Uniform Driver Interface

UDI SCSI Driver Specification Version 0.90

Note – This version (Version 0.90) is intended to be functionally complete. It is presented to the industry at large for broad public review. Upon completion of a two-month review period, review comments will be incorporated to form the final 1.0 specification. Review comments may be submitted to `http://www.sco.com/UDI/review.html`.

`http://www.sco.com/UDI/specs.html`

Table Of Contents

List of Reference Pages by Chapter

Chapter 2 SCSI Metalanguage

Alphabetical List of Reference Pages

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.

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).

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 HD. 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.

1.5.1 Definitions

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

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.

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 the SCSI Metalanguage channel.

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, etc.).

1.5.2 Acronyms

HBA Host Bus Adapter

HD SCSI HBA Driver

LUN Logical Unit Number

PD SCSI Peripheral Driver

SAM SCSI-3 Architectural Model

SCSI Small Computer System Interface

1.6 Versioning and Header Files

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. For this version of the specification, UDI_SCSI_VERSION must be set to 0x090:

```
#define UDI_SCSI_VERSION 0x090
```

Each device 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 SCSI UDI interfaces and services.

SCSI Metalanguage 2

2.1 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.

2.2 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. The PD registers its SCSI entry points by passing its ops vector to `udi_scsi_pd_ops_init`, and the HD registers its SCSI entry points by passing its ops vector to `udi_scsi_hd_ops_init`.

Note – The channel operation definitions only show function prototypes for the caller side interfaces; the callee-side interfaces (i.e. the function prototypes for the corresponding driver entry points) can be derived from the caller-side interfaces by removing the first parameter, the target channel. See "Channel Operations" on page 6-4 of the UDI Core Specification for additional information.

Each of the SCSI control blocks contains a transaction context pointer (***`tr_context`***). This pointer (together with the channel that a request is received on) is used to uniquely identify a transaction for purposes of aborts, and must therefore be unique among outstanding requests from a given requestor.

Warning – The transaction context pointer (***`tr_context`***) passed by the PD in request transactions must not be dereferenced by the HD and must be passed back unchanged to the PD in the corresponding acknowledgement control block. The only operations allowed on the ***`tr_context`*** by the HD are assignments and comparisons (e.g., for

purposes of aborts). Similarly, ***tr_context*** when passed by the HD to the PD in an event indication transaction must not be dereferenced by the PD and must be passed back unchanged by the PD in the corresponding event response.

Quantities that are naturally 64-bit integers (i.e., that have a MSB..LSB order, such as LUN and TAG) are split into two 32-bit integers and specified as to which is least significant and most significant. Since none of these 64-bit quantities require 64-bit arithmetic operations on the them, this allows the same driver source to work on compilers with and without 64-bit integer support on both big and little endian processors.

2.3 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 via an argument to each of the four `udi_scsi_xxx_cb_init` routines. 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.

2.4 Peripheral Driver and Host Bus Adapter 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 mustn't 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 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.

2.4.3 Aborts

The PD aborts SCSI I/O requests or control requests by passing the associated tr_context pointer to the HD in a scsi_abort_req operation. The HD will not respond with the corresponding scsi_abort_ack until the specified I/O or control request has been completed (ack'd) back to the PD.

2.4.4 Transfer Negotiation

The HD is responsible for maintaining the current state of synchronous or wide negotiation. 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_maximum_xfer`", has been set on the PD(s) 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.

In general, 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.

It is the responsibility of the PD to handle QUEUE FULL conditions by reducing its queue depth as necessary, via the `UDI_SCSI_CTL_SET_QUEUE_DEPTH` control request.

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 SCSI Trace Events

The following defines the rules and conventions in the SCSI Metalanguage for the use of the metalanguage-specific trace events (see the "Metalanguage Trace Events" in the "udi_trevent_t" on page 17-3 of the UDI Core Specification).

- `UDI_TREVENT_IO_SCHEDULED`
  - Trace at least the corresponding control block pointer and *tr_context*.
- `UDI_TREVENT_IO_COMPLETED`
  - Trace at least the corresponding control block pointer, *tr_context*, and *status*.
- `UDI_TREVENT_STATE_CHANGE`
  - Trace state changes which occur during the SCSI bind/unbind sequence. For the HD this includes tracing the state transitions from "unbound to a particular child" to "bind_req received" to "constraints propogated to child" to "bind_ack sent", and similarly for its unbind sequence. For the PD this includes tracing the state transitions from "unbound to a particular parent" to "bind_req sent" to "constraints received" to "bind_ack received", and similarly for its unbind sequence.
- `UDI_TREVENT_META_SPECIFIC_1`
  - Used by HD on detection of a bus hang.

- `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. In the case of sense data, the HD should only indicate that such data is being passed to the PD. Tracing the contents or interpretation of sense data is left to the PD.

2.6 Bindings for Instance Attributes

In each of the attribute tables below, the **ATTRIBUTE NAME** is a null-terminated string (see "Instance Attribute Names" on page 15-1 of the UDI Core Specification); the **TYPE** column specifies an attribute data type as defined in "udi_instance_attr_type_t" on page 15-6 of the UDI Core Specification; and the **SIZE** column specifies the valid sizes, in bytes, for each attribute.

2.6.1 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 21-42 of the UDI Core Specification).

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	Low-order 4 bytes of SCSI Target ID
`!scsi_target_hi`	`UDI_ATTR_UBIT32`	4	High-order 4 bytes of SCSI Target ID
`!scsi_lun`	`UDI_ATTR_UBIT32`	4	Low-order 4 bytes of SCSI LUN
`!scsi_lun_hi`	`UDI_ATTR_UBIT32`	4	High-order 4 bytes of SCSI LUN
`!scsi_inquiry`	`UDI_ATTR_STRING`	1..36	First 36 bytes of the SCSI INQUIRY data
`!scsi_dev_type`	`UDI_ATTR_UBIT32`	4	SCSI Peripheral Device Type
`!scsi_vendor_id`	`UDI_ATTR_UBIT32`	4	SCSI Device Vendor ID (from INQUIRY)
`!scsi_product_id`	`UDI_ATTR_UBIT32`	4	SCSI Device Product ID (from INQUIRY)
`!scsi_multi_lun`	`UDI_ATTR_UBIT32`	4	Multi-LUN Driver Instance

SCSI-3 architecturally allows for Target IDs and LUNs up to 64-bits in size, which are split into two 32-bit attributes.

The "`!scsi_bus`" attribute specifies the peripheral device's SCSI bus number, which is needed for multi-channel SCSI HBAs controlled by a single SCSI HD instance.

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_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.

The "`!scsi_vendor_id`" attribute provides the 8-byte vendor_id string from the SCSI INQUIRY data.

The "`!scsi_product_id`" attribute provides the 8-byte product_id string from the SCSI INQUIRY data.

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 16 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.

2.6.1.1 Filter Attributes

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

```
!scsi_bus
!scsi_target
!scsi_target_hi
!scsi_lun
!scsi_lun_hi
```

For each of these, the **`attr_stride`** in the filter element structure (see "udi_filter_element_t" on page 21-31 of the UDI Core Specification) is interpreted linearly; that is, the stride value is simply added to the numeric value of these attributes.

2.6.1.2 Locator Attribute

The locator attribute for SCSI peripheral devices encodes a concatenation of the filter attributes listed above, using the following syntax:

```
!locator = BBTTTTttttLLLLllll
```

where `BB` is a two-digit upper-case hexidecimal-encoded ASCII representation of `!scsi_bus`, and TTTT, tttt, LLLL, and llll are four-digit upper-case hexidecimal-encoded ASCII representations of `!scsi_target_hi`, `!scsi_target`, `!scsi_lun_hi`, and `!scsi_lun`, respectively.

2.6.2 Parent-Visible Attributes

Parent-visible instance attributes, as defined in Chapter 15, *"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 Child Attributes

ATTRIBUTE NAME	TYPE	SIZE	DESCRIPTION
`@scsi_maximum_xfer`	`UDI_ATTR_UBIT32`	4	Maximum transfer rate (in Mega-xfers/sec)
`@scsi_pd_bus_reset_allowed`	`UDI_ATTR_UBIT32`	4	Maximum transfer rate (in Mega-xfers/sec)

The "`@scsi_maximum_xfer`" attribute must be requested by the HD during child binding. If the attribute exists for a given child, the HD mustn't 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_maximum_xfer`".

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 child, the HD must fail any RESET_BUS control request from that child PD with UDI_STAT_NOT_SUPPORTED status.

2.7 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_CAC_PENDING           (2|MS)
#define UDI_SCSI_STAT_SELECTION_TIMEOUT     (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_DEV_RESET (10|MS)
#define UDI_SCSI_STAT_LINK_FAILURE          (11|MS)

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

2.8 Initialization Functions

This section defines the module global initialization functions provided in the SCSI Metalanguage. These are only callable from a driver's `init_module` routine. See "Module-Global Initialization Functions" on page 6-2 of the UDI Core Specification for additional information.

2.8.1 Channel Ops Vector Registration

There are two ops vectors in the SCSI Metalanguage: one that the PD uses on its end of the SCSI channel (`udi_scsi_pd_ops_t`), and one that the HD uses on its end of the channel (`udi_scsi_hd_ops_t`).

NAME

udi_scsi_pd_ops_init *Register SCSI Peripheral Driver entry points*

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;

void udi_scsi_pd_ops_init (
     udi_index_t ops_idx,
     udi_scsi_pd_ops_t *ops );
```

ARGUMENTS

ops_idx, ops are standard arguments described in "Channel Ops Properties Initialization" on page 6-2 of the UDI Core Specification.

DESCRIPTION

`udi_scsi_pd_ops_init` is called by a peripheral driver's `init_module` routine to register its SCSI Metalanguage entry points.

WARNINGS

This call may only be made from the `init_module` routine of one of the driver's modules.

EXAMPLE

The driver's `init_module` routine might include the following:

```
#define MY_SCSI_OPS   1 /* Ops for my SCSI HBA parent */
#define MY_OTHER_OPS 2 /* Some other ops */
const udi_scsi_pd_ops_t ddd_scsi_pd_ops = {
     ddd_scsi_channel_event_ind,
     ddd_scsi_bind_ack,
     ddd_scsi_io_ack,
     ddd_scsi_io_nak,
     ddd_scsi_ctl_ack,
     ddd_scsi_event_ind
};
...
void init_module(void)
{
     ...
     udi_scsi_pd_ops_init(
          MY_SCSI_OPS,
          &ddd_scsi_pd_ops);
     ...
}
```

NAME

udi_scsi_hd_ops_init *Register SCSI HBA Driver entry points*

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;

void udi_scsi_hd_ops_init (
     udi_index_t ops_idx,
     udi_scsi_hd_ops_t *ops );
```

ARGUMENTS

ops_idx, ops are standard arguments described in "Channel Ops Properties Initialization" on page 6-2 of the UDI Core Specification.

DESCRIPTION

`udi_scsi_hd_ops_init` is called by a SCSI HBA driver's `init_module` routine to register its SCSI Metalanguage entry points.

WARNINGS

This call may only be made from the `init_module` routine of one of the driver's modules.

EXAMPLE

The driver's `init_module` routine 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_OTHER_OPS 3  /* Some other ops */
const udi_scsi_hd_ops_t ddd_scsi_hd_ops = {
     ddd_scsi_channel_event_ind,
     ddd_scsi_bind_req,
     ddd_scsi_io_req,
     ddd_scsi_ctl_req,
     ddd_scsi_event_res
};
...
void init_module(void)
{
     ...
     udi_scsi_hd_ops_init(
          MY_SCSI_OPS,
          &ddd_scsi_hd_ops);
     ...
}
```

2.8.2 Control Block Properties Registration

NAME

udi_scsi_bind_cb_init — *Register SCSI bind control block properties*

SYNOPSIS

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

void udi_scsi_bind_cb_init (
        udi_index_t cb_idx,
        udi_size_t scratch_requirement);
```

ARGUMENTS

cb_idx, scratch_requirement are standard arguments described in "Control Block Properties Initialization" on page 6-3 of the UDI Core Specification.

DESCRIPTION

A device driver calls `udi_scsi_bind_cb_init` to register its scratch size requirements for SCSI bind control blocks.

WARNINGS

This call may only be made from the `init_module` routine of one of the driver's modules.

REFERENCES

`udi_cb_alloc, udi_scsi_bind_cb_t`

NAME

udi_scsi_io_cb_init *Register SCSI I/O control block properties*

SYNOPSIS

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

void udi_scsi_io_cb_init (
        udi_index_t cb_idx,
        udi_size_t scratch_requirement,
        udi_ubit8_t cdb_mem_size );
```

ARGUMENTS

cb_idx, scratch_requirement are standard arguments described in "Control Block Properties Initialization" on page 6-3 of the UDI Core Specification.

cdb_mem_size is the size in bytes of needed by the PD for the SCSI CDB memory area in SCSI I/O control blocks that will be allocated using ***cb_idx***.

DESCRIPTION

A device driver calls `udi_scsi_io_cb_init` to register its scratch and SCSI CDB size requirements for SCSI I/O control blocks.

WARNINGS

This call may only be made from the `init_module` routine of one of the driver's modules.

REFERENCES

`udi_cb_alloc, udi_scsi_io_cb_t`

NAME

udi_scsi_ctl_cb_init — *Register SCSI ctl control block properties*

SYNOPSIS

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

void udi_scsi_ctl_cb_init (
        udi_index_t cb_idx,
        udi_size_t scratch_requirement);
```

ARGUMENTS

cb_idx, scratch_requirement are standard arguments described in "Control Block Properties Initialization" on page 6-3 of the UDI Core Specification.

DESCRIPTION

A device driver calls `udi_scsi_ctl_cb_init` to register its scratch size requirements for SCSI ctl control blocks.

WARNINGS

This call may only be made from the `init_module` routine of one of the driver's modules.

REFERENCES

`udi_cb_alloc, udi_scsi_ctl_cb_t`

NAME

udi_scsi_event_cb_init *Register SCSI event control block properties*

SYNOPSIS

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

void udi_scsi_event_cb_init (
        udi_index_t cb_idx,
        udi_size_t scratch_requirement);
```

ARGUMENTS

cb_idx, scratch_requirement are standard arguments described in "Control Block Properties Initialization" on page 6-3 of the UDI Core Specification.

DESCRIPTION

A device driver calls `udi_scsi_event_cb_init` to register its scratch size requirements for SCSI event control blocks.

WARNINGS

This call may only be made from the `init_module` routine of one of the driver's modules.

REFERENCES

`udi_cb_alloc, udi_scsi_event_cb_t`

2.9 Binding Operations

Once the HD is bound to its parent, it can receive bindings for its PD children. As defined in “Binding Operations” on page 21-15 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 in steps 4b and 4d of that description in the Core Specification are 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` (step 4c), 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. While the HD has a PD bound exclusively, 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;
     void *tr_context;
     udi_ubit16_t events;
} udi_scsi_bind_cb_t;

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

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.

tr_context is a transaction context pointer passed by the PD to the HD in the request, and must be passed back unchanged by the HD in the corresponding acknowledgement.

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. HDs that control parallel SCSI buses must support `UDI_SCSI_EVENT_DEVICE_RESET` and `UDI_SCSI_EVENT_BUS_RESET`. The ***events*** field is ignored on SCSI unbind operations.

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_DEVICE_RESET - A SCSI bus device reset (BDR) has occurred on the Target ID for this PD.

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.

REFERENCES

`udi_scsi_bind_req, udi_scsi_bind_ack, udi_scsi_unbind_req, udi_scsi_unbind_ack, udi_scsi_bind_cb_init, 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_channel_t target_channel,
     udi_scsi_bind_cb_t *cb,
     udi_ubit16_t bind_flags,
     udi_ubit16_t queue_depth,
     udi_time_t timeout_granularity,
     udi_size_t max_sense_len,
     udi_size_t aen_buf_size );

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

ARGUMENTS

target_channel, cb are standard arguments described in "Channel Operations" on page 6-4 of the UDI Core Specification.

bind_flags contains flags restricting or qualifying this binding. The following flags are defined:

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.

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.

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 passed to `udi_scsi_bind_cb_init`). 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_channel_t target_channel,
     udi_scsi_bind_cb_t *cb,
     udi_status_t status );
```

ARGUMENTS

target_channel, cb are standard arguments described in "Channel Operations" on page 6-4 of the UDI Core Specification.

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` indicates that another PD is currently exclusively bound to the SCSI device (target/LUN) associated with this bind channel.

UDI_STAT_CANNOT_BIND_EXCLUSIVE indicates that one or more PDs are already bound to this device, so it cannot be bound exclusively.

`UDI_STAT_CANNOT_BIND` indicates that the HD cannot bind to this PD.

REFERENCES

`udi_scsi_bind_cb_t, udi_scsi_bind_req`

2.10 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_channel_t target_channel,
     udi_scsi_bind_cb_t *cb );
```

ARGUMENTS

target_channel, cb are standard arguments described in "Channel Operations" on page 6-4 of the UDI Core Specification.

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 passed to `udi_scsi_bind_cb_init`). 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_channel_t target_channel,
     udi_scsi_bind_cb_t *cb );
```

ARGUMENTS

target_channel, cb are standard arguments described in "Channel Operations" on page 6-4 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

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.

REFERENCES

`udi_scsi_bind_cb_t, udi_scsi_unbind_req`

2.11 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;
     void *tr_context;
     udi_size_t data_len;
     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            (1<<0)
#define UDI_SCSI_DATA_OUT           (1<<1)
#define UDI_SCSI_NO_DISCONNECT      (1<<2)
#define UDI_SCSI_OVERRUN            (1<<3)

/* 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
```

MEMBERS

gcb, tr_context are standard members at the front of SCSI control blocks, as defined in udi_scsi_bind_cb_t on page 2-16.

data_len is the number of data bytes to be transferred when passed in the request, or the number of bytes actually transferred when passed in the ack or nak. See `udi_scsi_io_req/ack/nak` for details on how this length interacts with ***flags*** and ***data_buf***.

data_buf is a buffer handle for 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.

UDI_SCSI_OVERRUN – The SCSI device wanted to transfer more data than the request had available. [NAK CASE]

UDI_SCSI_UNDERRUN – The SCSI device transferred less data than requested by the PD. Depending on the operation, this may or may not be considered an error. See the ***data_len*** field in `udi_scsi_io_cb_t` to calculate the amount of underrun. [CHANGE TO JUST USE data_len] [NAK CASE]

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 64-bit target id and a 64-bit LUN in SCSI's big endian format.

This pointer is set up 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 ***cdb_mem_size*** parameter to `udi_scsi_io_cb_init`, which must be incremented by 16 if this is a multi-LUN binding (maximum CDB size + 16).

DESCRIPTION

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

REFERENCES `udi_scsi_io_req, udi_scsi_io_ack, udi_scsi_io_cb_init, 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_channel_t target_channel,
     udi_scsi_io_cb_t *cb );
```

ARGUMENTS ***target_channel, cb*** are standard arguments described in "Channel Operations" on page 6-4 of the UDI Core Specification.

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 passed to `udi_scsi_io_cb_init`) and filling in all of its members.

If ***flags*** in the control block includes neither `UDI_SCSI_DATA_IN` nor `UDI_SCSI_DATA_OUT`, ***data_len*** is ignored and ***data_buf*** must be set to `NULL_BUF`. Otherwise ***data_buf*** must be set to `NULL_BUF` if ***data_len*** is zero, or to a valid buffer handle if ***data_len*** is not zero.

If ***data_buf*** is not `NULL_BUF`, it must contain exactly ***data_len*** bytes of valid data if ***flags*** includes `UDI_SCSI_DATA_OUT`, or zero valid data bytes if it does not.

REFERENCES `udi_scsi_io_cb_t, 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. This is valid only when ***req_status*** is UDI_SCSI_STAT_NONZERO_STATUS_BYTE.

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_NONZERO_STATUS_BYTE – SCSI status was returned by the device. See the ***scsi_status*** field for that value.

UDI_SCSI_STAT_CAC_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_DEV_RESET – The I/O command was aborted by a SCSI bus device 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_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_HW_PROBLEM – This command terminated with an indeterminate hardware error.

UDI_OK – No specific error (used with overrun/underrun conditions).

Note that **`UDI_OK`**, **`UDI_STAT_NOT_UNDERSTOOD`**, **`UDI_STAT_TIMEOUT`**, **`UDI_STAT_ABORTED`**, and **`UDI_STAT_HW_PROBLEM`** are common status codes whose constant values are defined in Chapter 8, *"Fundamental Types"* of the UDI Core Specification.

REFERENCES `udi_scsi_io_cb_t, udi_scsi_io_ack`

NAME

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

SYNOPSIS

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

void udi_scsi_io_ack (
     udi_channel_t target_channel,
     udi_scsi_io_cb_t *cb );
```

ARGUMENTS

target_channel, cb are standard arguments described in "Channel Operations" on page 6-4 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

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 a status of UDI_OK to the PD; otherwise, the `udi_scsi_io_nak` operation must be used.

If ***flags*** in the control block includes either `UDI_SCSI_DATA_IN` or `UDI_SCSI_DATA_OUT`, ***data_len*** must be set to the amount of data actually transferred. Otherwise, ***data_len*** is ignored.

If ***flags*** includes `UDI_SCSI_DATA_OUT`, the original ***data_buf*** must be freed or reused by the HD and ***data_buf*** must be set to NULL_BUF. If ***flags*** includes `UDI_SCSI_DATA_IN`, the ***data_buf*** member must have the same value as in the request, or must be the result of a series of one or more `udi_buf_copy` or `udi_buf_write` calls with ***data_buf*** as the destination buffer; and ***data_buf*** must reference a buffer which contains exactly ***data_len*** bytes of valid data.

REFERENCES

`udi_scsi_io_cb_t, udi_scsi_io_req`

NAME

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

SYNOPSIS

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

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

ARGUMENTS

target_channel, cb are standard arguments described in "Channel Operations" on page 6-4 of the UDI Core Specification.

status is the status of the I/O request.

sense_buf is a handle to the sense data buffer which is only valid if the appropriate status values are set in ***status***. If there's no valid sense data the HD must set this to NULL_BUF so that region-crossing code will not try to transfer the (non-existent) object referenced by ***sense_buf***. ***sense_buf*** must have a valid data size equal to the number of bytes of valid sense data. (This can be obtained by calling `udi_buf_get_size`.)

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, or to indicate a data overrun or underrun condition. 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.

The requirements on the setting of ***data_len*** are the same as those specified in `udi_scsi_io_ack`.

For `UDI_SCSI_DATA_IN`, the requirements on ***data_buf*** are the same as those specified in `udi_scsi_io_ack`, which in turn are the same as what is required here (on the nak) for `UDI_SCSI_DATA_OUT` (i.e., ***data_buf*** must contain ***data_len*** bytes of valid data in either the data-in or data-out case on the nak). On a data-out the data contents sent back on the ack must be the same as those which were sent on the corresponding req, to facilitate retries.

After receiving and processing a `udi_scsi_io_nak`, the PD must free ***sense_buf*** 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.

REFERENCES

`udi_scsi_io_cb_t, udi_scsi_io_req`

2.12 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;
     void *tr_context;
     udi_ubit8_t ctrl_func;
     void *orig_tr_context;
     udi_ubit16_t queue_depth;
} udi_scsi_ctl_cb_t;

/* ctrl_func codes */
#define UDI_SCSI_CTL_ABORT           1
#define UDI_SCSI_CTL_TERMINATE       2
#define UDI_SCSI_CTL_RESET_BUS       3
#define UDI_SCSI_CTL_RESET_DEVICE    4
#define UDI_SCSI_CTL_SET_QUEUE_DEPTH 5
#define UDI_SCSI_CTL_CLEAR_ACA       6
```

MEMBERS

gcb, tr_context are standard members at the front of SCSI control blocks, as defined in udi_scsi_bind_cb_t on page 2-16.

ctrl_func is one of the following SCSI control functions:

UDI_SCSI_CTL_ABORT -- Abort the SCSI I/O Request whose ***tr_context*** matches this control block's ***orig_tr_context***. The request being aborted will be completed with appropriate status before the control operation is completed.

UDI_SCSI_CTL_TERMINATE -- Same as UDI_SCSI_CTL_ABORT, but on a parallel SCSI bus send a SCSI-2 TERMINATE I/O PROCESS message to the SCSI device instead of a SCSI-2 ABORT message.

UDI_SCSI_CTL_RESET_BUS -- 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_RESET_DEVICE -- Issue a reset to the SCSI device. This will abort all I/Os outstanding on the device and will cause corresponding completion operations to be sent to the PD before responding with the `udi_scsi_ctl_ack`.

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.

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

orig_tr_context is the ***tr_context*** value of an I/O request to be terminated or aborted. Used only with UDI_SCSI_CTL_ABORT and UDI_SCSI_CTL_TERMINATE.

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.

REFERENCES

`udi_scsi_ctl_req, udi_scsi_ctl_ack, udi_scsi_ctl_cb_init, 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_channel_t target_channel,
     udi_scsi_ctl_cb_t *cb );
```

ARGUMENTS

target_channel, cb are standard arguments described in "Channel Operations" on page 6-4 of the UDI Core Specification.

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 passed to `udi_scsi_ctl_cb_init`). 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_channel_t target_channel,
     udi_scsi_ctl_cb_t *cb,
     udi_status_t status );
```

ARGUMENTS

target_channel, cb are standard arguments described in "Channel Operations" on page 6-4 of the UDI Core Specification.

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 8, *"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.

REFERENCES

`udi_scsi_ctl_cb_t, udi_scsi_ctl_req`

2.13 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;
     void *tr_context;
     udi_ubit8_t event;
     udi_buf_t aen_data_buf;
} udi_scsi_event_cb_t;
```

MEMBERS

gcb, tr_context are standard members at the front of SCSI control blocks, as defined in udi_scsi_bind_cb_t on page 2-16.

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 handle 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_BUF`. 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_BUF`. 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.

REFERENCES

`udi_scsi_event_ind, udi_scsi_event_res, udi_scsi_event_cb_init, 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_channel_t target_channel,
     udi_scsi_event_cb_t *cb );

udi_scsi_event_ind_op_t
     udi_scsi_event_ind_unused;
```

ARGUMENTS ***target_channel, cb*** are standard arguments described in "Channel Operations" on page 6-4 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 An HD uses this 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 passed to `udi_scsi_event_cb_init`).

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.

DEFAULT ENTRYPT `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.

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_channel_t target_channel,
     udi_scsi_event_cb_t *cb );
```

ARGUMENTS ***target_channel, cb*** are standard arguments described in "Channel Operations" on page 6-4 of the UDI Core Specification.

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

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.

REFERENCES `udi_scsi_event_cb_t, udi_scsi_event_ind`

Glossary *A*

adapter I/O hardware which provides a specific function or connectivity/bridging capability and which is accessed via a system bus. Also called a ***card***, a ***controller*** or an ***HBA***. The adapter is typically accessed via programmed I/O and may be capable of generating interrupts or DMA activity (or be capable of being a DMA target).

adapter driver device driver software responsible for managing an adapter.

controller see ***adapter***.

device physical hardware, under software control, which is typically attached either directly to an I/O bus or to an auxiliary bus (e.g. SCSI) attached to a directly-connected adapter. The device typically combines a hardware controller with the raw mechanism (disk controller with disk, display controller with frame buffer, etc.).

device driver a software module that turns I/O requests into control of a specific physical device or a hardware or protocol interface. A device driver contains all the device-specific code necessary to control and communicate with its hardware or logical function and provides a standard interface to the rest of the system. A driver may or may not control "raw" hardware.

device ID a numeric or string value with a device-interconnect specified format used to provide device identification. Usually stored in I/O card ROM.

device node a node in the *device tree*.

device tree an abstract data structure that represents the physical and logical topology of an I/O system. This data structure is usually thought of as an n-ary tree structure, but can occasionally have multiple parents for the same node, so is really a directed graph. Even with multiple parents, however, the graph ultimately has a single root. Each node represents a device instance.

driver see ***device driver***.

driver instance a set of one or more regions, all belonging to the same driver, that are associated with a particular instance of the driver's device. There may be multiple instances of a given driver, one for each physical device controlled or (in the case of software-only drivers) one for each logically-separate replication of a function. Each active device node has exactly one corresponding driver instance.

HBA Host Bus Adapter. Another name for an ***adapter***, but most commonly used for SCSI adapters.

Logical Unit Number (LUN) 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 second level of SCSI addressing: the "target" is level one and the "tag" is level

three. The address of one of many peripheral devices (like a SCSI disk) on the target interface which connects to the I/O system. Typically there are many SCSI disks drives on a single SCSI interface connect.

metalanguage an agreement between the respective channels of a driver and one or more clients; the operations that can be performed on each channel; the syntax and semantics of each operation; and the relationships among, and allowable sequences of, a set of channel operations. There are, for example, (in other UDI Specifications) a metalanguage defined for SCSI devices and one for Networking devices. Other metalanguages will be defined as more UDI drivers for other classes of devices are needed.

parent driver instance of a pair of communicating driver instances, the one whose position in the device tree is closer to the root of the tree. For example, a SCSI host bus adapter is usually the child of a SCSI disk. The parent driver instance is typically the *server* in this relationship.

peripheral driver an I/O driver controlling a specific peripheral device or class of devices.

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 definition. 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..

tag nominally the third level of SCSI I/O addressing that designates a specific I/O request; the “target” is level one and the “LUN” is level two. The use of tags is typical in “tagged command queueing” on SCSI-2 and “tagged tasks” on SCSI-3.

Target ID (Target) The first level of SCSI I/O addressing, normally corresponding with a given crate on a SCSI bus. (The “LUN” is level two and the “tag” is level three of SCSI addressing.)