET – The I/O command was aborted by a SCSI bus reset generated internally by the HD, probably to resolve a SCSI bus hang. This I/O command may or may not have actually been started on the device. It is unknown if this particular command caused a bus hang.

UDI_SCSI_STAT_ABORTED_RMT_BUS_RESET – The I/O command was aborted by a SCSI bus reset generated by a device on the SCSI bus other than the adapter that this HD controls. This I/O command may or may not have actually been started on the device.

UDI_SCSI_STAT_ABORTED_REQ_BUS_RESET – The I/O command was aborted by a SCSI bus reset requested by the PD. This I/O command may or may not have actually been started on the device.

UDI_SCSI_STAT_ABORTED_REQ_TGT_RESET – The I/O command was aborted by a SCSI target reset requested by the PD. This I/O command may or may not have actually been started on the device.

UDI_SCSI_STAT_LINK_FAILURE – The link between the adapter and the device has failed. It is unknown whether this command was active at the time of hardware failure.

UDI_SCSI_STAT_NONZERO_STATUS_BYTE – Device request returned a non-zero SCSI status byte and no other status value applies. If another status code is applicable the HD must set ***req_status*** to the other applicable value. The PD must check ***scsi_status*** regardless of the value of ***req_status***.

UDI_STAT_NOT_UNDERSTOOD – Request received from the PD was invalid.

UDI_STAT_TIMEOUT – This command was timed out by the HD and aborted.

UDI_STAT_ABORTED – I/O command was successfully aborted or terminated by a control operation sent by the PD. The command may or may not have been actually started at the device.

UDI_STAT_DATA_OVERRUN – This target device attempted to send more data than was requested.

UDI_STAT_DATA_UNDERRUN – This target device sent less data than was requested.

UDI_STAT_HW_PROBLEM – This command terminated with an indeterminate hardware error.

Note that **UDI_STAT_NOT_UNDERSTOOD**, **UDI_STAT_TIMEOUT**, **UDI_STAT_ABORTED**, **UDI_STAT_HW_PROBLEM**, **UDI_STAT_OVERRUN**, and **UDI_STAT_UNDERRUN** are common status codes whose constant values are defined in Chapter 9, *"Fundamental Types"*, of the UDI Core Specification.

REFERENCES `udi_scsi_io_cb_t, udi_scsi_io_ack`

3.9 Control Operations

NAME

udi_scsi_ctl_cb_t *Control block for SCSI control operations*

SYNOPSIS

```
#include <udi.h>
#include <udi_scsi.h>

typedef struct {
     udi_cb_t gcb;
     udi_ubit8_t ctrl_func;
     udi_ubit16_t queue_depth;
} udi_scsi_ctl_cb_t;

/* Values for ctrl_func */
#define UDI_SCSI_CTL_ABORT_TASK_SET       1
#define UDI_SCSI_CTL_CLEAR_TASK_SET       2
#define UDI_SCSI_CTL_LUN_RESET            3
#define UDI_SCSI_CTL_TGT_RESET            4
#define UDI_SCSI_CTL_BUS_RESET            5
#define UDI_SCSI_CTL_CLEAR_ACA            6
#define UDI_SCSI_CTL_SET_QUEUE_DEPTH      7

/* Control Block Group Number */
#define UDI_SCSI_CTL_CB_NUM               3
```

MEMBERS

gcb, tr_context are standard members at the front of SCSI control blocks, as defined in udi_scsi_bind_cb_t on page 3-7.

ctrl_func is one of the following SCSI control functions:

UDI_SCSI_CTL_ABORT_TASK_SET -- Sends the appropriate interconnect-specific operation to abort all tasks in the Logical Unit's task set for the requesting initiator. This will cause the the corresponding completion operations to be sent to the PD before responding with the `udi_scsi_ctl_ack`. On interconnects that do not support this operation, ***ctrl_func*** is typically a no-op and the status UDI_STAT_NOT_SUPPORTED is returned in the `udi_scsi_ctl_ack`.

UDI_SCSI_CTL_CLEAR_TASK_SET -- Sends the appropriate interconnect-specific operation to abort all tasks in the Logical Unit's task set for all initiators. This will cause the the corresponding completion operations to be sent to the PDs before responding with the `udi_scsi_ctl_ack`. On interconnects that do not support this operation, ***ctrl_func*** is typically a no-op and the status UDI_STAT_NOT_SUPPORTED is returned in the `udi_scsi_ctl_ack`.

UDI_SCSI_CTL_LUN_RESET -- Sends the appropriate interconnect-specific operation to perform a logical unit reset as defined by SAM-2. This will cause the the corresponding completion operations to be sent to the PDs before responding with the `udi_scsi_ctl_ack`. On interconnects that do not

support this operation, ***ctrl_func*** is typically a no-op and the status UDI_STAT_NOT_SUPPORTED is returned in the `udi_scsi_ctl_ack`.

UDI_SCSI_CTL_TGT_RESET -- Sends the appropriate interconnect-specific operation to perform a target-level reset to the SCSI device. This will abort all I/Os outstanding on the LUNs attached to the specified SCSI target and will cause corresponding completion operations to be sent to the PD before responding with the `udi_scsi_ctl_ack`. Note that this may be used

UDI_SCSI_CTL_BUS_RESET -- Reset the SCSI bus (parallel SCSI only) associated with this SCSI device. This will abort all I/Os outstanding on the bus and will cause corresponding completion operations to be sent to the PD before responding with the `udi_scsi_ctl_ack`. On serial SCSI links this ***ctrl_func*** is typically a no-op and the status UDI_STAT_NOT_SUPPORTED is returned in the `udi_scsi_ctl_ack`.

UDI_SCSI_CTL_CLEAR_ACA -- Clear auto-contingent-allegiance condition at the SCSI device.

UDI_SCSI_CTL_SET_QUEUE_DEPTH -- Change the maximum number of commands the HD is allowed to have pending to the device on behalf of this PD simultaneously. This takes effect with respect to subsequent requests received from the PD. QUEUE FULL conditions are handled by the PD.

queue_depth **is the maximum** number of commands the HD is allowed to have pending to the device simultaneously. Used only with UDI_SCSI_CTL_SET_QUEUE_DEPTH.

DESCRIPTION

The control block for SCSI Control operations is used between the PD and HD to process a SCSI Control request.

In order to use this type of control block it must be associated with a control block index by including `UDI_SCSI_CTL_CB_NUM` in a `udi_cb_init_t` in the driver's `udi_init_info`.

REFERENCES

`udi_scsi_ctl_req, udi_scsi_ctl_ack, udi_cb_alloc`

NAME

udi_scsi_ctl_req *Request a SCSI control operation (PD-to-HD)*

SYNOPSIS

```
#include <udi.h>
#include <udi_scsi.h>

void udi_scsi_ctl_req (
     udi_scsi_ctl_cb_t *cb );
```

ARGUMENTS

cb is a pointer to a SCSI ctl control block.

TARGET CHANNEL

The target channel for this operation is the bind channel connecting a SCSI PD to its parent HD.

DESCRIPTION

A PD uses this operation to send a SCSI Control request to its parent HD.

The PD must prepare for the `udi_scsi_ctl_req` operation by allocating a `scsi_ctl` control block (calling `udi_cb_alloc` with a ***`cb_idx`*** that was previously associated with `UDI_SCSI_CTL_CB_NUM`). Next, the PD fills in the control block and sends it to the HD in a `udi_scsi_ctl_req` operation.

REFERENCES

`udi_scsi_ctl_cb_t, udi_scsi_ctl_ack`

NAME

udi_scsi_ctl_ack *Ack completion of SCSI control request (HD-to-PD)*

SYNOPSIS

```
#include <udi.h>
#include <udi_scsi.h>

void udi_scsi_ctl_ack (
     udi_scsi_ctl_cb_t *cb,
     udi_status_t status );
```

ARGUMENTS

cb is a pointer to a SCSI ctl control block.

status is the status of the SCSI control request, and shall be one of the following values:

UDI_OK -- Normal completion.

UDI_STAT_HW_PROBLEM -- Adapter hardware error prevented completion of control request.

UDI_STAT_NOT_UNDERSTOOD -- Control request block is invalid.

UDI_STAT_NOT_SUPPORTED -- Control request block is not in error, but the specific request type is not supported by the HD.

UDI_SCSI_CTL_STAT_FAILED -- Control request failed for some other unspecified reason.

Note that all the status codes except **UDI_SCSI_CTL_STAT_FAILED** are common status codes whose constant values are defined in Chapter 9, *"Fundamental Types"*, of the UDI Core Specification.

TARGET CHANNEL

The target channel for this operation is the bind channel connecting a SCSI HD to its child PD.

DESCRIPTION

`udi_scsi_ctl_ack` is called by an HD to acknowledge completion of a SCSI Control request back to a child PD (indicating success or failure), as requested by a `udi_scsi_ctl_req` operation.

WARNINGS

The control block must be the same control block as passed to the HD in the corresponding `udi_scsi_ctl_req` operation.

REFERENCES

`udi_scsi_ctl_cb_t, udi_scsi_ctl_req`

3.10 Event Operations

NAME

udi_scsi_event_cb_t *Control block for SCSI event operations*

SYNOPSIS

```
#include <udi.h>
#include <udi_scsi.h>

typedef struct {
     udi_cb_t gcb;
     udi_ubit8_t event;
     udi_buf_t *aen_data_buf;
} udi_scsi_event_cb_t;

/* Control Block Group Number */
#define UDI_SCSI_EVENT_CB_NUM              4
```

MEMBERS

gcb is a standard members at the front of SCSI control blocks, as defined in udi_scsi_bind_cb_t on page 3-7.

event is the type of asynchronous event. See the ***event*** field in the `udi_scsi_bind_cb_t` for valid event types.

aen_data_buf is a pointer to a data buffer containing AEN data and is only valid if UDI_SCSI_EVENT_AEN is set in ***event***; otherwise it must be set to `NULL`. In the AEN case, ***aen_data_buf*** must contain ***aen_buf_size*** bytes of valid data, as specified in the `udi_scsi_bind_req`. If ***aen_buf_size*** was zero, ***aen_data_buf*** must be `NULL`. It is legal in the SCSI architecture to send zero bytes of data with an AEN, so if the PD's device supports AEN but always sends zero bytes of data (indicating to the PD that it should go check its device) then this would be an example where an ***aen_buf_size*** of zero would be appropriate.

See `udi_scsi_event_ind` and `udi_scsi_event_res` for additional details on the usage of AEN buffers.

DESCRIPTION

The SCSI event control block is used between the HD and its PD children to notify the PD of an asynchronous event.

In order to use this type of control block it must be associated with a control block index by including `UDI_SCSI_EVENT_CB_NUM` in a `udi_cb_init_t` in the driver's `udi_init_info`.

REFERENCES

`udi_scsi_event_ind, udi_scsi_event_res, udi_cb_alloc`

NAME

udi_scsi_event_ind *SCSI event notification (HD-to-PD)*

SYNOPSIS

```
#include <udi.h>
#include <udi_scsi.h>

void udi_scsi_event_ind (
     udi_scsi_event_cb_t *cb );
```

ARGUMENTS

cb is a pointer to a SCSI event control block.

TARGET CHANNEL

The target channel for this operation is the bind channel connecting a SCSI HD to its child PD.

PROXIES

udi_scsi_event_ind_unused *Proxy for udi_scsi_event_ind*

```
udi_scsi_event_ind_op_t
     udi_scsi_event_ind_unused;
```

`udi_scsi_event_ind_unused` may be used as a PD's `udi_scsi_event_ind` entry point if the PD never enables any events for notification.

DESCRIPTION

An HD uses the `udi_scsi_event_ind` operation to send an event notification to its child PD.

The HD must prepare for the `udi_scsi_event_ind` operation by allocating a SCSI event control block (calling `udi_cb_alloc` with a ***cb_idx*** that was previously associated with `UDI_SCSI_EVENT_CB_NUM`).

If ***event*** is UDI_SCSI_EVENT_AEN and ***aen_buf_size*** in the `udi_scsi_bind_req` was nonzero, the HD must also obtain an AEN buffer containing ***aen_buf_size*** valid bytes. In this case, if the size of the AEN data received from the device is greater than ***aen_buf_size***, only the first ***aen_buf_size*** byte will be placed in the buffer; if the size of the data is less than ***aen_buf_size***, the remaining bytes will be part of the buffer's valid data range, but their values unspecified.

Next, the HD sends the SCSI event control block to the PD with a `udi_scsi_event_ind` operation. The HD does not need to wait to receive a response before sending another `udi_scsi_event_ind`; multiple indications may be pending at once.

Whether or not an HD supports a particular type of event notification, and whether or not the PD has enabled those events, is negotiated in the SCSI bind operations.

Note that some events (e.g., SCSI bus reset) can be triggered by a PD through control operations. The event is still sent to the PD that requested the control operation, and is sent before the control operation completes.

REFERENCES

`udi_scsi_event_cb_t, udi_scsi_event_res`

NAME

udi_scsi_event_res *Acknowledge a SCSI event (PD-to-HD)*

SYNOPSIS

```
#include <udi.h>
#include <udi_scsi.h>

void udi_scsi_event_res (
     udi_scsi_event_cb_t *cb );
```

ARGUMENTS

cb is a pointer to a SCSI event control block.

TARGET CHANNEL

The target channel for this operation is the bind channel connecting a SCSI HD to its child PD.

DESCRIPTION

The `udi_scsi_event_res` operation is used by a PD to acknowledge an event indication from its parent HD, as delivered by a `udi_scsi_event_ind` operation.

If ***event*** is UDI_SCSI_EVENT_AEN, the ***aen_data_buf*** handle must have the same value as was received in the `udi_scsi_event_ind`, and the buffer itself must not have been modified by the PD.

WARNINGS

The control block must be the same control block as passed to the PD in the corresponding `udi_scsi_event_ind` operation.

REFERENCES

`udi_scsi_event_cb_t, udi_scsi_event_ind`

3.11 Utility Functions

NAME

udi_scsi_inquiry_to_string *Encode SCSI INQUIRY data as a string*

SYNOPSIS

```
#include <udi.h>
#include <udi_scsi.h>

void udi_scsi_inquiry_to_string (
     const udi_ubit8_t *inquiry_data,
     udi_size_t inquiry_len,
     char *str );
```

ARGUMENTS

inquiry_data is a pointer to up to 36 bytes of SCSI INQUIRY data.

inquiry_len is the length of the ***inquiry_data*** array.

str is a pointer to a character array, at least 45 bytes long, that will be filled in with the encoded string.

DESCRIPTION

This utility function converts binary SCSI INQUIRY data to a string, encoded as required for the “identifier” attribute when the device does not have a LUN World-Wide ID. See Section 2.5.3.2.1, “identifier attribute,” on page 2-8 for details.

Index