Solaris

SOSD Driver and API

Design

Version 0.6.2
May 2008

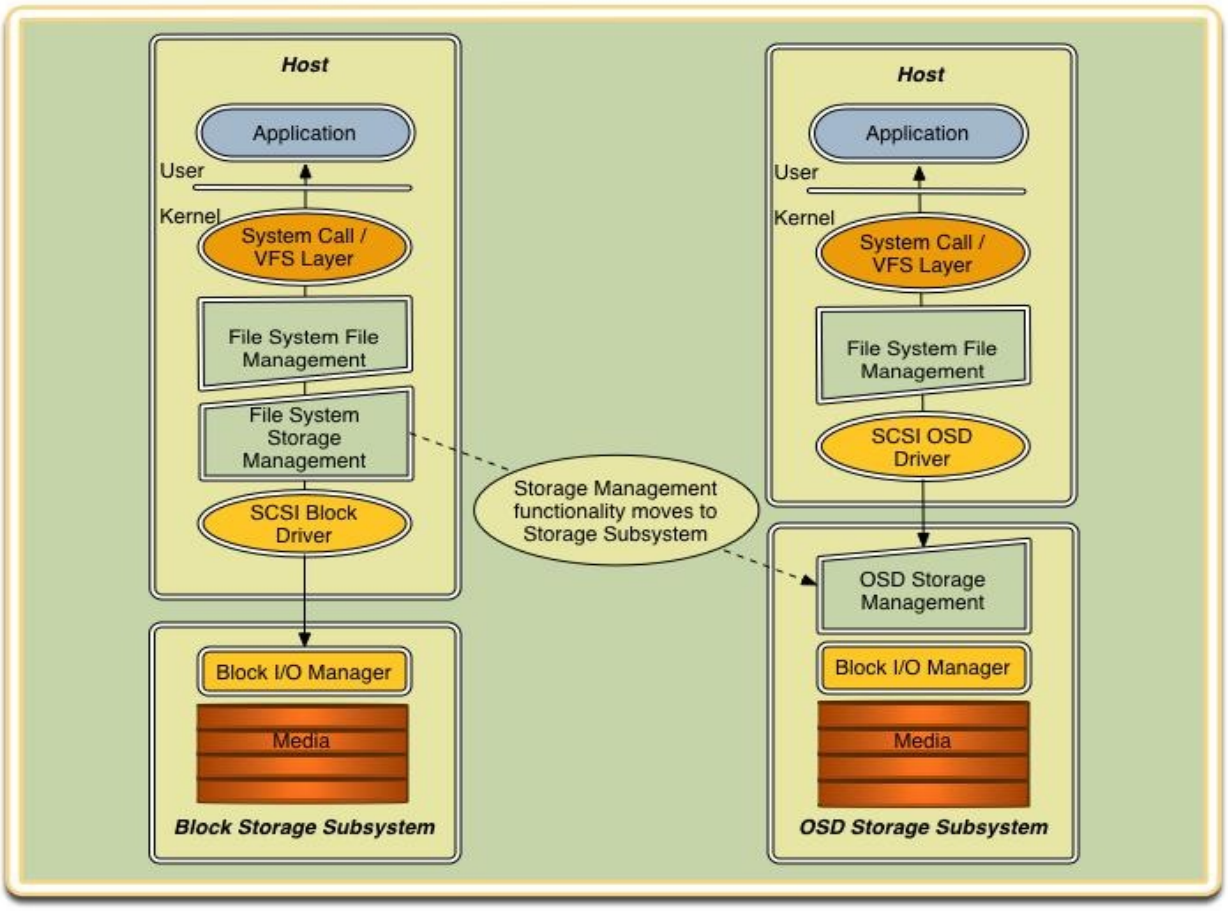
Table of Contents

1 Introduction

The SCSI OSD driver (sosd) project implements a SCSI target driver, sosd for Solaris OS. The sosd target driver fits into the Solaris DDI framework and interacts with SCSA layer for transport services.

1.1 Background

Traditional target drivers that currently exist in Solaris OS are block-based drivers. Examples of such drivers are sd (disk), st(tape) and sgen. Block-based device such as a disk or tape provides basic support for storing and accessing data using using the SCSI protocol. Object-based Storage Devices (OSD) use the same SCSI protocol but they have a feature rich command set and provides a wide variety of services to the client that account for enhanced performance of the overall sub-system as well as simplified management and control of the data bring stored.



ANSI T10 SCSI OSD Version 1 of the command set extensions that include object based semantics is available at http://www.t10.org/ftp/t10/drafts/osd/osd-r10.pdf. SNIA OSD Technical Work Group is working on OSD-2, which is further extension to this command set.

The sosd driver implements the OSD-2 command set extensions available at http://www.t10.org/ftp/t10/drafts/osd2/osd2r03.pdf and provides an API to the file system that desires to communicate with an Object-based Storage Device.

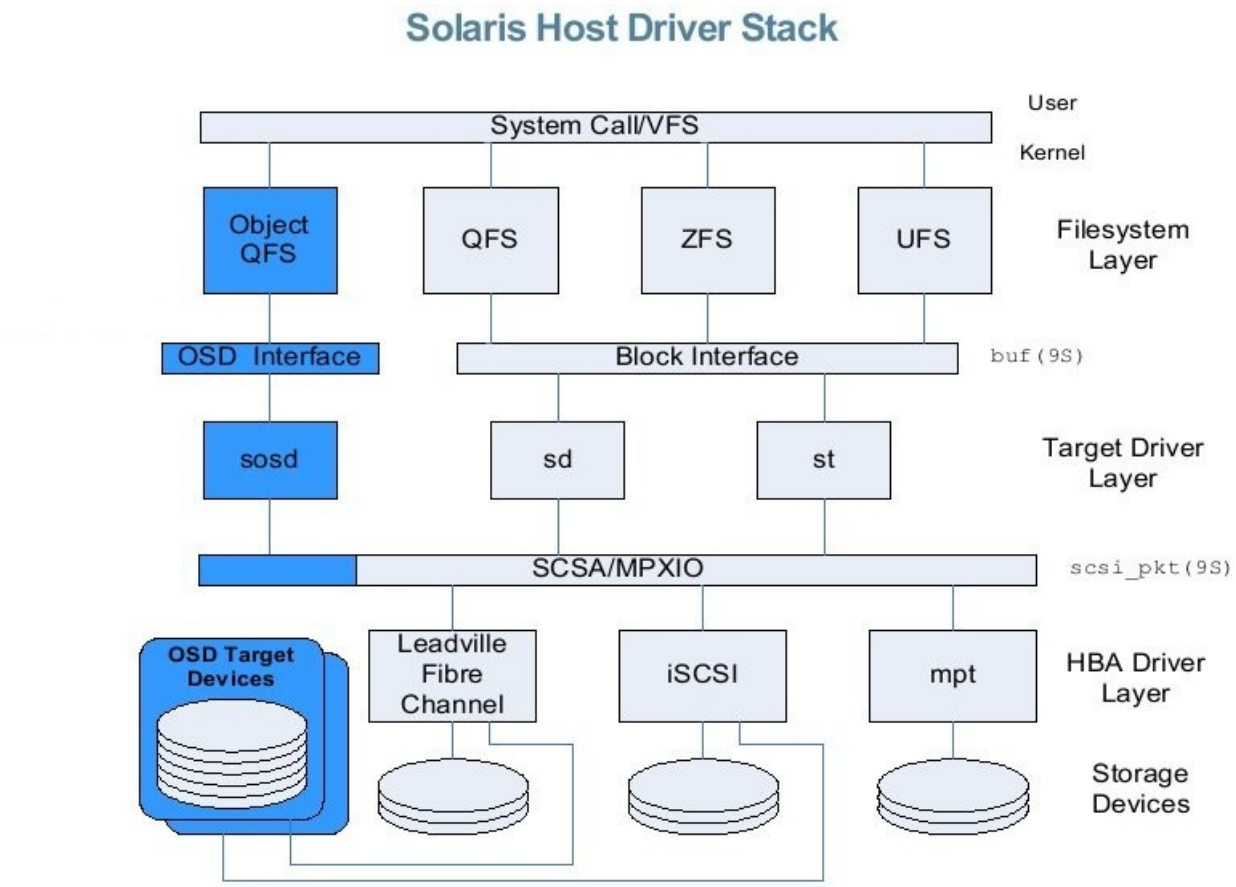
1.2 Design Goals

- Provide target driver support in a Solaris host for OSD devices.
- Provide an API to file system layer for sending OSD commands to target, including API to open and close the device.
- Modify SCSA/MPxIO and iSCSI to support OSD commands.

2 Architecture

The major components of a Solaris OSD initiator are shown in the block diagram below.

2.1 Component Block Diagram



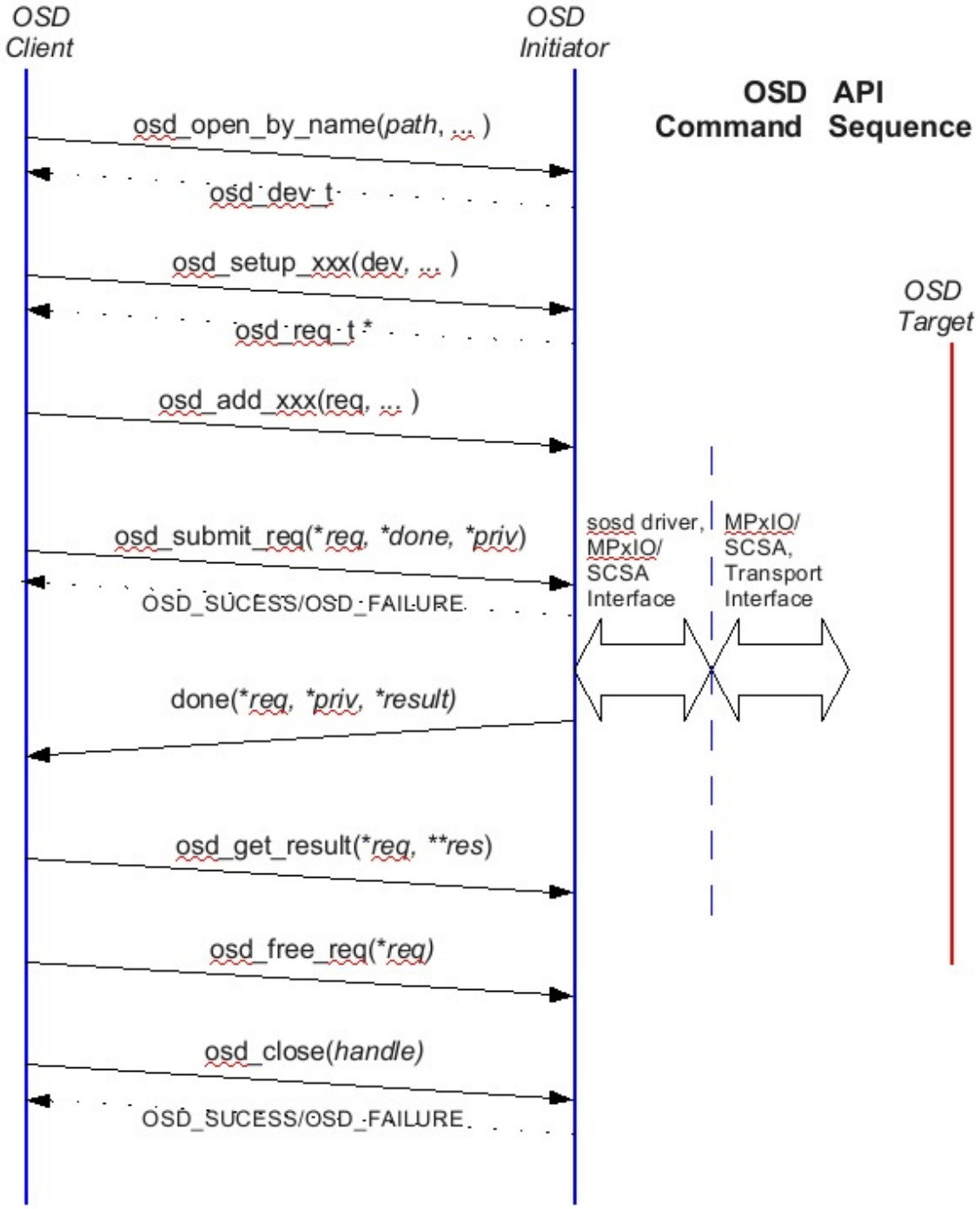
2.2 Component Descriptions

sd, st	Solaris SCSA class target drivers for SCSI block (T10 SBC) devices. sd for disk and st for tape,and sgen fall in this category
sosd	SCSI Object-based Storage Target Driver
QFS, ZFS, UFS	Block-based File systems
Object QFS	QFS with object command set support
OSD Interface	scsi_osd kernel module for sosd driver and file-system communication
MPxIO	Multipathing Software (STMS)
SCSA	Sun Common SCSI Architecture is a framework that provides the

	interface between SCSI class drivers and SCSI HBA drivers on the initiator side
OSD Target Devices	Storage Device that support Objects.
Leadville, iSCSI, mpt	Transport layer driver stacks.

3 Driver API

This section describes the API between a file-system and the sosd target driver. Clients need to include osd.h header file. The following diagram gives an overview of the command sequence with the API described in this section.



3.1 Constants

3.1.1 Common Global Definitions

```
#define      OSD_CDB_SIZE        224
```

3.1.2 Result Codes

The following are defined values for err_code in osd_result_t. Most API functions that return an int return either OSD_SUCCESS or OSD_FAILURE.

```
#define OSD_SUCCESS               0     /* The requested operation has been */
                                        /* performed successfully */
                                        /* Implies resid_data is not valid */
#define OSD_FAILURE               1     /* General failure code */
#define OSD_RECOVERED_ERROR       2     /* Command completed with sense data */
#define OSD_CHECK_CONDITION       3     /* A SCSI check condition occurred */
                                        /* while performing the request. */
#define OSD_INVALID               4     /* An invalid parameter was detected */
                                        /* in the request */
#define OSD_TOOBIG                5     /* The requested data transfer is */
                                        /* too large for the system. */
#define OSD_BADREQUEST            6     /* The requested operation cannot be */
                                        /* performed due to an unrecoverable */
                                        /* error in the request */
#define OSD_LUN_ERROR             7     /* The requested operation cannot be */
                                        /* performed due to an unrecoverable */
                                        /* error in the LUN */
#define OSD_LUN_FAILURE           8     /* The LUN has become permanently */
                                        /* unusable or inaccessible */
#define OSD_BUSY                  9     /* The requested operation cannot be */
                                        /* performed at this time */
#define OSD_RESERVATION_CONFLICT 10     /* A SCSI Reservation conflict was */
                                        /* encountered at the LUN */
#define OSD_RESIDUAL             11     /* One or more data transfers for the */
                                        /* request have a nonzero residual */
                                        /* Command succeeded, not all data was */
                                        /* transferred, look at resid_data */
#define OSD_NORESOURCES          12     /* Insufficient resources available */
                                        /* to complete the request */
```

3.1.3 Other Definitions

Various flags that could be defined for most OSD requests.
The flags are set using the osd_add_flags_to_req() function.

```
#define OSD_O_DPO                       0x00000001   /* Disable page out */

#define OSD_O_FUA                       0x00000002   /* Force unit access */

#define OSD_O_IM_NONE                   0x00000100   /* No Isolation Method */
#define OSD_O_IM_STRICT                 0x00000200   /* Strict Isolation Method */
#define OSD_O_IM_RANGE                  0x00000400   /* Range Isolation Method */
#define OSD_O_IM_FUNC                   0x00000800   /* Functional Isolation */
#define OSD_O_IM_VENDOR                 0x00001000   /* Vendor Specific Isolation */

#define OSD_O_TC_DEFAULT                0x00002000   /* Default timestamp updates */
#define OSD_O_TC_DISABLED               0x00004000   /* No timestamp updates */
```

3.2 Data Structures

3.2.1 OSD Device Handle

NAME

osd_dev_t

SYNOPSIS

```
#include <osd.h>
```

DESCRIPTION

osd_dev_t is the device handle that is given to the client when the device is opened (using osd_open_by_name()). This is implementation private data type and the client cannot make any assumptions about the contents of osd_dev_t data type. The client is expected to send this data type back to the driver with the osd_setup_xxx() functions.

3.2.2 OSD Request

NAME

osd_req_t

SYNOPSIS

```
#include <osd.h>
struct osd_req {
      osd_dev_t         oh;
      uint8_t           state;
      uint16_t          service_action;
      osd_result_t      result;
      void              (*done)(osd_req_t *, void *, osd_result_t *);
      void              *client_private;
      void              *sosd_req;
} osd_req_t;
```

DESCRIPTION

osd_req_t is the data structure that is setup upon client's request. The client must use osd_get_result() function to get the result instead of directly accessing this structure's result member. The client cannot make any assumptions about the driver private sosd_req_t data type.

The defined values for state field are as follows:

```
OSD_REQ_ALLOCATED       0x1
OSD_REQ_SUBMITTED       0x2
OSD_REQ_COMPLETED       0x3
```

3.2.3 OSD Result

NAME

osd_result_t

SYNOPSIS

```
#include <osd.h>
typedef struct osd_result {
      uint8_t     err_code;
      uint16_t    err_field_offset;
      uint16_t    service_action;
      osd_resid_t resid_data;
      uint32_t    sense_data_len;
      void        *sense_data;
} osd_result_t;
```

DESCRIPTION

osd_result_t is the data structure that contains the result of a completed OSD request. It is returned as an argument to the client's callback routine and also meant to be used with osd_get_result() API.

err_code values are defined in Section 3.1.2.
sense_data points to SCSI SENSE DATA data if err_code is OSD_RECOVERED_ERROR or OSD_CHECK_CONDITION and the length of the defined sense data is indicated by sense_data_len.
The SENSE DATA is defined in SPC-3 Section 4.5. OSD specific descriptors are defined in OSD Spec (osd2r03) Section 4.15.2.
In some cases where err_code is not OSD_SUCCESS and in all cases where err_code is OSD_RESIDUAL, there resid_data will be valid.

3.2.4 OSD Resid

This is the data at resid_data when err_code is OSD_RESDIUAL. These fields correspond to the Data In and Data Out memory segments in the the osd_request. When nonzero, each field contains a 'residual' value that is the number of data bytes from the I/O operation that were not transferred as requested. A residual value of zero indicates that all the requested bytes for the segment were transferred.
The residual value is set upon I/O completion, before the done() function is called to return the osd_req_t to the client.

```
typedef struct osd_resid {
     /* Data-In segments */
     uint64_t    ot_in_command_resid;    /* DI Command/Parameter Data */
     uint64_t    ot_in_ret_attr_resid;   /* DI Retrieved Attributes */
     uint64_t    ot_in_integrity_resid;  /* DI Integrity Check Value */

     /* Data-Out segments */
     uint64_t    ot_out_command_resid;   /* DO Command/Parameter Data */
     uint64_t    ot_out_set_attr_resid;  /* DO Set Attributes */
     uint64_t    ot_out_get_attr_resid;  /* DO Get Attributes */
     uint64_t    ot_out_integrity_resid; /* DO Integrity Check Value */
} osd_resid_t;
```

3.3 Functions

The API functions are broadly classified into four categories.

- Setup request

  - Allocate request.
  - Add attributes, capability/security and flags to request.
- Submit request
- Get Result of the request
- Free request

3.3.1 Setup OSD Request

The 'Setup' API functions shall be used by the client as an initial step in the I/O operation. Before using these functions, the client needs to obtain the osd device handle by opening the device. See Section 3.4 for device opening details. All osd_setup_xxx() functions return osd_req_t, a driver private structure pointer, where as the osd_add_xxx() functions return void. See Example in Section 3.7 for a typical use case.

3.3.1.1 Setup FORMAT OSD command

```
NAME
      osd_setup_format_osd

SYNOPSIS
      #include <osd.h>
      osd_req_t *osd_setup_format_osd(osd_dev_t oh,
          uint64_t formatted_capacity);

PARAMETERS
      oh                      OSD device handle
      formatted_capacity      Capacity of the lun in bytes to be formatted

DESCRIPTION
      This function is used to setup a FORMAT LUN command to OSD device.

RETURN_VALUES
      Non-null ptr            Successful setup
      NULL pointer            General failure
```

3.3.1.2 Setup CREATE PARTITION command

```
NAME
      osd_setup_create_partition

SYNOPSIS
     #include <osd.h>
      osd_req_t *osd_setup_create_partition(osd_dev_t oh,
          uint64_t requested_partition_id);

PARAMETERS
      oh                      OSD device handle
      requested_partition_id  Partition ID requested by the client
                              to be created on the OSD device

DESCRIPTION
      This function is used to setup a CREATE PARTITION command.
```

```
RETURN VALUES
      Non-null ptr            Successful setup
      NULL pointer            General failure
```

3.3.1.3 Setup REMOVE PARTITION command

```
NAME
      osd_setup_remove_partition

SYNOPSIS
      #include <osd.h>
      osd_req_t *osd_setup_remove_partition(osd_dev_t oh,
          uint64_t partition_id);

PARAMETERS
      oh                OSD device handle
      partition_id      Partition ID to be removed on the OSD device

DESCRIPTION
      This function is used to setup a REMOVE PARTITION command.

RETURN VALUES
      Non-null ptr            Successful setup
      NULL pointer            General failure
```

3.3.1.4 Setup CREATE command

```
NAME
      osd_setup_create_object

SYNOPSIS
      #include <osd.h>
      osd_req_t *osd_setup_create_object(osd_dev_t oh,
          uint64_t partition_id, uint64_t requested_object_id,
          uint16_t num_objs);

PARAMETERS
      oh                      OSD device handle
      partition_id            Partition ID on the OSD device on which
                              the object needs to be created
      requested_object_id     Object ID requested by the client to be
                              created on the device.
      num_objs                Total number of objects to be created with
                              this command. Object ID is unique, the
                              least of  which is requested_object_id.

DESCRIPTION
      This function is used to setup a CREATE (object) command.

RETURN VALUES
      Non-null ptr            Successful setup
      NULL pointer            General failure
```

3.3.1.5 Setup REMOVE command

NAME

osd_setup_remove_object

SYNOPSIS

```
#include <osd.h>
osd_req_t *osd_setup_remove_object(osd_dev_t oh,
    uint64_t partition_id, uint64_t object_id);
```

PARAMETERS

oh	OSD device handle
partition_id	Partition ID on the OSD device on which the object needs to be created
object_id	Object ID to be removed.

DESCRIPTION

This function is used to setup a REMOVE (object) command.

RETURN VALUES

Non-null ptr	Successful setup
NULL pointer	General failure

3.3.1.6 Setup CREATE AND WRITE command

NAME

osd_setup_create_and_write

SYNOPSIS

```
#include <osd.h>
osd_req_t *osd_setup_create_and_write(osd_dev_t oh,
    uint64_t partition_id, uint64_t object_id, uint64_t len,
    uint64_t starting_byte_addr, uint8_t num_iovecs, iovec_t *iov);
```

PARAMETERS

oh	OSD device handle
partition_id	Partition ID on the OSD device
object_id	Object ID to be created
len	Number of bytes to be transferred
start_byte_addr	offset where the write should commence in the specified object relative to 1st byte(byte 0)
num_iovecs	Number of contiguous iovec_t structures setup for data transfer
iov	Address of 1st iovec_t structure

DESCRIPTION

This function is used to setup a CREATE AND WRITE (object) command.

RETURN VALUES

Non-null ptr	Successful setup
NULL pointer	General failure

3.3.1.7 Setup CREATE AND WRITE bp command

NAME

osd_setup_create_and_write_bp

SYNOPSIS

```
#include <osd.h>
osd_req_t *osd_setup_create_and_write_bp(osd_dev_t oh,
    uint64_t partition_id, uint64_t object_id, uint64_t len,
    uint64_t starting_byte_addr, struct buf *bp);
```

PARAMETERS

oh	OSD device handle
partition_id	Partition ID on the OSD device
object_id	Object ID to be created
len	Number of bytes to be transferred
start_byte_addr	offset where the write should commence in the specified object relative to 1st byte(byte 0)
bp	Pointer to buf structure created by client

DESCRIPTION

This function is used to setup a CREATE AND WRITE (object) command, the the buf structure is given by the client.

RETURN_VALUES

Non-null ptr	Successful setup
NULL pointer	General failure

3.3.1.8 Setup WRITE command

NAME

osd_setup_write

SYNOPSIS

```
#include <osd.h>
osd_req_t *osd_setup_write(osd_dev_t oh,
    uint64_t partition_id, uint64_t object_id, uint64_t len,
    uint64_t starting_byte_addr, uint8_t num_iovecs, iovec_t *iov);
```

PARAMETERS

oh	OSD device handle
partition_id	Partition ID on the OSD device
object_id	Object ID to be written to.
len	Number of bytes to be transferred
start_byte_addr	offset where the write should commence in the specified object relative to 1st byte(byte 0)
num_iovecs	Number of contiguous iovec_t structures setup for data transfer
iov	Address of 1st iovec_t structure

DESCRIPTION

This function is used to setup a WRITE command to the OSD device.

RETURN VALUES

Non-null ptr	Successful setup
NULL pointer	General failure

3.3.1.9 Setup WRITE bp command

NAME

osd_setup_write_bp

SYNOPSIS

```
#include <osd.h>
osd_req_t *osd_setup_write_bp(osd_dev_t oh,
    uint64_t partition_id, uint64_t object_id, uint64_t len,
    uint64_t starting_byte_addr, struct buf *bp);
```

PARAMETERS

oh	OSD device handle
partition_id	Partition ID on the OSD device
object_id	Object ID to be written to.
len	Number of bytes to be transferred
start_byte_addr	offset where the write should commence in the specified object relative to 1st byte(byte 0)
bp	Pointer to struct buf setup by the client.

DESCRIPTION

This function is used to setup a WRITE command to the OSD device, where the buf pointer is directly given.

RETURN VALUES

Non-null ptr	Successful setup
NULL pointer	General failure

3.3.1.10 Setup READ command

NAME

osd_setup_read

SYNOPSIS

```
#include <osd.h>
osd_req_t *osd_setup_read(osd_dev_t oh,
    uint64_t partition_id, uint64_t object_id, uint64_t len,
    uint64_t starting_byte_addr, uint8_t num_iovecs, iovec_t *iov);
```

PARAMETERS

oh	OSD device handle
partition_id	Partition ID on the OSD device
object_id	Object ID to be read from
len	Number of bytes to be transferred
start_byte_addr	offset where the read should commence in the specified object relative to 1st byte(byte 0)

	num_iovecs	Number of contiguous iovec_t structures setup for data transfer
	iov	Address of 1st iovec_t structure

DESCRIPTION

This function is used to setup a READ command to OSD device.

RETURN VALUES

Non-null ptr	Successful setup
NULL pointer	General failure

3.3.1.11 Setup READ bp command

NAME

osd_setup_read_bp

SYNOPSIS

```
#include <osd.h>
osd_req_t *osd_setup_read_bp(osd_dev_t oh,
    uint64_t partition_id, uint64_t object_id, uint64_t len,
    uint64_t starting_byte_addr, struct buf *bp);
```

PARAMETERS

oh	OSD device handle
partition_id	Partition ID on the OSD device
object_id	Object ID to be read from.
len	Number of bytes to be transferred
start_byte_addr	offset where the read should commence in the specified object relative to 1st byte(byte 0)
bp	Pointer to struct buf setup by the client.

DESCRIPTION

This function is used to setup a READ command to the OSD device, where the buf pointer is directly given.

RETURN VALUES

Non-null ptr	Successful setup
NULL pointer	General failure

3.3.1.12 Setup APPEND command

NAME

osd_setup_append

SYNOPSIS

```
#include <osd.h>
osd_req_t *osd_setup_append(osd_dev_t oh,
    uint64_t partition_id, uint64_t object_id, uint64_t len,
    uint8_t num_iovecs, iovec_t *iov);
```

PARAMETERS

oh	OSD device handle

```
      partition_id            Partition ID on the OSD device
      object_id               Object ID to be appended to.
      len                     Number of bytes to be transferred
      num_iovecs              Number of contiguous iovec_t structures
                              setup for data transfer
      iov                     Address of 1st iovec_t structure

DESCRIPTION
      This function is used to setup an APPEND command to the OSD device.

RETURN VALUES
      Non-null ptr            Successful setup
      NULL pointer            General failure
```

3.3.1.13 Setup APPEND bp command

```
NAME
      osd_setup_append_bp

SYNOPSIS
      #include <osd.h>
      osd_req_t *osd_setup_append_bp(osd_dev_t oh,
          uint64_t partition_id, uint64_t object_id, uint64_t len,
          struct buf *bp);

PARAMETERS
      oh                      OSD device handle
      partition_id            Partition ID on the OSD device
      object_id               Object ID to be appended to.
      len                     Number of bytes to be transferred
      bp                      Pointer to struct buf setup by the client.

DESCRIPTION
      This function is used to setup an APPEND command to the OSD device,
      where the buf pointer is provided by the client.

RETURN VALUES
      Non-null ptr            Successful setup
      NULL pointer            General failure
```

3.3.1.14 Setup FLUSH command

```
NAME
      osd_setup_flush

SYNOPSIS
     #include <osd.h>
     #include <scsi/scsi_osd.h>
      osd_req_t *osd_setup_flush(osd_dev_t oh, uint8_t flush_scope
          uint64_t partition_id, uint64_t object_id, uint64_t flush_len,
          uint64_t flush_start_byte_addr);

PARAMETERS
      oh                      OSD device handle
```

flush_scope	Defined in scsi_osd.h (See Notes)
partition_id	Partition ID on the OSD device
object_id	Object ID to be created
flush_len	Number of bytes to be flushed
flush_start_byte_addr	offset where the flush should commence in the specified object relative to 1st byte(byte 0)

DESCRIPTION

This function is used to setup a FLUSH command to OSD device.

RETURN_VALUES

Non-null ptr	Successful setup
NULL pointer	General failure

NOTES

The flush_scope values defined in scsi_osd.h are as follows:.

```
#define OSD_FLUSH_SCOPE_USER_DATA_AND_ATTRIBUTES        0x0
#define OSD_FLUSH_SCOPE_USER_ATTRIBUTES_ONLY            0x1
#define OSD_FLUSH_SCOPE_USER_DATA_RANGE_AND_ATTRIBUTES  0x2
```

3.3.1.15 Setup FLUSH OSD command

NAME

osd_setup_flush_osd

SYNOPSIS

```
#include <osd.h>
#include <scsi/scsi_osd.h>
 osd_req_t *osd_setup_flush_osd(osd_dev_t oh,
     uint8_t flush_scope);
```

PARAMETERS

oh	OSD device handle
flush_scope	Defined in scsi_osd.h (See Notes)

DESCRIPTION

This function is used to setup a FLUSH OSD command to OSD device.

RETURN_VALUES

Non-null ptr	Successful setup
NULL pointer	General failure

NOTES

The flush_scope values defined in scsi_osd.h are as follows:.

```
#define OSD_FLUSH_OSD_SCOPE_PARTITION_LIST                0x0
#define OSD_FLUSH_OSD_SCOPE_ROOT_OBJECT_ATTRIBUTES_ONLY   0x1
#define OSD_FLUSH_OSD_SCOPE_ALL                           0x2
```

3.3.1.16 Setup CLEAR command

NAME

osd_setup_clear

SYNOPSIS

```
#include <osd.h>
osd_req_t *osd_setup_clear(osd_dev_t oh,
    uint64_t partition_id, uint64_t object_id, uint64_t len,
    uint64_t starting_byte_address);
```

PARAMETERS

oh	OSD device handle
partition_id	Partition ID on the OSD device
object_id	Object ID to be cleared.
len	Number of bytes to be transferred
starting_byte_address	offset where the clear should commence in the specified object relative to 1st byte(byte 0)

DESCRIPTION

This function is used to setup an CLEAR command to the OSD device.

RETURN_VALUES

Non-null ptr	Successful setup
NULL pointer	General failure

3.3.1.17 Setup PUNCH command

NAME

osd_setup_punch

SYNOPSIS

```
#include <osd.h>
osd_req_t *osd_setup_punch(osd_dev_t oh,
    uint64_t partition_id, uint64_t object_id, uint64_t len,
    uint64_t starting_byte_address);
```

PARAMETERS

oh	OSD device handle
partition_id	Partition ID on the OSD device
object_id	Object ID to be punched
len	Number of bytes to be transferred
starting_byte_address	offset where the punch should commence in the specified object relative to 1st byte(byte 0)

DESCRIPTION

This function is used to setup an PUNCH command to the OSD device.

RETURN_VALUES

Non-null ptr	Successful setup
NULL pointer	General failure

3.3.1.18 Setup OBJECT STRUCTURE CHECK command

NAME

```
      osd_setup_object_structure_check

SYNOPSIS
      #include <osd.h>
      osd_req_t *osd_setup_object_structure_check(osd_dev_t oh,
          uint64_t partition_id);

PARAMETERS
      oh                      OSD device handle
      partition_id            Partition ID on the OSD device

DESCRIPTION
      This function is used to setup an OBJECT STRUCTURE CHECK command to
      the OSD device that verifies the integrity of the OBSD.

RETURN VALUES
      Non-null ptr            Successful setup
      NULL pointer            General failure
```

3.3.1.19 Setup SET ATTRIBUTES

```
NAME
      osd_setup_set_attr

SYNOPSIS
     #include <osd.h>
      osd_req_t *osd_setup_set_attr(osd_dev_t oh,
          uint64_t partition_id, uint64_t object_id);

PARAMETERS
      oh                      OSD device handle
      partition_id            Partition ID on the OSD device
      object_id               Object ID to lookup

DESCRIPTION
      The call allocates the request and initializes the service action to
      SET ATTRIBUTES.
      This needs to be followed with an osd_add_set_*_attr_to_req() call
      to fill in the remaining fields of the CDB.

RETURN VALUES
      Non-null ptr            Successful setup
      NULL pointer            General failure

NOTES
```

3.3.1.20 Setup GET ATTRIBUTES

```
NAME
      osd_setup_get_attr

SYNOPSIS
     #include <osd.h>
```

```
osd_req_t *osd_setup_get_attr(osd_dev_t oh,
    uint64_t partition_id, uint64_t object_id);
```

PARAMETERS

oh	OSD device handle
partition_id	Partition ID on the OSD device
object_id	Object ID to lookup

DESCRIPTION

The call allocates the request and initializes the service action to GET ATTRIBUTES.
This needs to be followed with an osd_add_get_*_attr_to_req() call to fill in the remaining fields of the CDB.

RETURN VALUES

Non-null ptr	Successful setup
NULL pointer	General failure

NOTES

3.3.1.21 Add SET ATTRIBUTES command (in PAGE format) using CDB fields

NAME

osd_add_set_page_1attr_cdb

SYNOPSIS

```
#include <osd.h>
void osd_add_set_page_1attr_cdb(osd_req_t *req,
    uint32_t attr_page, uint32_t attr_num,
    uint32_t attr_len, char *attr_val);
```

PARAMETERS

req	OSD request obtained using osd_setup_xxx()
attr_page	Page number of attribute value to be set
attr_num	Attribute number within the attribute page
attr_len	Length of attribute value in bytes. to be retrieved
attr_val	Pointer to attribute value (max 18 bytes)

DESCRIPTION

The function can be used to add a set attribute command to a request where the attribute value is less than (or equal to) 18 bytes.
The attribute value is stored directly in the CDB, without having to use a Data-out buffer.

NOTES

The function will ASSERT if attr_len is greater than 18.

3.3.1.22 Add SET PAGE ATTRIBUTES to a Request

NAME

osd_add_set_page_1attr_to_req

SYNOPSIS

```
#include <osd.h>
 void osd_add_set_page_1attr_to_req(osd_req_t *req,
     uint32_t set_attr_page, uint32_t set_attr_num,
     uint32_t set_attr_len, uint32_t *set_attr_val);
```

PARAMETERS

req	OSD Request that is already setup using one of the setup APIs.
set_attr_page	Page number of attribute value to Set
set_attr_num	Attribute number with attribute page
set_attr_len	Length of attribute value in bytes
set_attr_val	Value of attribute to set.

DESCRIPTION

This function is used to ADD SET ATTRIBUTES in PAGE format to an OSD OSD request that has already been setup.

NOTES

The current limitation on using this function is that it can only be used with SET ATTRIBUTES service action (established using osd_setup_get_attr() function) in a non-BIDI situation. Calling this function multiple times will cause only the last call to take effect. Invalid setup of CDB will result in failure of the command during submit time.

3.3.1.23 Add GET PAGE ATTRIBUTES to a Request

NAME

osd_add_get_page_attr_to_req

SYNOPSIS

```
#include <osd.h>
 void osd_add_get_page_attr_to_req(osd_req_t *req,
     uint32_t get_attr_page, uint32_t get_attr_alloc_len,
     void *ret_attr);
```

PARAMETERS

req	OSD Request that is already setup using one of the setup APIs.
get_attr_page	Atribute page number to be retrieved
get_attr_alloc_len	Memory allocated for attribute value being retrieved.
ret_attr	Pointer to location where attributes need to be retrieved for client's use.

DESCRIPTION

This function is used to ADD GET ATTRIBUTES in PAGE format to an OSD OSD request that has already been setup.

NOTES

The current limitation on using this function is that it can only be used with GET ATTRIBUTES service action (established using osd_setup_get_attr() function) in a non-BIDI situation.

Calling this function multiple times will cause only the last call to take effect. Invalid setup of CDB will result in failure of the command during submit time.

3.3.1.24 Add SET LIST ATTRIBUTES entry to request

NAME

osd_add_set_list_entry_to_req

SYNOPSIS

```
#include <osd.h>
void osd_add_set_list_entry_to_req(osd_req_t *req,
    uint32_t set_attr_page, uint32_t set_attr_num,
    uint32_t set_attr_len, void *set_attr_val);
```

PARAMETERS

req	OSD request already setup
set_attr_page	Attribute page number
set_attr_num	Attribute number
set_attr_len	Length of the attribute
set_attr_val	Pointer to the attribute value

DESCRIPTION

This function is used to ADD SET ATTRIBUTES in LIST format. The function can be iteratively used by the client, where the client passes one attribute per iteration and the driver constructs a list of attributes that need to be set.

NOTES

The current limitation on using this function is that it can only be used with SET ATTRIBUTES service action (established using osd_setup_get_attr() function) in a non-BIDI situation.

3.3.1.25 Add GET LIST ATTRIBUTES entry to request

NAME

osd_add_get_list_entry_to_req

SYNOPSIS

```
#include <osd.h>
void osd_add_get_list_entry_to_req(osd_req_t *req,
    uint32_t get_attr_page, uint32_t get_attr_num,
    uint32_t get_attr_alloc_len, void *ret_attr_buf);
```

PARAMETERS

req	OSD request already setup
get_attr_page	Attribute page number to be retrieved.
get_attr_num	Attribute number to be retrieved.
get_attr_alloc_len	Buffer Size to store retrieved attr.
ret_attr_buf	Buf Pointer to store retrieved attributes

DESCRIPTION

```
      This function is used to ADD GET ATTRIBUTES in LIST format.
      The function can be iteratively used by the client, where the client
      passes one attribute per iteration and the driver constructs
      a list of attributes that need to be retrieved.

NOTES
      This function will have buffers transferred in both directions and
      hence it needs BIDI to be implemented to work successfully.
```

3.3.1.26 Add CAPABILITY and SECURITY to request

```
NAME
      osd_add_capability_security_to_req

SYNOPSIS
     #include <osd.h>
      void osd_add_capability_security_to_req(osd_req_t *req,
          osd_capability_format_t *ocap,
          osd_security_parameters_t *osec);

PARAMETERS
      req                     OSD request already setup
      ocap                    Pointer to osd_capability_format_t
                              structure (defined in scsi_osd.h).
      osec                    Pointer to osd_security_parameters_t
                              structure (defined in scsi_osd.h).

DESCRIPTION
      This function is used to ADD CAPABILITY and SECURITY parameters
      to an already setup request.

      osd_capability_format_t and osd_security_parameters_t are defined
      in scsi_osd.h

NOTES
```

3.3.1.27 Add FLAGS to request

```
NAME
      osd_add_flags_to_req

SYNOPSIS
     #include <osd.h>
      void osd_add_flags_to_req(osd_req_t *req, uint32_t flags);

PARAMETERS
      req                     OSD request already setup
      flags                   combination of flags to be added to the
                              request. Ex: (OSD_O_DPO | OSD_O_TC_DEFAULT)

DESCRIPTION
```

This function adds OSD request related flags to the command. Valid flags are defined in Section 3.1.3.

NOTES

3.3.2 Submit OSD Request

NAME

osd_submit_req

SYSNOPSIS

```
#include <osd.h>
int osd_submit_req(osd_req_t *req,
    void (* done)(osd_req_t *, void *, osd_result_t *),
    void *ct_priv);
```

PARAMETERS

req	osd_req_t pointer returned by osd_setup_XXX APIs
done	routine to callback into client after completion
ct_priv	Client private data that is sent back as 2nd argument to callback routine for OSD requests.

DESCRIPTION

The osd_submit_req() function should be used following the osd_setup_xxx (and osd_add_xxx()) functions by the client. This function sends the command to the underlying transport. The osd_setup_xxx() API functions only allocate and fill the CDB as per the client's requests. done() is the callback routine which will be invoked by the driver on the completion of the request.

The 3rd argument to the done() callback is a completed osd_result_t structure which indicates the status of the command. Clients can use the status from this structure instead of issuing a separate osd_get_result() call.

RETURN_VALUES

OSD_SUCCESS	Successful completion
OSD_FAILURE	General failure
OSD_INVALID	Invalid parameter in the request
OSD_TOOBIG	Requested data transfer too large for the system
OSD_BADREQUEST	Requested operation cannot be performed due to an unrecoverable error in the request
OSD_LUN_ERROR	Requested operation cannot be performed due to and unrecoverable error in the LUN
OSD_LUN_FAILURE	LUN has become permanently unusable or inaccessible
OSD_BUSY	Requested operation cannot be performed at this time
OSD_NORESOURCES	Insufficient resources available to complete the request.
OSD_RESERVATION_CONFLICT	A SCSI Reservation conflict was encountered at the LUN

3.3.3 Get Result of OSD Request

```
NAME
      osd_get_result

SYNOPSIS
      #include <osd.h>
      void osd_get_result(osd_req_t *req, osd_result_t **result);

PARAMETERS
      req         osd_req_t ptr that was used to submit the request
      result      Pointer to pointer to osd_result_t structure

DESCRIPTION
      This function would be used by the client to obtain the result of a
      completed request.

      The 'result' argument is a double pointer to osd_result_t.
      Although, the result of a submitted request is given along with the
      callback, this function could be used anytime after the done()
      callback is received but before the request is freed using the
      osd_free_req() call.

      Result structure defined as osd_result_t in Section 3.2.3.
```

3.3.4 Free OSD Request

```
NAME
      osd_free_req

SYNOPSIS
      #include <osd.h>
      void osd_free_req(osd_req_t *req);

PARAMETERS
      req   osd_req_t ptr that was used to setup and submit the request

DESCRIPTION
      This function is used when the client is done with the I/O operation,
      to free up osd request resources. The osd_req_t pointer must not be
      used after this call.
```

3.4 Device Open/Close methods

This section discusses procedures to open (and close) an OSD device.

3.4.1 OPEN

Open the device using the device file path.

```
int osd_open_by_name(char *path, int flags, cred_t *cr, osd_dev_t *ohp);
```

Notes:

path	absolute path to the device (/dev/osd/osd<guid>,root)
flags	FEXCL, FNDELAY, FREAD, FWRITE (for kernel clients)
cr	Pointer to the credentials structure, obtained using CRED() or ddi_get_cred()
oh	pointer to osd_dev_t that will be filled by the driver upon a successful open.

3.4.2 CLOSE

API to Close the device.

```
int osd_close(osd_dev_t oh, int flags, cred_t *cr);
```

3.5 Miscellaneous Utility API functions

The following are a few general purpose API functions which aid in retrieving driver or underlying transport layer characteristics.

3.5.1 Maximum DMA Transfer size

The driver layer will not attempt to break-up a single I/O request from the filesystem into multiple DMA requests. This function returns the maximum allowed DMA transfer size (in bytes) so that client can limit the I/O to the specified limit. Return value of -1 indicates a failure to retrieve the maximum DMA transfer size.

```
int osd_get_max_dma_size(osd_dev_t oh);
```

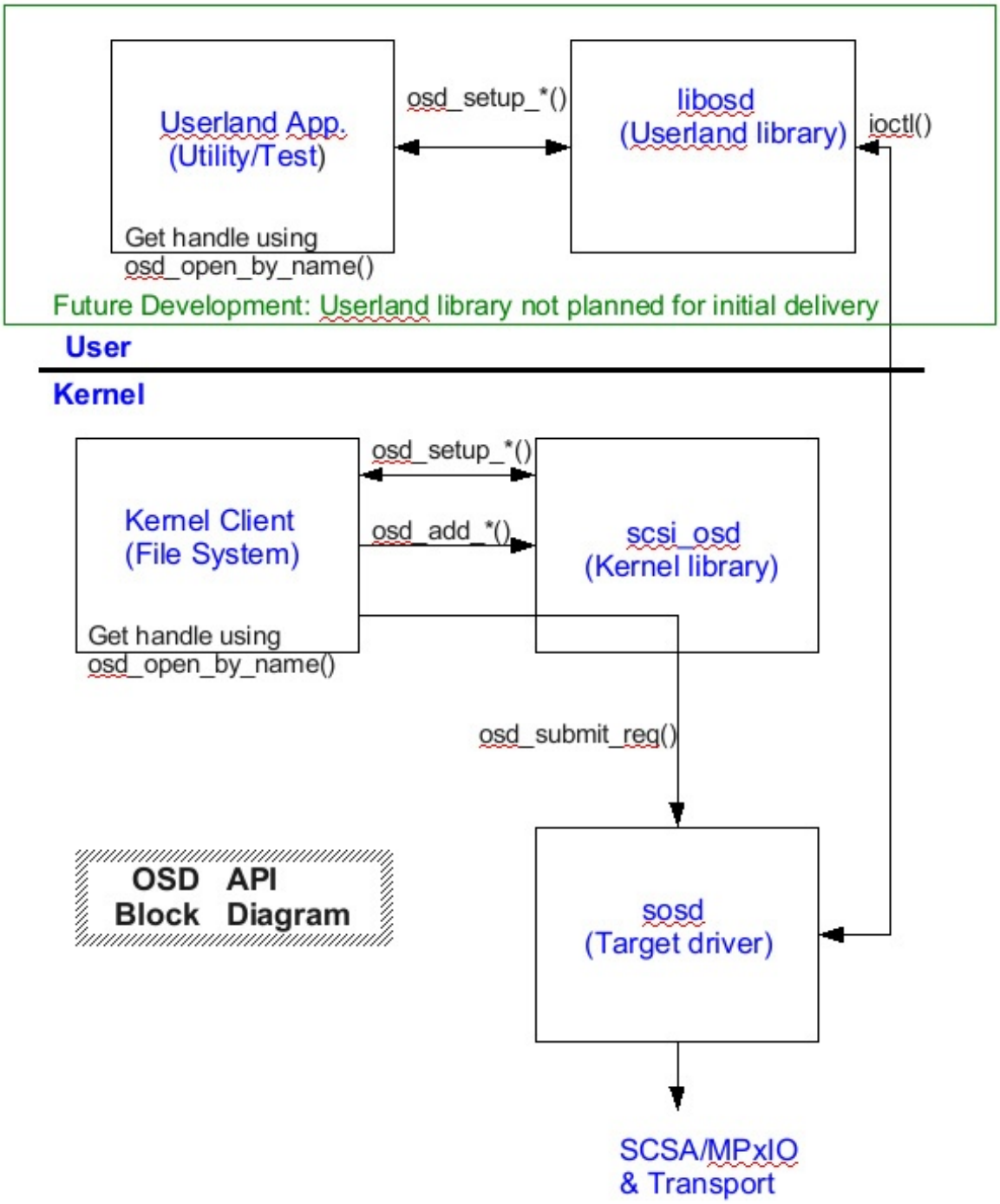
3.5.2 API Version

This function returns an unsigned int value of the OSD API version that the client is currently using.

```
uint32_t osd_get_api_version();
```

3.6 Driver and kernel module Interaction

The following block diagram depicts how various modules interact with each other.


Userland App.
(Utility/Test)
Get handle using
osd_open_by_name()
osd_setup_*()
libosd
(Userland library)
ioctl()
Future Development: Userland library not planned for initial delivery
User
Kernel
Kernel Client
(File System)
Get handle using
osd_open_by_name()
osd_setup_*()
osd_add_*()
scsi_osd
(Kernel library)
osd_submit_req()
sosd
(Target driver)
SCSA/MPxIO
& Transport
OSD API
Block Diagram

3.7 API Use Case (Kernel mode)

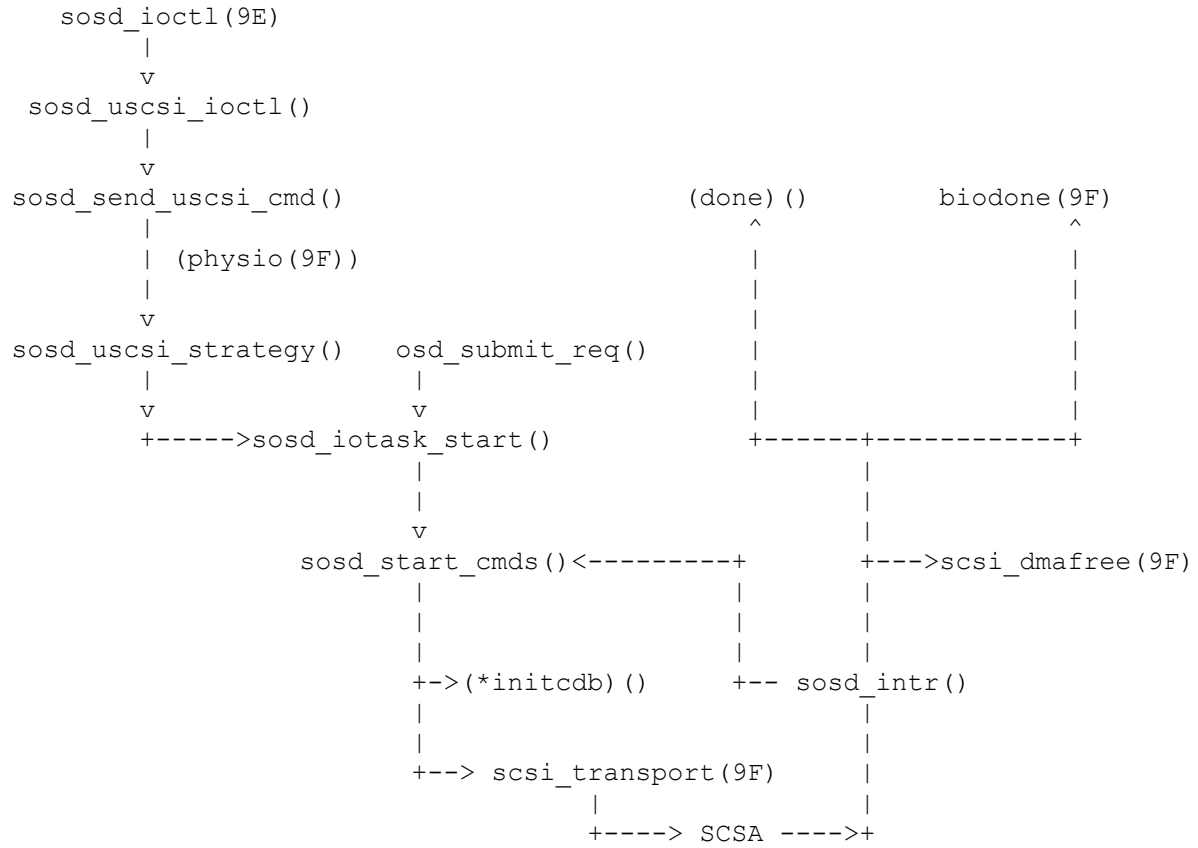
```
int done(osd_req_t *req, void *ct_priv, osd_result_t *res);

int osd_fs_write()
{
      int               err;
      int               part_id = 0x11171;    /* example */
      int               obj_id = 0x11181;     /* example */
      int               len = 256;            /* example */
      int               start_byte_addr = 0;  /* example */
      int               niov = 1;             /* example */
      iovec_t           iov[1];               /* example */
      void              *priv;
      cred_t            *kcred;
      dev_t             devp;
      osd_req_t         *req;
      osd_dev_t         handle;

      kcred = ddi_get_cred();

      /* Get handle using device pathname */
      if (err = osd_open_by_name(
          "/dev/osd/osd010000144F1D837C00002A00479FAC04,root",
          FREAD|FWRITE, kcred, &handle) != OSD_SUCCESS) {
            syslog(LOG_ERR, "osd_fs_write: osd_open_by_name() Failed.");
            return (OSD_FAILURE);
      }

      /*
       * Setup a WRITE cmd
       *
       * handle -       handle returned by ioctl cmd
       * part_id -      partition id the write cmd is directed to
       * obj_id -       object id of the object being written to
       * niov -         # of iovec structures used (currently we use only 1)
       * iov -          Pointer to (first) iovec structure.
       * start_byte_addr - Starting byte address of the write cmd
       */
      /*
       * Use case assumes iov[0]->iov_base points to valid data and
       * iov[0]->iov_len contains length of data to be transferred.
       */
      if (req = osd_setup_write(handle, part_id, obj_id, len,
          start_byte_addr, niov, &iov[0])) == NULL) {
            syslog(LOG_ERR, "osd_fs_write: osd_setup_write() Failed.");
            return (OSD_FAILURE);
      }

      /* Send the osd request. */
      if (err = osd_submit_req(req, done, priv) != OSD_SUCCESS) {
            syslog(LOG_ERR, "osd_submit_req error: Command failed.\n", err);
      }
      rval = osd_close(handle, FREAD | FWRITE, kcred);
}
```

```
void
done(osd_req_t *req, void *priv, osd_result_t *res)
{

      if (res->err_code != OSD_SUCCESS) {
            /* Process the result data to find out what went wrong */
            if (res->err_code == OSD_CHECK_CONDITION) ||
                (res->err_code == OSD_RECOVERED_ERROR) {
                  /*
                   * Interpret res->sense_data
                   */
            }

            /* resid data could be valid if (err_code != OSD_SUCCESS) */
            if (res->resid.ot_in_command_resid)
                  /* Handle resid data */
            if (res->resid.ot_in_ret_attr_resid)
                  /* Handle resid data */
            if (res->resid.ot_in_integrity_resid)
                  /* Handle resid data */
            if (res->resid.ot_out_command_resid)
                  /* Handle resid data */
            if (res->resid.ot_out_set_attr_resid)
                  /* Handle resid data */
            if (res->resid.ot_out_get_attr_resid)
                  /* Handle resid data */
            if (res->resid.ot_out_integrity_resid)
                  /* Handle resid data */

            /* Handle other conditions without Sense/Resid data */
      }

      /* Free the OSD Request */
      err = osd_free_req(req);
}
```

4 SOSD Driver Internals

4.1 Basic Architecture

This section describes the sosd driver IO processing sequence and the interface between sosd and MPxIO/SCSA.



sosd_iotask_start() is the entry point for the API code to send commands into sosd driver's processing queue. osd_submit_req() API from scsi_osd module calls this entry point to send commands down to the target driver. The target driver does further validation before it sends the command to the transport.

4.2 Device Names

One minor node for the entire OSD lun shall be created along with a symbolic link in the /dev/osd directory.

A sample listing of an OSD device node enumerated under MPxIO would be as follows:

```
osd010000144F1D837C00002A00479FAC04,root -> ../../devices/scsi_vhci/object-
store@g010000144f1d837c00002a00479fac04:osd0,root
```

5 Limitations

The following are some limitations that are temporarily applicable to some of the _ATTRIBUTES_ commands especially when multiple operations are being performed in the same command. The bi-directional limitation applies if both retrieval and setting of attributes is requested. Similarly, if the transfer size of a WRITE command is too large to map DMA resources or if a WRITE command also has a SET_ATTRIBUTES command added, the command break-up limitation applies.

5.1 Bi-directional commands

The OSD command set also allows bi-directional SCSI transfers. Each OSD command can transfer read or write data while getting and/or setting attributes with the same operation. Solaris does not currently allow bi-directional SCSI transfers and a general-purpose solution to enable such transfers would impact a large set of Solaris components ranging from SCSA to the bus nexus drivers. Since the immediate consumers of the sosd driver do not require require bi-directional transfers this capability will be deferred to a future project.

5.2 Command break-up

Several OSD protocol commands guarantee atomicity to the client. sosd target driver will not try to break-up an OSD command in an effort to complete a request in multiple steps. If the underlying HBA cannot satisfy the request the request will be returned to the application with OSD_TOOBIG error.

6 References

- ANSI T10 OSD Standard: http://www.t10.org/ftp/t10/drafts/osd/osd-r10.pdf
- ANSI T10 OSD-2 Standard: http://www.t10.org/ftp/t10/drafts/osd2/osd2r03.pdf
- OpenSolaris web page of OSD project: http://opensolaris.org/os/project/osd/