iSCSI Data Mover

Design Specification

Version 0.3

June 2008

Table of Contents

1 Introduction

1.1 Background

The iSCSI Extensions for RDMA (iSER) project will implement an iSER initiator and iSER target using InfiniBand as a transport. iSER will provide a storage interconnect that leverages all the capabilities of InfiniBand including

- High bandwidth
- Low CPU utilization due to the use of Remote Direct Memory Access (RDMA)
- Single network connection (multiple protocols can share an IB link)

As a transport for the iSCSI protocol, iSER also provides all the advantages of iSCSI technology including

- iSCSI naming services
- Centralized management via iSNS

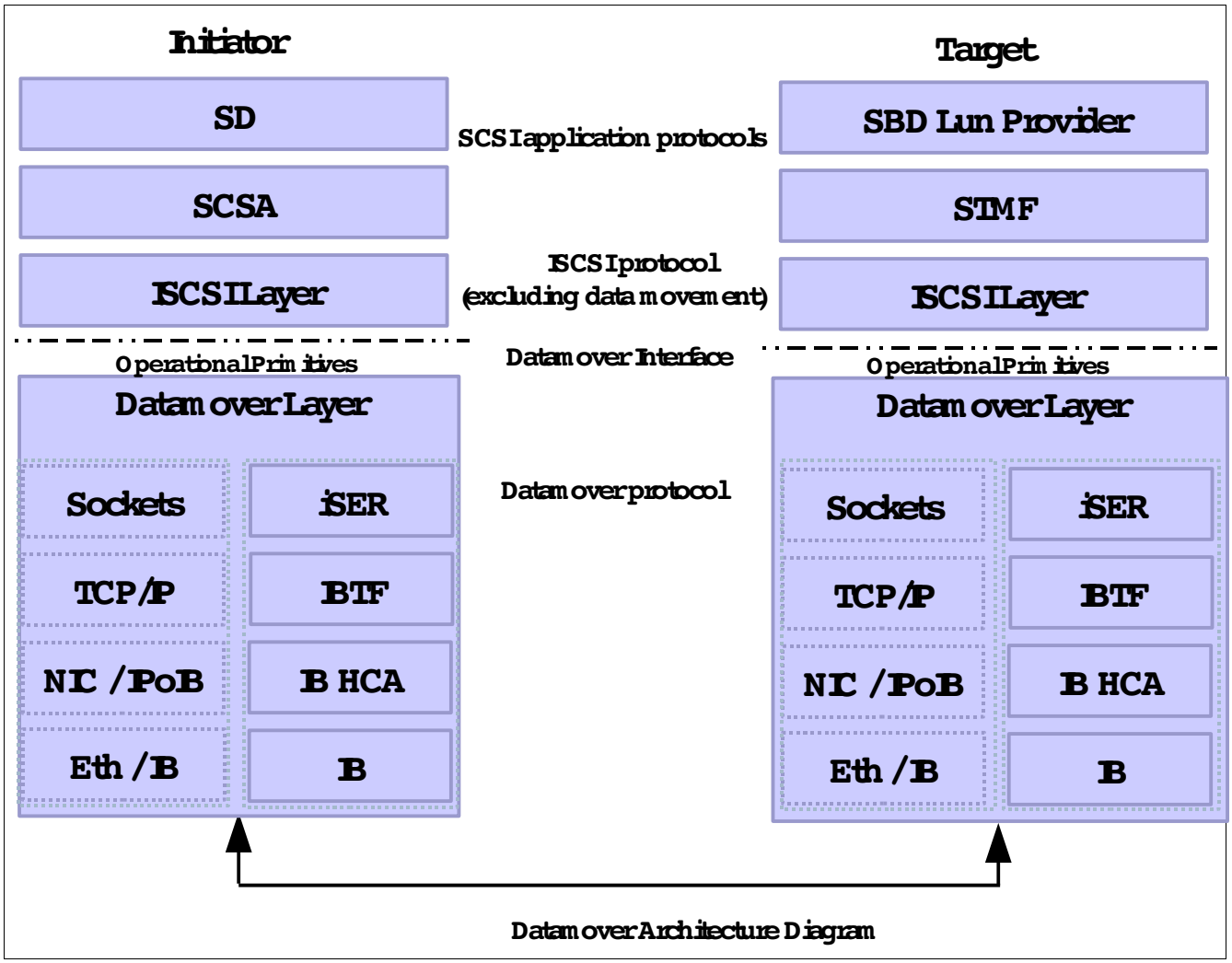
Solaris has existing iSCSI initiator and iSCSI target implementations. These implementations are socket-based and unfortunately do not include provisions to accommodate the iSER protocol. This document describes a new kernel module called iSCSI Data Mover (IDM). IDM provides a modular transport API and exports an interface to the iSCSI code (initiator and target) that abstracts the transport details. Additionally, IDM implements a pluggable transport interface which provides for future transport extensibility.

1.2 RFC Datamover Architecture

IDM is based on the Datamover Architecture described in RFC5047. The Datamover RFC defines a set of primitives that abstract data movement from the rest of the iSCSI protocol.

Datamover Primitive	Description
Send_Control	Request the outbound transfer of an iSCSI control-type PDU
Put_Data	Request the outbound transfer of data Data-in PDU
Get_Data	Request the inbound transfer of data
Allocate_Connection_Resources	Request the allocation of all transport-specific connection resources
Deallocate_Connection_Resources	Request the deallocation of all transport-specific connection resources
Enable_Datamover	Request that a specific iSCSI connection be

	transitioned to Datamover-assisted mode
Connection_Terminate	Request that a specific connection be terminated and all associated connection resources be freed
Notice_Key_Value	Request that specific key-value pairs be noted
Deallocate_Task_Resources	Request the deallocation of all task resources



Datamover Architecture Diagram

Since the Datamover RFC describes an architecture not a protocol there is no need to rigidly comply to the details of the RFC – the only method to determine "compliance" would be visual inspection of the source code. IDM follows the general philosophy of Datamover and is free to add or remove primitives as needed. Where there is a direct parallel between an IDM API and a Datamover API this document will highlight that relationship.

1.3 Design Goals

- IDM handles most of the connection details including connection setup, teardown and tracking the current state of the connection
- iSCSI target and iSCSI initiator will share a single IDM kernel module
- All IDM transport implementations will also be shared
- IDM will manage all data transfers
- IDM will not require iSCSI knowledge outside the scope of the connection (e.g. The iSCSI session will be managed by the initiator and target)

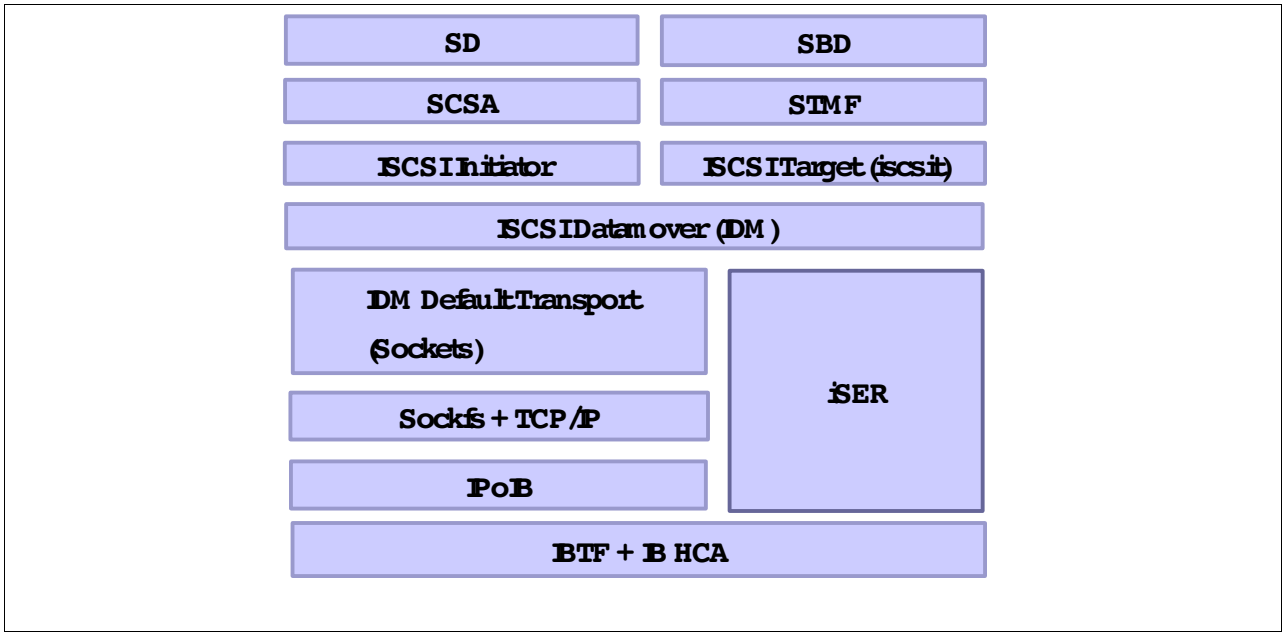
2 Architecture

2.1 Overview

iSCSI Data Mover (IDM) is a kernel module that enables one or more IDM client modules (either an iSCSI initiator, iSCSI target, or both) to use the set of available IDM transport modules (for example, iSER). IDM provides connection oriented services including

- Connection setup/teardown
- Connection-related event notifications
- iSCSI Connection State machine
- iSCSI task services
- Buffer services
- Transmission and reception of iSCSI PDUs
- Validation of iSCSI PDUs against the current connection state

2.2 Component Block Diagram



2.3 Component Descriptions

SD	Solaris SCSA class driver for SCSI block (T10 SBC) devices. Used on the initiator side
SCSA	Sun Common SCSI Architecture is a framework that provides the interface between SCSI class drivers and SCSI HBA drivers on the initiator side
ISCSI Initiator	A virtual HBA driver that implements the iSCSI initiator protocol and presents itself as a SCSA HBA driver.
SBD	STMF lun provider for SCSI block (T10 SBC) devices. Used on the target side.
STMF	SCSI Target Mode Framework provides the interface between SCSI lun providers and SCSI port providers on the target side.
ISCSI Target (iscsit)	STMF port provider implementing the iSCSI target protocol
ISCSI Datamover (IDM)	Implements iSCSI connection services an abstracts the underlying transport layers (IDM default transport and iSER)
IDM Default Transport	IDM transport module for sockets over TCP/IP. Maps the IDM PDU and data services onto socket operations
Sockfs + TCP/IP	Solaris kernel sockets API and TCP/IP stack
iSER	“iSCSI Extensions for RDMA “ is an IDM transport module mapping the IDM PDU and data services onto RDMA operations
IPoIB	IP transport over InfiniBand fabric. ISER connections begin as TCP/IP connections using IPoIB and IDM Default Transport and are later transitioned to iSER mode at which point IPoIB is no longer used. See the iSER design spec for more details.
IBTF + IB HCA	InfiniBand Transport Framework and Infiniband Host Channel Adapter

3 Component APIs

3.1 Connection Services

3.1.1 idm_conn_req_t

NAME

idm_conn_req_t – IDM connection request structure

SYNOPSIS

```
#include <sys/idm.h>
```

DESCRIPTION

This structure contains the required parameters to initiate a connection to a remote iSCSI target.

STRUCTURE MEMBERS

```
int             cr_domain;
int             cr_type;
int             cr_protocol;
boolean_t       cr_bound;
idm_sockaddr_t  cr_bound_addr;
idm_sockaddr_t  cr_ini_dst_addr;
idm_conn_ops_t  cr_conn_ops;
```

cr_domain	Socket domain for new connection
cr_type	Socket type for new connection
cr_protocol	Socket protocol for new connection
cr_bound	Boolean indicating whether to bind the socket to the address in “cr_bound_addr”.
cr_bound_addr	Address to use in socket bind
cr_ini_dst_addr	Destination address for connection
cr_conn_ops	A set of IDM client callbacks used by IDM to communicate with the IDM client about the new connection

SEE ALSO

idm_ini_conn_create(), idm_ini_conn_connect(), idm_ini_conn_disconnect(), idm_ini_conn_destroy(), idm_svc_req_t, idm_conn_ops_t

3.1.2 idm_ini_conn_create, idm_ini_conn_connect, idm_ini_conn_disconnect, idm_ini_conn_destroy

NAME

idm_ini_conn_create, idm_ini_conn_connect, idm_ini_conn_disconnect, idm_ini_conn_destroy – Used by an iSCSI initiator to connect to an iSCSI target service

SYNOPSIS

```
#include <sys/idm.h>
idm_status_t idm_ini_conn_create(idm_conn_req_t *cr,
      idm_conn_t **new_con);
idm_status_t idm_ini_conn_connect(idm_conn_t *ic)
void idm_ini_conn_disconnect(idm_conn_t *ic);
void idm_ini_conn_destroy(idm_conn_t *ic);
```

PARAMETERS

`cr`	Pointer to a connection request structure representing the desired connection parameters
`new_con`	Output parameter containing a pointer to the new connection context
`ic`	Pointer to a connection context

DESCRIPTION

idm_ini_conn_create

Using by an iSCSI initiator to create an iSCSI connection context to a remote node. This function will not actually initiate the connection.

idm_ini_conn_connect

Start the iSCSI connection state machine for the specified connection context. The connection state machine will attempt to connect to the remote node configured in the call to idm_ini_conn_create.

idm_ini_conn_disconnect

Perform a disconnect on the specified connection by generating an event to the connection state machine. Asynchronous after the state machine has gone through the necessary steps and the connection is idle the client will be notified via the “client_notify” callback.

idm_ini_conn_destroy

Release connection resources. This is the complement of idm_ini_conn_create.

RETURN_VALUES

IDM_STATUS_SUCCESS if successful, any other value indicates failure

SEE ALSO

idm_tgt_svc_create, idm_tgt_svc_connect, idm_tgt_svc_disconnect, idm_tgt_svc_destroy

3.1.3 idm_svc_req_t

NAME

idm_svc_req_t – Request the creation of an iSCSI TCP/IP service

SYNOPSIS

```
#include <sys/idm.h>
```

DESCRIPTION

This structure contains parameters defining the creation of a listening socket connection. The iSCSI target will use this to request the creation of an iSCSI TCP/IP service instance

STRUCTURE MEMBERS

```
uint16_t          sr_port;
ldi_ident_t       sr_li;
idm_conn_ops_t    sr_conn_ops;
```

sr_port	TCP/IP port number on which to listen
sr_li	LDI handle of initiator or target
sr_conn_ops	A set of IDM client callbacks used by IDM to communicate with the IDM client about new connections

SEE ALSO

idm_tgt_svc_create, idm_tgt_svc_connect, idm_tgt_svc_disconnect, idm_tgt_svc_destroy

3.1.4 idm_tgt_svc_create, idm_tgt_svc_connect, idm_tgt_svc_disconnect, idm_tgt_svc_destroy

NAME

idm_tgt_svc_create, idm_tgt_svc_connect, idm_tgt_svc_disconnect, idm_tgt_svc_destroy – Used by the target to manage iSCSI TCP/IP service instances

SYNOPSIS

```
#include <sys/idm.h>

idm_status_t idm_tgt_svc_create(idm_svc_req_t *sr, idm_svc_t
      **new_svc);

idm_status_t idm_tgt_svc_connect(idm_svc_t *is);

void idm_tgt_svc_disconnect(idm_svc_t *is);

void idm_tgt_svc_destroy(idm_svc_t *is);
```

PARAMETERS

`sr`	Pointer to a “server” connection request structure representing the desired listening socket
`new_svc`	Output parameter containing a pointer to the new server context
`is`	Pointer to a server context

DESCRIPTION

idm_tgt_svc_create

Used by an iSCSI target to create an iSCSI TCP/IP service context

idm_tgt_svc_connect

Used by an iSCSI target to enable an iSCSI TCP/IP service context. Upon return the TCP/IP service is listening for connections on the requested port.

idm_tgt_svc_disconnect

Stop listening for connections on the associated iSCSI TCP/IP service

idm_tgt_svc_destroy

Release any resources for the associated iSCSI TCP/IP service

RETURN_VALUES

IDM_STATUS_SUCCESS indicates success, any other value indicates failure

SEE ALSO

idm_ini_conn_create, idm_ini_conn_connect, idm_ini_conn_disconnect, idm_ini_conn_destroy

3.1.5 idm_notice_key_values

NAME

idm_notice_key_values – key value pair assembly and negotiation for an iSCSI connection

SYNOPSIS

```
#include <sys/idm.h>
idm_status_t  idm_notice_key_values(idm_conn_t *ic,
   nv_list_t *request_nvl, nv_list_t *response_nvl, nv_list_t
  *negotiated_nvl);
```

PARAMETERS

`ic`	Pointer to a connection context
`request_nvl`	Nvlist containing Login Request key value pairs
`response_nvl`	Nvlist containing Login Response key value pairs
`negotiated_nvl`	Nvlist containing negotiated login key value pairs

DESCRIPTION

The iSCSI layer calls this routine to notify the IDM layer of a set of key value pair lists for use in this connection's login negotiations.

It is used by the iSCSI initiator to request that the IDM and lower (transport) layers set preferred key value pairs in the request_nvl. Once returned, the iSCSI initiator will add its own key value pairs and assemble the Login Request PDU.

When a Login Request is received on the target-side, the iSCSI layer will populate a request_nvl and pass it to the IDM and transport layers via this mechanism. Each layer will then process it (see transport_notice_key_values()) and return the three lists: request_nvl, response_nvl and negotiated_nvl. The request_nvl contains all key value pairs from the Login Request that are handled at the iSCSI layer, and need to be parsed. If there are unknown key value pairs in the request_nvl at this point, it defines a protocol error. If a key value pair is agreeable, it is moved into the negotiated_nvl. If it needs to be selected from a field or changed, it is moved to the response_nvl and copied onto the negotiated_nvl.

Once complete, the iSCSI layer has a full picture of the negotiated key value pairs at this point in the login negotiation in negotiated_nvl, and a list of any key value pairs that need to be sent back to the initiator in response_nvl. The iSCSI target can then assemble and send its Login Response.

NOTES

This routine serves as an implementation of the Notice_Key_Values DA primitive described in [DA], but goes beyond the RFC's requirement, adding the functionality of the lower layers modifying the key value pairs if desired.

SEE ALSO

idm_conn_t, nv_list_t

3.2 Buffer Services

3.2.1 idm_buf_t

NAME

idm_buf_t – IDM buffer structure

SYNOPSIS

```
#include <sys/idm.h>
```

DESCRIPTION

This structure contains the I/O buffer that is affiliated with a SCSI command.

STRUCTURE MEMBERS

```
idm_conn_t      *idb_ic;
void            *idb_buf;
uint64_t        idb_buflen;
size_t          idb_bufoffset;
uint64_t        idb_xferlen;
void            *idb_mr_handle;
void            (*idb_callback)(idm_buf_t *buf);
Void            *idb_cb_arg;
```

idb_ic	The associated connection handle
idb_buf	A buffer to hold the I/O data
idb_buflen	length of the buffer
idb_bufoffset	offset into the total data transfer
idb_xferlen	length of the total data transfer
idb_mr_handle	A memory region handle used by iSER only
idb_callback	An iSCSI target callback used by IDM to notify completing the retrieval of the data or sending the data

`idb_cb_arg`	Private data

SEE ALSO

idm_buf_alloc, idm_buf_setup, idm_buf_teardown, idm_buf_free, idm_buf_bind

3.2.2 idm_buf_alloc, idm_buf_free, idm_buf_bind_in, idm_buf_bind_out, idm_buf_unbind

3.2.3 NAME

idm_buf_alloc, idm_buf_free, idm_buf_bind_in, idm_buf_bind_out, idm_buf_unbind – Buffer operations

SYNOPSIS

```
#include <sys/idm.h>
idm_buf_t  *idm_buf_alloc(idm_conn_t *ic, void *bufptr,
           uint64_t buflen);
void       idm_buf_free(idm_buf_t *idb);
void       idm_buf_bind_in(idm_task_t *idt, idm_buf_t *idb);
void       idm_buf_bind_out(idm_task_t *idt, idm_buf_t *idb);
void       idm_buf_unbind(idm_buf_t *idb);
```

PARAMETERS

`ic`	Pointer to the idm_conn_t structure representing the iSCSI connection associated with the IDM connection
`bufptr`	Pointer to a memory region that the idm_buf_t will represent. A value of NULL indicates that memory should be allocated.
`buflen`	length of the data buffer
`idb`	Pointer to a buffer context allocated with idm_buf_alloc()
`idt`	Pointer to a task context, used to affiliate a buffer to a SCSI command

DESCRIPTION

idm_buf_alloc

Allocates memory of the specified size for the I/O buffer meant for the Data-Out/Data-In for the command. If bufptr == NULL then idm_buf_alloc will allocate a memory region of size buflen.

idm_buf_free

Release a buffer handle along with the associated buffer that was allocated with idm_buf_alloc().

idm_buf_bind_in, idm_buf_bind_out

Associate a buffer with a task. The iSCSI initiator calls these functions to associate the SCSI transfer buffers with the task and will only call this once per transfer direction.

idm_buf_unbind

Breaks the association of task and buffer that was previously created by calling idm_buf_bind_in or idm_buf_bind_out.

NOTES

iSCSI target does not need to call idm_buf_bind_in/idm_buf_bind_out because the calls to idm_buf_tx_to_ini and idm_buf_rx_from_ini implicitly associate a buffer with a task.

RETURN_VALUES

A non null value of idm_buf_t indicates success, NULL indicates failure.

SEE ALSO

idm_task_t, idm_conn_t, idm_buf_tx_to_ini, idm_buf_rx_from_ini

3.3 Task Services

3.3.1 idm_task_t

NAME

idm_task_t – IDM task context

SYNOPSIS

```
#include <sys/idm.h>
```

DESCRIPTION

This structure contains the attributes of a task. In IDM, each SCSI command is referred to as a task and the idm_task_t structure houses all the of the required contexts and handles for requesting a data transfer start, including the connection context and affiliated buffers.

STRUCTURE MEMBERS

```
idm_conn_t    *idt_ic;
boolean_t     idt_active;
void          *idt_private;
```

```
uint32_t      idt_tt;
list_t        idt_inbufv;
list_t        idt_outbufv;
```

idt_ic	Pointer to the idm_conn_t structure representing the iSCSI connection associated with the IDM connection
idt_active	A boolean to indicate that the task is active
idt_private	Pointer to store client specific task context
idt_tt	A unique task tag that the initiator can use as an ITT and the target can use as a TTT
idt_inbufv	List of input buffers from which data is read on to the transport layer
idt_outbufv	List of output buffers to which data is read from the transport layer

SEE ALSO

idm_task_alloc, idm_task_start, idm_task_done, idm_task_free, idm_task_find

3.3.2 idm_task_alloc, idm_task_start, idm_task_done, idm_task_free, idm_task_find

NAME

idm_task_alloc, idm_task_start, idm_task_done, idm_task_free – iSCSI task operations

SYNOPSIS

```
#include <sys/idm.h>
idm_task_t *idm_task_alloc(idm_conn_t *ic);
void       idm_task_start(idm_task_t *idt);
void       idm_task_done(idm_task_t *idt);
void       idm_task_free(idm_task_t *idt);
idm_task_t *idm_task_find(uint32_t tt);
```

PARAMETERS

ic	Pointer to the idm_conn_t structure representing the iSCSI connection associated with the IDM connection
idt	Pointer to a task context that represents the SCSI command
tt	A task tag that uniquely identifies the SCSI command in operation

DESCRIPTION

idm_task_alloc

Allocate a idm_task_t() structure. The task tag will be set to a unique value that can be used by the initiator as an ITT or by the target as a TTT.

idm_task_start

Create an association between the connection and the task.

idm_task_done

Delete an association between the connection and the task that was previously established with the call to idm_task_start.

idm_task_free

Free the idm_task_t and the resources allocated for the task.

idm_task_find

Lookup a task using the task tag

RETURN_VALUES

A non null value of idm_task_t indicates success, NULL indicates failure

SEE ALSO

idm_conn_t

3.4 PDU Services

3.4.1 idm_pdu_t

NAME

idm_pdu_t – iSCSI PDU context

SYNOPSIS

```
#include <sys/idm.h>
```

DESCRIPTION

The structure houses an iSCSI PDU.

STRUCTURE MEMBERS

```
idm_conn_t       *isp_ic;
```

```
iscsi_hdr_t       *isp_hdr;
uint32_t          isp_hdrlen;
void              *isp_data;
uint32_t          isp_datalen;
void              *isp_private;
void              (*isp_callback)(idm_pdu_t *pdu, idm_status_t status);
```

isp_ic	Pointer to the idm_conn_t structure representing the iSCSI connection associated with the IDM connection
isp_hdr	Pointer to the iSCSI header including BHS and AHS
isp_bhslen	Header length including BHS and AHS
isp_data	Data payload
isp_datalen	Data payload length
isp_private	Pointer to private data maintained by the iSCSI client
isp_callback	Function pointer called with the PDU has been processed. For transmit PDU's this is called by the transport module when the PDU has been submitted to the underlying physical transport for transmission and for receive PDU's this is called by the client when the received PDU has been processed.
isp_status	Status of the PDU

SEE ALSO

idm_pdu_alloc, idm_pdu_free, idm_pdu_init, idm_pdu_init_hdr, idm_pdu_int_data, idm_pdu_tx, idm_pdu_complete

3.4.2 idm_pdu_alloc, idm_pdu_free, idm_pdu_init, idm_pdu_init_hdr, idm_pdu_init_data, idm_pdu_tx, idm_pdu_complete

NAME

idm_pdu_alloc, idm_pdu_free, idm_pdu_init, idm_pdu_init_hdr, idm_pdu_init_data, idm_pdu_tx, idm_pdu_complete

SYNOPSIS

```
#include <sys/idm.h>
idm_pdu_t  *idm_pdu_alloc(uint_t hdrlen, uint_t datalen);
void       idm_pdu_free(idm_pdu_t *pdu);
void       idm_pdu_init(idm_pdu_t *pdu, idm_conn_t *ic,
```

```
                void *private, void (*isp_callback)(idm_pdu_t *pdu,
                idm_status_t status);
   void         idm_pdu_init_hdr(idm_pdu_t *pdu, uint8_t *hdr,
                uint_t hdrlen);
   void         idm_pdu_init_data(idm_pdu_t *pdu, uint8_t *data,
                uint_t datalen);
   void         idm_pdu_tx(idm_pdu_t *pdu);
   void         idm_pdu_complete(idm_pdu_t *pdu, idm_status_t status);
```

PARAMETERS

`ic`	Pointer to the idm_conn_t structure representing the iSCSI connection associated with the IDM connection
`pdu`	Pointer to the PDU context that houses an iSCSI PDU
`hdr`	Pointer to the start of the iSCSI BHS
`hdrlen`	Length of the iSCSI header including BHS and AHS
`data`	Pointer to payload data
`datalen`	Length of payload data
`status`	Status of the PDU transfer

DESCRIPTION

idm_pdu_alloc

Allocates a PDU along with a memory for header and data. The header memory region will be hdrlen bytes long and the data memory region will be datalen bytes long.

idm_pdu_free

Free the pdu, header memory region and data memory region allocated previously allocated through the call to idm_pdu_alloc(). This function will only free resources allocated via idm_pdu_alloc – if the PDU is modified after allocation to point to alternate header or data memory regions those memory regions will not be freed.

idm_pdu_tx

This is IDM's implementation of the 'Send_Control' operational primitive. This function is invoked by an initiator iSCSI layer requesting the transfer of a iSCSI command PDU or a target iSCSI layer requesting the transfer of a iSCSI response PDU. The PDU will be transmitted as-is by the local Datamover layer to the peer iSCSI layer in the remote iSCSI node.

idm_pdu_complete

This is the completion callback with the status. For transmit PDU's this is called by the transport module when the PDU has been submitted to the underlying physical transport for transmission and for receive PDU's this is called by the client when the received PDU has been processed.

RETURN_VALUES

A non-null value of idm_pdu_t from idm_pdu_alloc() indicates success, NULL indicates failure.

SEE ALSO

idm_conn_t, idm_pdu_t

3.5 Data Transfer Services

3.5.1 idm_buf_tx_to_ini, idm_buf_rx_from_ini

NAME

idm_buf_tx_to_ini, idm_buf_rx_from_ini

SYNOPSIS

```
#include <sys/idm.h>
idm_status_t     idm_buf_tx_to_ini(idm_task_t *idt, idm_buf_t *idb,
               uint32_t offset, uint32_t xfer_length,
               void (*idb_buf_cb)(idm_buf_t *idb, void *cb_arg,
               idm_status_t status), void *cb_arg);
idm_status_t     idm_buf_rx_from_ini(idm_task_t *idt, idm_buf_t *idb,
               uint32_t offset, uint32_t xfer_length,
               void (*idb_buf_cb)(idm_buf_t *idb, void *cb_arg,
               idm_status_t status), void *cb_arg);
```

PARAMETERS

idt	Pointer to a task context that represents the SCSI command
idb	Pointer to a buffer context that is affiliated with the task
offset	Offset for transfer within the buffer represented by idb
xfer_length	Desired transfer length within the buffer represented by idb
idb_buf_cb	Called when the transfer has completed
cb_arg	Argument passed to the callback

DESCRIPTION

idm_buf_tx_to_ini

This is IDM's implementation of the 'Put_Data' operational primitive. This function is invoked by a target iSCSI layer to request its local Datamover layer to transmit the Data-In PDU to the peer iSCSI

layer on the remote iSCSI node. This data transfer takes place transparently to the remote iSCSI layer, i.e. without its participation. Using sockets, IDM would implement the data transfer by segmenting the data buffer into appropriately sized iSCSI PDUs and transmitting them to the initiator. iSER would perform the transfer using RDMA write.

idm_buf_rx_from_ini

This is IDM's implementation of the 'Get_Data' operational primitive. This function is invoked by a target iSCSI layer to request its local Datamover layer to retrieve certain data identified by the R2T PDU from the peer iSCSI layer on the remote node. When an iSCSI node sends an R2T PDU to its local Datamover layer, the local and remote Datamover layers transparently bring about the data transfer requested by the R2T PDU, without the participation of the iSCSI layers. On iSER, the data transfer is performed using RDMA read.

Both idm_buf_tx_to_ini and idm_buf_rx_from_ini create an association between the buffer and task. This association is removed when the buffer callback is called indicating the transfer is complete.

RETURN_VALUES

IDM_STATUS_SUCCESS indicates success, any other value indicates failure

SEE ALSO

idm_task_t, idm_buf_t

3.6 Connection Operations Vector

3.6.1 idm_conn_ops_t

NAME

idm_conn_ops_t – Ops vector for IDM client (ISCSI initiator or target)

SYNOPSIS

```
#include <sys/idm.h>
```

DESCRIPTION

The iSCSI initiator or target registers an ops vector that IDM uses to provide information about connection related events.

STRUCTURE MEMBERS

```
void          (*client_rx_scsi_cmd)(idm_conn_t *ic, idm_pdu_t *pdu);
void          (*client_rx_scsi_rsp)(idm_conn_t *ic, idm_pdu_t *pdu);
void          (*client_rx_misc)(idm_conn_t *ic, idm_pdu_t *pdu);
```

`void`	`(*client_client_notify)(idm_conn_t *ic, idm_client_notify_t cn, uintptr_t data);`
client_rx_scsi_cmd	Called when IDM receives an iSCSI “SCSI Command” PDU (opcode=0x01)
client_rx_scsi_rsp	Called when IDM receives an iSCSI “SCSI Response” PDU (opcode=0x21)
client_rx_misc	Called when IDM receives an iSCSI PDU except for “SCSI Command” and “SCSI Response.
client_notify	Called to report various connection-related events including connection state changes

SEE ALSO

client_rx_scsi_cmd, client_rx_scsi_rsp, client_rx_misc, client_notify

3.6.2 client_rx_scsi_cmd, client_rx_scsi_rsp, client_rx_misc

NAME

client_rx_scsi_cmd, client_rx_scsi_rsp, client_rx_misc – Process a received iSCSI PDU

SYNOPSIS

```
#include <sys/idm.h>
void client_rx_scsi_cmd(idm_conn_t *ic, idm_pdu_t *pdu);
void client_rx_scsi_rsp(idm_conn_t *ic, idm_pdu_t *pdu);
void client_rx_misc(idm_conn_t *ic, idm_pdu_t *pdu);
```

PARAMETERS

`ic`	Pointer to a connection context
`pdu`	Pointer to an idm_pdu_t structure representing the received iSCSI PDU

DESCRIPTION

IDM calls these functions after an iSCSI PDU has been received off the connection and validated. To avoid requiring the client to de-multiplex the PDU opcode on performance sensitive commands, IDM will call client_rx_scsi_cmd for all PDU's with opcode 0x01 (SCSI Command) and client_rx_scsi_rsp for all PDU's with opcode 0x21. IDM will call client_rx_misc() for any other types of iSCSI PDU's.

SEE ALSO

idm_conn_ops_t

3.6.3 client_notify

NAME

client_notify – Notify client of connection-related events

SYNOPSIS

```
#include <sys/idm.h>

idm_status_t client_notify(idm_conn_t *ic, idm_client_notify_cb_t cn,
                                    uintptr_t data);
```

PARAMETERS

ic	Pointer to a connection context

cn	Notification type:
	`typedef enum {`
	`CN_CONNECT_ACCEPT = 1, /* Target only */`
	`CN_FFP_ENABLED, /* Full feature phase enabled */`
	`CN_FFP_DISABLED, /* Left full feature phase */`
	`CN_CONNECT_LOST, /* Connection failure */`
	`CN_CONNECT_DESTROY /* Connection destroyed */`
	`} idm_client_notify_t;`

data	Data associated with the notification.

DESCRIPTION

IDM calls the client entry point when the state of the iSCSI connection changes. The client must handle the notification without blocking since it will be called in the context of the IDM connection state machine.

RETURN VALUES

IDM_STATUS_SUCCESS	Notification accepted
IDM_STATUS_REJECT	Notification rejected (this will shutdown the associated connection)

SEE ALSO

idm_conn_ops_t

3.7 Transport Operations Vector

3.7.1 Overview

The IDM module provides the Solaris iSCSI implementation with the ability to transparently take advantage of not only the native Sockets transport, but also a Datamover Protocol transport as well. In order to facilitate the use of multiple Datamover transports, IDM will implement a pluggable transport layer that allows IDM to utilize other Datamover Protocol implementations as they become available, with no changes to the iSCSI layer, and minimal changes to IDM to add a transport handle for lookup (see below).

Kernel modules that implement a Datamover Protocol will implement the transport-specific routines described in this section and register with the IDM (via idm_register()). Via this method, the IDM will provide a transparent mechanism for iSCS I to take advantage of any qualified Datamover Protocol available in the future. It also has the benefit of completely abstracting the IDM transport layer from the iSCSI layer.

Initially the project will provide the iSER over InfiniBand transport (iSER-IB) to iSCSI without any specific knowledge of the underlying iSER protocol itself; all usage will be via the IDM layer, and therefore will be abstracted from iSCSI directly. This could be done by implementing the iSER-IB functionality in the IDM module itself, but this makes future utilization of other Datamover Protocols as a transport cumbersome, and would require modification of the IDM module. Rather, this implementation will use the pluggable transport layer described in this section to provide future extensibility in the transport layer. Additional Datamover Protocol implementations, such as iSER over iWarp, may be implemented and deployed without altering the IDM and upper iSCSI software layers.

3.7.2 Usage in IDM

Once a Datamover Protocol module is registered with IDM via idm_register(), it is referred to as an IDM transport, and is represented by an IDM transport handle. This handle (idm_transport_t) encodes data regarding the transport, including its capabilities and an operations vector for IDM's use (see idm_transport_ops_t below).

```
typedef enum {
        IDM_TRANSPORT_TYPE_ISER,
        IDM_TRANSPORT_NUM_TYPES
} idm_transport_type_t;


typedef struct {
        idm_transport_type_t            type;
        char                            device_path[IDM_PATHLEN_MAX];
        ldi_handle_t                    ldi_hdl;
        idm_transport_impl_ops_t        *it_ops;
        idm_transport_caps_t            *it_caps;
} idm_transport_t;
```

During IDM initialization, a list of IDM transport handles is initialized. This requires IDM to have pre-existing knowledge of all supported Datamover Protocol implementations, as each implementation will have specific idm_transport_t data. This allows IDM to selectively choose to initialize and use any given Datamover Protocol implementation via the use of the LDI (Layered Driver Interface) mechanism in Solaris. Specifically in this respect, IDM has pre-existing knowledge of a Datamover Protocol driver's device path, for opening the driver.

Note that we do not place the IDM transport handle for the Sockets transport in the list of IDM transports, as it is the native transport of IDM and we know it must exist for us to move forward with IDM operation. We instead will keep the native IDM transport handle in the global IDM namespace.

```
idm_transport_list idm_transport_t[IDM_TRANSPORT_NUM_TYPES] = {
        /* iSER transport handle */
        {IDM_TRANSPORT_TYPE_ISER,
        "/devices/ib/iser@0:iser",
        NULL,
        NULL,
        NULL,
        NULL}
};
```

This list represents IDM's knowledge of all supported IDM transport driver modules. Therefore, as new driver modules become available, this list must be added to in order for IDM to use them.

Once this list is initialized, IDM can reference all supported Datamover Protocol implementations and attempt to use the Datamover Protocol-assisted mode on any new session.

An additional benefit of attempting to detect and use transports in this way is that we will indirectly support hotplug of new underlying transports, as we will continue to refresh our picture of available Datamover Protocols. The method described in the pseudo code below can be deployed during session establishment to accomplish this.

```
given idm_conn_t *ic:
for (TYPE = 0 to IDM_NUM_TRANSPORT_TYPES) {
        transport = idm_transport_list[TYPE]
        if (transport.ldi_hdl == NULL) {
                ldi_open_by_name(transport.device_path)
                // this open will trigger the transport driver's idm_register()
                // in turn, idm_register() will populate this transports caps and ops
        }
        if ((transport.ldi_hdl != NULL) && (transport.ops->conn_is_capable())) {
                ic->transport_hdl = &transport
                break;
```

```
            }
        }
```

Note here that in the case where IDM configures a Datamover Protocol transport and discovers that it is provided via this connection (i.e. iSER-IB is capable over this IPoIB connection), we choose to use it and cease further iteration, as we will always choose to use the Datamover Protocol over the native Sockets transport. Note that if a connection provides more than one transport, we will choose to use the first available transport in the list. This will not be an issue for the initial project, or with any currently available Datamover Protocols, but may be explored in the future if required.

Implicit in the pseudo code above is a requirement of an IDM transport driver to implement all applicable transport ops and call idm_register() in it's open() routine. The refreshed IDM transport handle returned to IDM via idm_register() replaces the current entry stored on the list in the IDM state structure.

A note for clarity: the transport_* routines below are implemented by the transport driver and invoked via the idm_transport_impl_ops_t structure by IDM. Under no circumstances should the iSCSI layer attempt to invoke these routines directly. If iSCSI needs access to any of these routines, a wrapper IDM routine should be implemented (see idm_notice_key_values() / transport_notice_key_values()).

3.7.3 idm_transport_ops_t

NAME

idm_transport_ops_t – Ops vector for IDM transport drivers

SYNOPSIS

```
#include <sys/idm.h>
```

DESCRIPTION

The IDM transport driver registers an ops vector that IDM uses to invoke transport-specific implementations of IDM routines.

STRUCTURE MEMBERS

```
void            (*transport_tx)(idm_conn_t *ic, idm_pdu_t *pdu);
idm_status_t    (*transport_buf_tx_to_ini)(idm_task_t *idt,
                    idm_buf_t *idb);
void            (*transport_rx_datain)(idm_conn_t *ic, idm_pdu_t *pdu);
idm_status_t    (*transport_buf_rx_from_ini)(idm_task_t *idt,
                    idm_buf_t *idb);
void            (*transport_rx_rtt)(idm_conn_t *ic, idm_pdu_t *pdu);
void            (*transport_rx_dataout)(idm_conn_t *ic, idm_pdu_t *pdu);
idm_status_t    (*transport_alloc_conn_rsrc)(idm_conn_t *ic);
```

```
idm_status_t      (*transport_free_conn_rsrc)(idm_conn_t *ic);
idm_status_t      (*transport_enable_datamover)(idm_conn_t *ic);
idm_status_t      (*transport_conn_terminate)(idm_conn_t *ic);
idm_status_t      (*transport_free_task_rsrcs)(idm_task_t *it,
                      idm_conn_t *ic);
idm_status_t      (*transport_notice_key_values)(idm_conn_t *ic,
                      nv_list_t *request_nvl, nv_list_t *response_nvl,
                  nv_list_t *negotiated_nvl);
idm_status_t      (*transport_conn_is_capable)(idm_conn_t *ic,
                      idm_transport_caps_t *caps);
idm_status_t      (*transport_buf_setup)(idm_buf_t *idb);
void              (*transport_buf_teardown)(idm_buf_t *idb);
```

SEE ALSO

transport_tx, transport_buf_tx_to_ini, transport_rx_datain, transport_buf_rx_from_ini, transport_rx_rtt, transport_rx_dataout, transport_alloc_conn_rsrc, transport_free_conn_rsrc, transport_enable_datamover, transport_conn_terminate, transport_free_task_rsrcs, transport_validate_keywords, transport_conn_is_capable, transport_buf_alloc, transport_buf_free, idm_conn_t, idm_task_t, idm_buf_t, idm_pdu_t, idm_transport_caps_t, nvlist_t

3.7.4 idm_transport_caps_t

NAME

idm_transport_caps_t – capabilities of a transport

SYNOPSIS

```
#include <sys/idm.h>
```

DESCRIPTION

The IDM transport driver registers an ops vector that IDM uses to invoke transport-specific implementations of IDM routines.

```
uint32_t          private_flags;  /* for use by the transport layer */
boolean_t         rcap;           /* transport is RDMA Capable */
boolean_t         native;         /* transport is IDM native */
```

```
                    /* more to follow */
```

SEE ALSO

3.7.5 transport_tx

NAME

transport_tx – Send a control-type PDU

SYNOPSIS

```
#include <sys/idm.h>
void     transport_tx(idm_conn_t *ic, idm_pdu_t *pdu);
```

PARAMETERS

`ic`	Pointer to a connection context
`pdu`	Pointer to an idm_pdu_t structure representing the Control-type PDU

DESCRIPTION

The IDM layer calls this function from either the intiator- or target-side to send a Control-type PDU via the transport layer. IDM must fully assemble the PDU prior and ensure that the connection is active and fully configured prior to calling this routine.

NOTES

This routine implements the Send_Control DA primitive described in [DA].

SEE ALSO

idm_transport_ops_t, idm_conn_t, idm_pdu_t

3.7.6 transport_rx_datain, transport_rx_rtt

NAME

transport_rx_datain, transport_rx_rtt– Initiator transport data primitives

SYNOPSIS

```
#include <sys/idm.h>
void     transport_rx_datain(idm_conn_t *ic, idm_pdu_t *pdu);
void     transport_rx_rtt(idm_conn_t *ic, idm_pdu_t *pdu);
```

PARAMETERS

`ic`	Pointer to a connection context
`pdu`	Pointer to an idm_pdu_t structure representing the received Data-in or R2T PDU

DESCRIPTION

The IDM layer calls these functions on the initiator-side to handle receipt of inbound Data-type PDUs. The transport driver layer on the initiator gets a Data-type PDU and notifies the IDM layer so that the PDU can be validated against the current connection state. If IDM can service the PDU, then the IDM layer invokes these routines to call back into the transport driver to handle either managing the inbound data (in the Data-in case) or sending data back to the target (in the R2T case).

SEE ALSO

idm_conn_ops_t, idm_conn_t, idm_pdu_t

3.7.7 transport_buf_tx_to_ini, transport_buf_rx_from_ini

NAME

transport_buf_tx_to_ini, transport_buf_rx_from_ini – Target transport data primitives

SYNOPSIS

```
#include <sys/idm.h>
idm_status_t   transport_buf_tx_to_ini(idm_task_t *idt, idm_buf_t *idb);
idm_status_t   transport_buf_rx_from_ini(idm_task_t *idt, idm_buf_t *idb);
```

PARAMETERS

`idt`	Pointer to a task context
`idb`	Pointer to a buf context

DESCRIPTION

IDM calls these functions on the target-side to initiate data transfer. A task tag is passed into the transport layer via this routine, which houses all of the required contexts and handles for requesting a data transfer start, including the connection context and affiliated buffers.

NOTES

The transport_buf_tx_to_ini() routine implements the Put_Data DA primitive described in [DA]. The transport_buf_rx_from_ini() routine implements the Get_Data DA primitive described in [DA].

SEE ALSO

idm_conn_ops_t, idm_task_t, idm_buf_t

3.7.8 transport_rx_dataout

NAME

transport_rx_dataout – Target transport data-in handler

SYNOPSIS

```
#include <sys/idm.h>
void          transport_rx_dataout(idm_conn_t *ic, idm_pdu_t *pdu);
```

PARAMETERS

`ic`	Pointer to a connection context
`pdu`	Pointer to an idm_pdu_t structure representing the received Data-in or R2T PDU

DESCRIPTION

The IDM layer calls this function on the target-side to handle receipt of inbound Data-in PDUs. The transport driver layer on the target gets a Data-in PDU and alerts the IDM layer, so that the PDU can be validated against the current connection state. If IDM can service the PDU, then the IDM layer invokes this routine to call back into the transport driver to handle managing the inbound data.

SEE ALSO

idm_conn_ops_t, idm_conn_t, idm_pdu_t

3.7.9 transport_alloc_conn_rsrc, transport_free_conn_rsrc

NAME

transport_alloc_conn_rsrc – Transport driver resource allocation and free routines

SYNOPSIS

```
#include <sys/idm.h>
idm_status_t  transport_alloc_conn_rsrc(idm_conn_t *ic);
idm_status_t  transport_free_conn_rsrc(idm_conn_t *ic);
```

PARAMETERS

`ic`	Pointer to a connection context

DESCRIPTION

The IDM layer calls these routines on either the initiator- or target-side to manage the connection's transport-specific resources.

The transport_alloc_conn_rsrc() routine will allocate all necessary resources for transitioning this connection from native-mode transport to this datamover transport. The transport driver is responsible for then allocating any resources directly required for at least transitioning the connnection reference in ic, but may optionally allocate additional resources.

The transport_free_conn_rsrc() routine frees all transport-specific resources bound to the connection referenced in ic.

NOTES

The transport_alloc_conn_rsrc() routine implements the Allocate_Connection_Resources DA primitive described in [DA]. The transport_free_conn_rsrc() routine implements the Deallocate_Connection_Resources DA primitive described in [DA].

SEE ALSO

idm_conn_ops_t, idm_conn_t

3.7.10 transport_enable_datamover, transport_conn_terminate

NAME

transport_enable_datamover, transport_conn_terminate – Transport driver enable/disable

SYNOPSIS

```
#include <sys/idm.h>
idm_status_t   transport_enable_datamover(idm_conn_t *ic);
idm_status_t   transport_conn_terminate(idm_conn_t *ic);
```

PARAMETERS

`ic`	Pointer to a connection context

DESCRIPTION

The IDM layer calls these routines on either the initiator- or target-side to control transition of a connection to and from the transport.

The transport_enable_datamover() routine transitions, or graduates, a connection from native-mode to the transport-mode connection. For example, if the transport were iSER, upon successful completion of this routine, the connection referred to by ic is in iSER-assisted mode. This routine implements the Enable_Datamover DA primitive described in [DA].

The transport_conn_terminate() routine transitions a datamover-enabled connection referred to by ic back to native-mode transport, and frees all transport-specific connection resources for ic. Upon successful completion of this routine, the IDM layer **must** transition the connection back to native-mode or another transport.

NOTES

The transport_enable_datamover() routine implements the Enable_Datamover DA primitive described in [DA]. The transport_conn_terminate() routine implements the Connection_Terminate DA primitive described in [DA].

SEE ALSO

idm_conn_ops_t, idm_conn_t

3.7.11 transport_free_task_rsrcs

NAME

transport_free_task_rsrc – Transport driver task resource cleanup

SYNOPSIS

```
#include <sys/idm.h>
idm_status_t  transport_free_task_rsrcs(idm_task_t *it,
   idm_conn_t *ic);
```

PARAMETERS

`it`	Pointer to a task context
`ic`	Pointer to a connection context

DESCRIPTION

The IDM layer calls this routine to free up all task-related transport-specific resources in the case that the connection transitions to an error state or the task otherwise ends abnormally.

NOTES

This routine implements the Deallocate_Task_Resources DA primtive described in [DA].

SEE ALSO

idm_conn_ops_t, idm_task_t, idm_conn_t

3.7.12 transport_notice_key_values

NAME

transport_notice_key_values – Transport driver validation of iSCSI keywords for use in negotiation

SYNOPSIS

```
#include <sys/idm.h>
idm_status_t  transport_notice_keyvalues(idm_conn_t *ic,
   nv_list_t *request_nvl, nv_list_t *response_nvl, nv_list_t
*negotiated_nvl);
```

PARAMETERS

`ic`	Pointer to a connection context

`request_nvl`	Nvlist containing Login Request key value pairs
`response_nvl`	Nvlist containing Login Response key value pairs
`negotiated_nvl`	Nvlist containing negotiated login key value pairs

DESCRIPTION

The IDM layer calls this routine to notify the transport layer of a set of key value pair lists for use in this connection's login negotiations. On the initiator-side, it is used by the transport layer to set requested key value pairs in the request_nvl, which ultimately will be assembled by the iSCSI initiator and sent in the Login Request PDU. The IDM layer on the target-side will receive the populated request_nvl from iSCSI, and will invoke this routine whith will iterate through each of the values in the request_nvl. If the key value pair is agreed upon by the transprot layer, it is moved to the negotated_nvl.. If it's value must be changed or selected from a field, it is moved to the response_nvl with it's new value, and copied to the negotiated_nvl. If it is unkown, it is skipped and passed back up to IDM, remaining in the request_nvl.

NOTES

This routine serves as an implementation of the Notice_Key_Values DA primitive described in [DA], but goes beyond the RFC's requirement, adding the functionality of the transport layer modifying the key value pairs if desired.

SEE ALSO

idm_conn_ops_t, idm_conn_t, nv_list_t

3.7.13 transport_conn_is_capable

NAME

transport_conn_is_capable – Transport driver functional probe

SYNOPSIS

```
#include <sys/idm.h>
idm_status_t  transport_conn_is_capable(idm_conn_t *ic,
   idm_transport_caps_t *caps);
```

PARAMETERS

`ic`	Pointer to a connection context
`caps`	Describes the transport capabilities

DESCRIPTION

The IDM layer calls this routine to detect whether or not the connection referred to in ic is provided over this transport driver. If the connection is over this transport, the idm_transport_caps_t structure will be populated with information regarding the transport.

SEE ALSO

idm_conn_ops_t, idm_conn_t, idm_transport_caps_t

3.7.14 transport_buf_setup, transport_buf_teardown

NAME

transport_buf_setup, transport_buf_xteardown – Transport driver buffer services

SYNOPSIS

```
#include <sys/idm.h>
idm_status_t  transport_buf_setup(idm_buf_t *idb);
void          transport_buf_free(idm_buf_t *idb);
```

PARAMETERS

`idb`	Pointer to a buf context

DESCRIPTION

The IDM layer will use these routines to request that the transport layer perform any required operation on the memory resources in the buffer referred to in idb.

The transport_buf_setup() routine will be invoked from idm_buf_alloc(), after memory is allocated (if requested) and the idm_buf_t is configured. If the transport layer has any required operations (such as Memory Registration in InfiniBand), it is to be done here. Otherwise, this routine may be left NULL.

The transport_buf_teardown() routine will be invoked from idm_buf_free() to undo any transport-specific memory operations done in transport_buf_setup(). If not required, it may be left NULL. If a transport driver populates transport_buf_setup(), it **must** populate transport_buf_teardown().

SEE ALSO

idm_conn_ops_t, idm_buf_t

4 Connection Management

4.1 Connection State Machine

IDM tracks connection state using a state machine derived from the standard iSCSI state machine described in RFC3720. Each state in the IDM state machine can be mapped to a particular RFC3720 connection state. IDM adds the following states (see state description table for details):

- Init Error (S9) – Maps to RFC3720 state S1
- In Cleanup (S10) – Maps to RFC3720 state S9 (this state actually comes from the "connection cleanup" state machine in the RFC)
- Complete (S11) – Maps to RFC3720 state S1

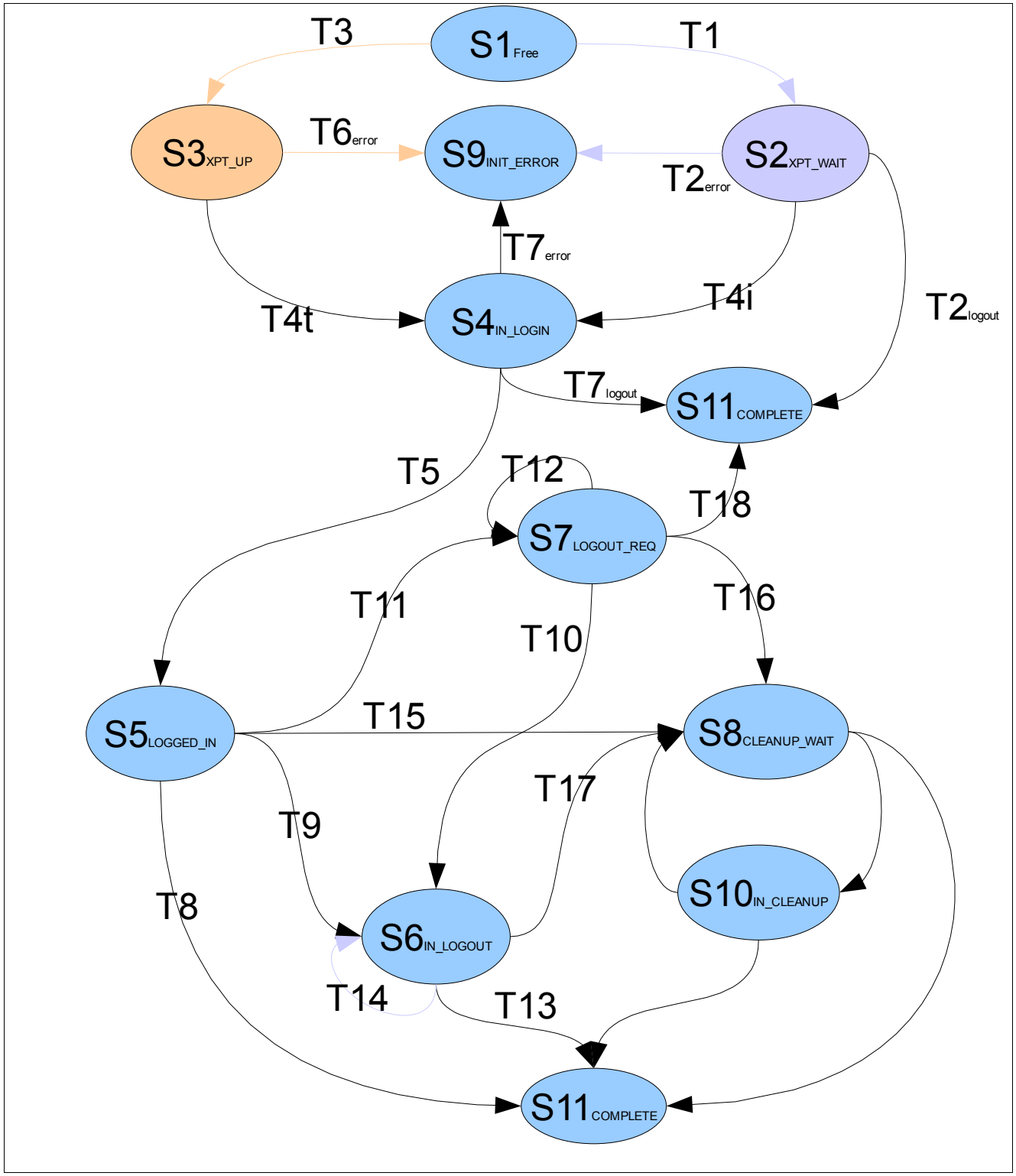
RFC3720 describes the transitions between states and there is a limited set of discrete events that causes each transition. The event description table below enumerates each of the possible events. These events form the inputs to the IDM connection state machine.

Certain state transitions in the state machine are meaningful to the IDM clients. For example a transition to "S5 Logged In" indicates that the connection is in full feature phase and can be used for processing SCSI commands. The connection state machine will notify the clients when an important state change has occurred by calling client_notify().

All iSCSI PDU's transmitted or received through IDM are validated against the current connection state. In the general case IDM will generate a state machine event corresponding to a transmit or receive request and the state machine itself determines the ultimate disposition of the PDU. For example, if a connection associated with an iSCSI target receives a "login request" PDU and the connection state is "S4 In Login" then the PDU would be forwarded on to the IDM client (target) for processing. On the other hand if the state was "S5 Logged In" then a new connection state machine event might be generated indicating that a protocol error had occurred and the login request would not be delivered to the IDM client. As a consequence the IDM clients should not need to validate the iSCSI PDU's they receive against the connection state since the IDM module has already done so.

To improve performance IDM maintains a piece of summary state indicated whether a connection is in full feature phase. SCSI-related PDU's can only be processed in full feature phase so IDM can check the summary state and if full feature phase is enabled the SCSI-related PDU's are forwarded directly to the IDM client bypassing the connection state machine.

New tasks can be started in "S5 Logged In" state and existing tasks will be processed during "S5 Logged In", "S6 In Logout" and "S7 Logout Request" states.


T3
S1 Free
T1
S3 XPT_UP
T6 error
S9 INIT_ERROR
S2 XPT_WAIT
T2 error
T7 error
T4t
S4 IN_LOGIN
T4i
T2 logout
T7 logout
S11 COMPLETE
T5
T12
T18
S7 LOGOUT_REQ
T11
T10
T16
T15
S5 LOGGED_IN
S8 CLEANUP_WAIT
T17
T9
T8
S10 IN_CLEANUP
S6 IN_LOGOUT
T14
T13
S11 COMPLETE

4.2 Connection states

	State	Description (bold italics indicate quotes from RFC3720)
S1	FREE	***State on instantiation, or after successful connection closure.***
S2	XPT_WAIT	***Waiting for a response to its transport connection establishment request.***
S3	XPT_UP	***Waiting for the Login process to commence.***
S4	IN_LOGIN	***Waiting for the Login process to conclude, possibly involving several PDU exchanges.***
S5	LOGGED_IN	***In Full Feature Phase, waiting for all internal, iSCSI, and transport events***
S7	LOGOUT_REQ	***Waiting for logout response***
S8	CLEANUP_WAIT	***Waiting for cleanup process to start***
S9	INIT_ERROR	***Terminate state, login failed to complete***
S10	IN_CLEANUP	***Waiting for cleanup process to conclude***
S11	COMPLETE	***Terminal state***

4.3 Connection Events

Event	Description (bold italics indicate quotes from RFC3720)
CONNECT_REQ	***Transport connect request was made (e.g., TCP SYNsent)***
CONNECT_FAIL	Failed to establish transport connection
CONNECT_SUCCESS	***Transport connection established, thus prompting the initiator to start the iSCSI Login.***
CONNECT_ACCEPT	***Received a valid transport connection request that establishes the transport connection***
CONNECT_REJECT	Target is unable to accept the connection
LOGIN_RCV	***Initial iSCSI Login Request was received***
LOGIN_TIMEOUT	***Target timed out waiting for an iSCSI Login***
LOGIN_SUCCESS_RCV	***The final iSCSI Login Response with a Status-Class of zero was received***
LOGIN_SUCCESS_SND	***The final iSCSI Login Request to conclude the Login Phase was received, thus prompting the target to send the final iSCSI Login Response with a Status-Class of zero***
LOGIN_FAIL_RCV	***The final iSCSI Login Response was received with a non-zero Status-Class.***
LOGIN_FAIL_SND	***The final iSCSI Login Request to conclude the Login Phase was received, prompting the target to send the final iSCSI Login Response with a non-zero***

	Status-Class
LOGOUT_THIS_CONN_RCV	***An iSCSI Logout request was received***
LOGOUT_THIS_CONN_SND	***An internal event that indicates the readiness to start the Logout process was received, thus prompting an iSCSI Logout to be sent by the initiator***
LOGOUT_THIS_CONN_SUCCESS	***An iSCSI Logout response (success) was received***
LOGOUT_THIS_CONN_FAIL	***Logout response, (failure i.e., a non-zero status) was received, or Logout timed out***
LOGOUT_OTHER_CONN_RCV	***An iSCSI Logout request was received***
LOGOUT_OTHER_CONN_SND	***An internal event that indicates the readiness to start the Logout process was received, thus prompting an iSCSI Logout to be sent by the initiator***
LOGOUT_OTHER_CONN_SUCCESS	***An iSCSI Logout response (success) was received***
LOGOUT_OTHER_CONN_FAIL	***Logout response, (failure i.e., a non-zero status) was received, or Logout timed out***
LOGOUT_SESSION_RCV	Logout “close the session” request received, possibly on another connection
LOGOUT_SESSION_SND	Logout “close the session” request sent, possibly on another connection
LOGOUT_SESSION_SUCCESS_RCV	
LOGOUT_SESSION_SUCCESS_SND	***an internal event of receiving a Logout response (success) on another connection for a "close the session"***
LOGOUT_TIMEOUT	***An internal event that indicates connection state timeout***
ASYNC_LOGOUT_RCV	***Async PDU with AsyncEvent "Request Logout" was received.***
ASYNC_LOGOUT_SND	***An internal event that requires the decommissioning of the connection is received,thus causing an Async PDU with an AsyncEvent "Request Logout" to be sent.***
ASYNC_DROP_CONN_RCV	***Async PDU with AsyncEvent "Drop connection" (for this CID).***
ASYNC_DROP_CONN_SND	***Internal emergency cleanup event was received which prompts an Async PDU with AsyncEvent "Drop connection" (for this CID)***
ASYNC_DROP_ALL_CONN_RCV	***Async PDU with AsyncEvent "Drop all connections".***
ASYNC_DROP_ALL_CONN_SND	***Internal emergency cleanup event was received which prompts an Async PDU with AsyncEvent "Drop all connections"***
TRANSPORT_FAIL	***Transport timed out, a transport reset was received, or transport disconnect indication was received.***
MISC_TX	idm_pdu_tx() was called to transmit an iSCSI PDU. Certain iSCSI PDU's may be mapped to alternate events. For example calling idm_pdu_tx() with an iSCSI logout PDU will cause the generation of LOGOUT_SND instead of MISC_TX.
TX_PROTOCOL_ERROR	idm_pdu_tx() was called to transmit an iSCSI PDU that is not allowed by the current state.
MISC_RX	idm_pdu_rx() was called to process a received iSCSI PDU. Certain iSCSI PDU's may be mapped to alternate events. For example calling idm_pdu_rx() with an

	iSCSI logout PDU will cause the generation for LOGOUT_RCV instead of MISC_TX
RX_PROTOCOL_ERROR	idm_pdu_rx() was called to process a received iSCSI PDU that is not allowed by the current state
CONN_REINSTATE_SND	***An internal event requesting the connection (or session) reinstatement was received, thus prompting a connection (or session) reinstatement Login to be sent transitioning CSM-I to state IN_LOGIN.*** CSM-I represents the new connection in the connection reinstatement process.
CONN_REINSTATE_RCV	***A connection/session reinstatement Login was received (on CSM-I) while in state XPT_UP.*** CSM-I represents the new connection in the connection reinstatement process.
CONN_REINSTATE_SUCCESS_RCV	***CSM-I reached state LOGGED_IN.*** CSM-I represents the new connection in the connection reinstatement process.
CONN_REINSTATE_SUCCESS_SND	
CONN_REINSTATE_FAIL	***CSM-I failed to reach LOGGED_IN and arrived into*** COMPLETE instead

4.4 Connection State Transitions

	From State	To State	Events (See event description table)
T1	S1Free	S2XPT_WAIT	CONNECT_REQ
T2error	S1Free	S9INIT_ERROR	CONNECT_FAIL
T2logout	S2XPT_WAIT	S11Complete	LOGOUT_CLOSE_SESSION_SUCCESS_SND, LOGOUT_CLOSE_SESSION_SUCCESS_RCV
T3	S1Free	S3XPT_UP	CONNECT_ACCEPT
T4i	S2XPT_WAIT	S4IN_LOGIN	CONNECT_SUCCESS (initiator only)
T4t	S3XPT_UP	S4IN_LOGIN	LOGIN_RCV (target only)
T5	S4IN_LOGIN	S5LOGGED_IN	LOGIN_SUCCESS
T6error	S3XPT_UP	S9INIT_ERROR	LOGIN_TIMEOUT, TRANSPORT_FAIL
T7error	S4IN_LOGIN	S9INIT_ERROR	LOGIN_FAIL, TRANSPORT_FAIL
T8	S5LOGGED_IN	S11Complete	LOGOUT_CLOSE_SESSION_SUCCESS_SND, LOGOUT_CLOSE_SESSION_SUCCESS_RCV, CONN_REINSTATE_SUCCESS
T9	S5LOGGED_IN	S6IN_LOGOUT	LOGOUT_SND, LOGOUT_RCV
T10	S7LOGOUT_REQ	S6IN_LOGOUT	LOGOUT_SND, LOGOUT_RCV

T11	S5LOGGED_IN	S7LOGOUT_REQ	ASYNC_LOGOUT_SND, ASYNC_LOGOUT_RCV
T12	S7LOGOUT_REQ	S7LOGOUT_REQ	ASYNC_LOGOUT_SND, ASYNC_LOGOUT_RCV
T13	S6IN_LOGOUT	S11Complete	LOGOUT_SUCCESS_SND, LOGOUT_SUCCESS_RCV, LOGOUT_CLOSE_SESSION_SUCCESS_SND, LOGOUT_CLOSE_SESSION_SUCCESS_RCV
T14	S6IN_LOGOUT	S6IN_LOGOUT	ASYNC_LOGOUT_RCV
T15	S5LOGGED_IN	S8CLEANUP_WAIT	TRANSPORT_FAIL, ASYNC_DROP_CONN_SND, ASYNC_DROP_CONN_RCV, ASYNC_DROP_ALL_CONN_SND, ASYNC_DROP_ALL_CONN_RCV
T16	S7LOGOUT_REQ	S8CLEANUP_WAIT	TRANSPORT_FAIL, ASYNC_DROP_CONN_SND, ASYNC_DROP_CONN_RCV, ASYNC_DROP_ALL_CONN_SND, ASYNC_DROP_ALL_CONN_RCV
T17	S6IN_LOGOUT	S8CLEANUP_WAIT	LOGOUT_FAILURE_SND, LOGOUT_FAILURE_RCV, TRANSPORT_FAIL, ASYNC_DROP_CONN_SND, ASYNC_DROP_CONN_RCV, ASYNC_DROP_ALL_CONN_SND, ASYNC_DROP_ALL_CONN_RCV
T18	S7LOGOUT_REQ	S11COMPLETE	LOGOUT_CLOSE_SESSION_SUCCESS_SND, LOGOUT_CLOSE_SESSION_SUCCESS_RCV, CONN_REINSTATE_SUCCESS
M1	S8CLEANUP_WAIT	S11COMPLETE	LOGOUT_TIMEOUT, LOGOUT_CLOSE_SESSION_SUCCESS_SND, LOGOUT_CLOSE_SESSION_SUCCESS_RCV
M2	S8_CLEANUP_WAIT	S10IN_CLEANUP	CONN_REINSTATE_CSM_I_SND, CONN_REINSTATE_CSM_I_RCV, EXPLICIT_LOGOUT_CSM_E_SND, EXPLICIT_LOGOUT_CSM_E_RCV
M3	S10IN_CLEANUP	S8CLEANUP_WAIT	CONN_REINSTATE_CSM_I_FAIL, EXPLICIT_LOGOUT_CSM_E_FAIL
M4	S10IN_CLEANUP	S11COMPLETE	CONN_REINSTATE_CSM_I_SUCCESS, EXPLICIT_LOGOUT_CSM_E_SUCCESS LOGOUT_CLOSE_SESSION_SUCCESS_SND, LOGOUT_CLOSE_SESSION_SUCCESS_RCV

5 References

Internet Small Computer Systems Interface (iSCSI)

DA: Datamover Architecture for Internal Small Computer System Interface

Internet Small Computer System Interface (iSCSI) Extensions for Remote Direct Memory Access (RDMA)