SCSI OSD Driver

Design

Version 0.3
February 2008

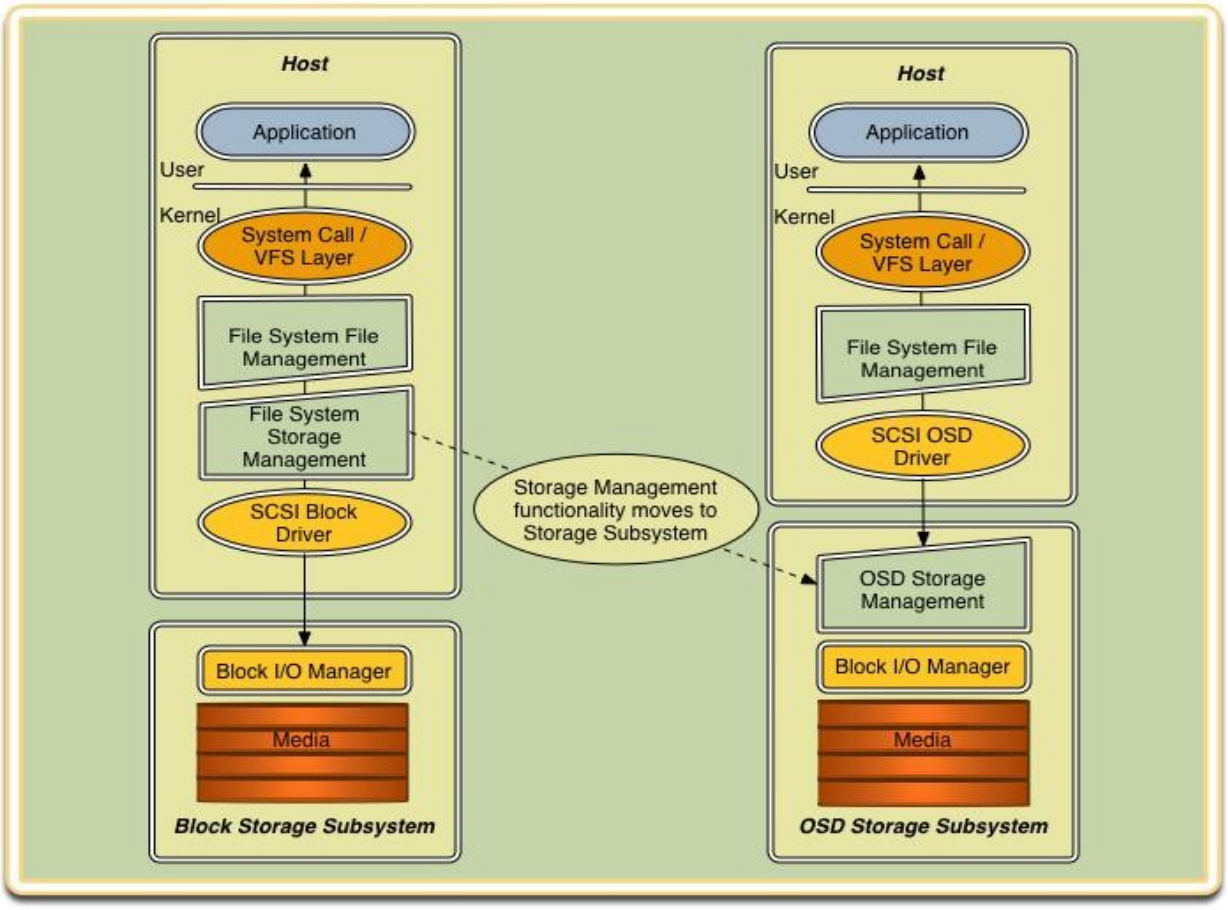
Table of Contents

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 ses(enclosure services). 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 avilable at http://www.t10.org/ftp/t10/drafts/osd2/osd2r02.pdf and provides an API to the file system that desires to communicate with an Object-based Storage Device.

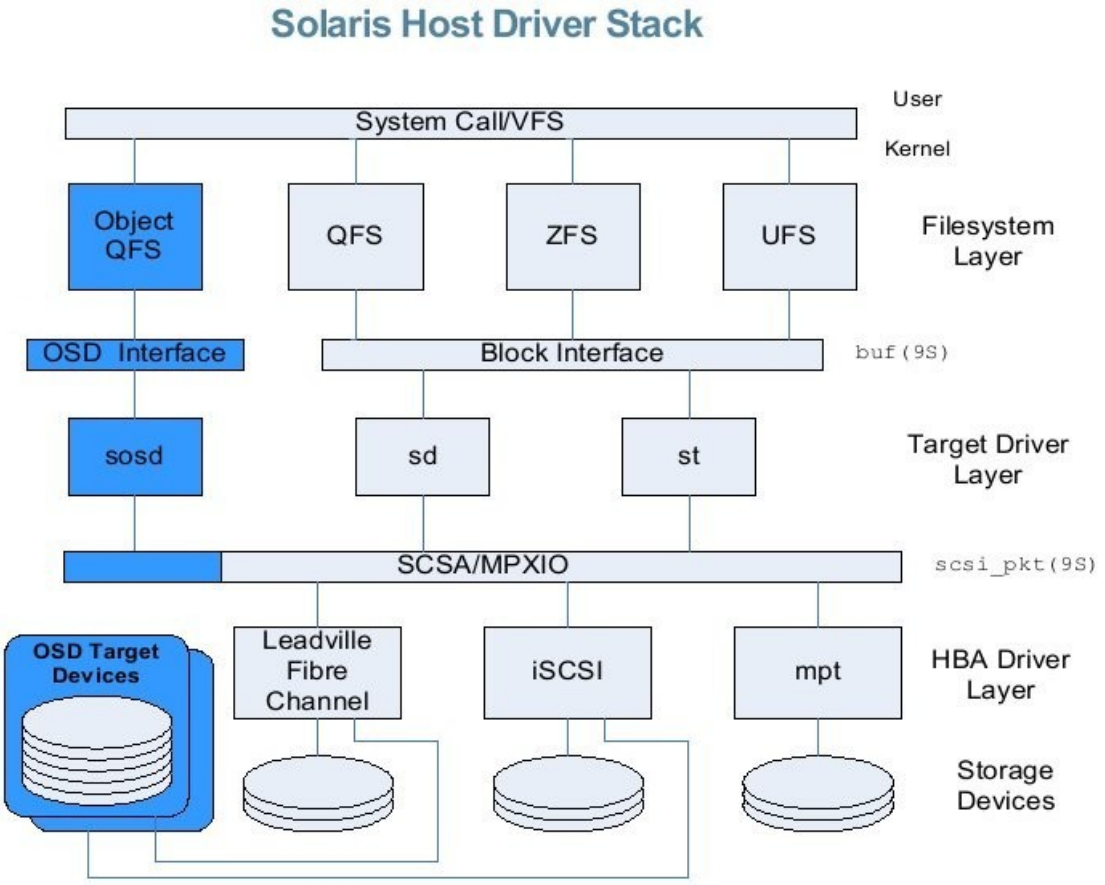
1.2 Design Goals

- 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



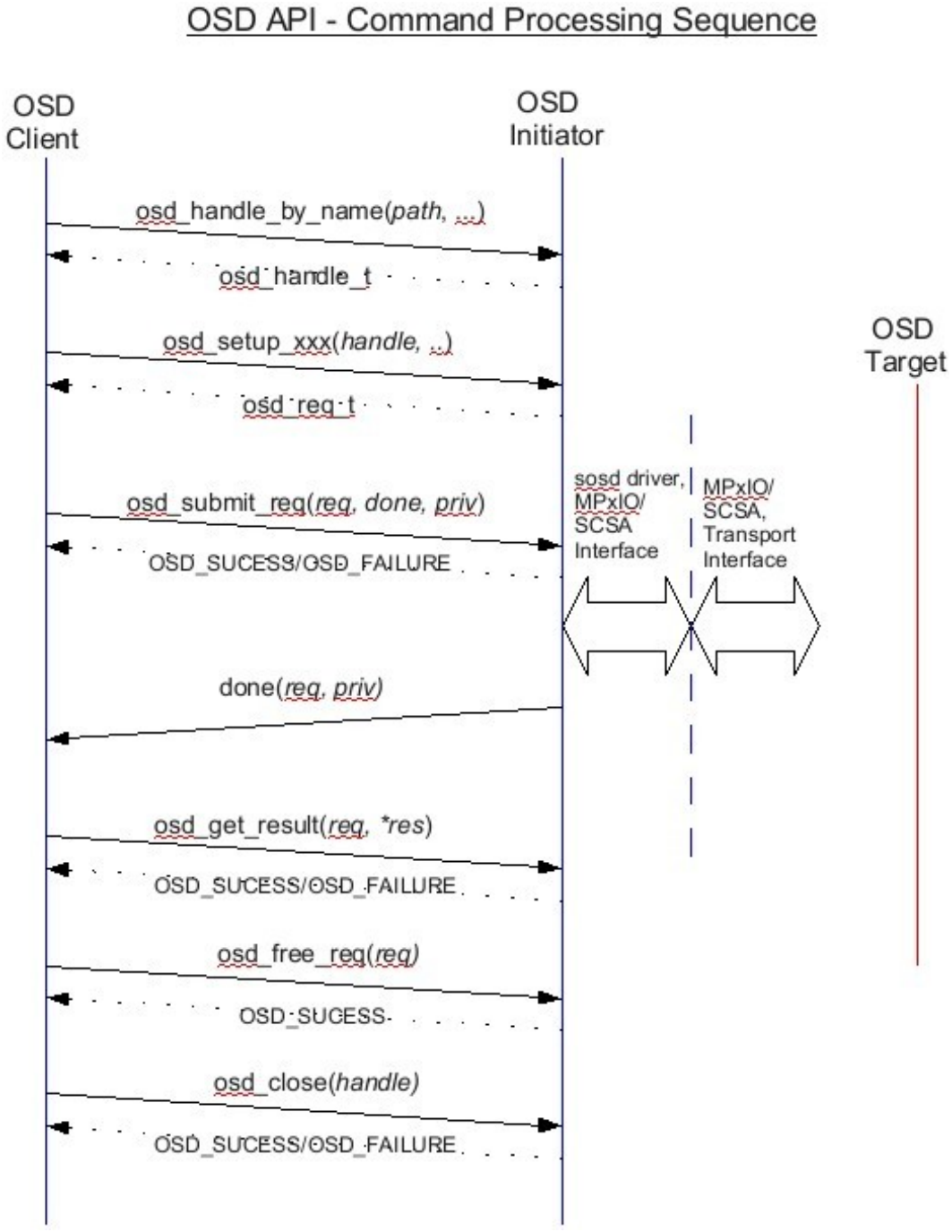
2.2 Component Descriptions

sd, st	Solaris SCSA class target drivers for SCSI block (T10 SBC) devices. sd for disk and st for tape. ses and sgen also 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	New API 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 and link to libosd library to be able to use the API. The following diagram gives an overview of the command sequence with the APIs described in this section.



3.1 Constants

3.1.1 Globals

```
#define      OSD_MAX_CMD_SIZE          256
```

3.1.2 Result Codes

The following are defined values for result_code (in osd_result_t) and return values for the functions: osd_result_t, osd_setup_XXX(), osd_submit_req() & osd_free_req().

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

3.1.3 Error Types

The following are defined values for err_type is osd_result_t.

```
#define OSD_ERRTYPE_NONE        0       /* No error info available; */
                                        /* errp is undefined */
#define OSD_ERRTYPE_SENSE       1       /* SCSI sense data available; errp */
                                        /* contains sense data */
                                        /* This may be set when err_type is */
                                        /* set to OSD_CHECK_CONDITION code. */
#define OSD_ERRTYPE_RESID       2       /* One or more nonzero transfer */
                                        /* residuals; errp points to an */
                                        /* osd_resid_t struct (TBD). This */
                                        /* is set when err_type is set to */
                                        /* the OSD_RESIDUAL code. */
```

3.2 Data Structures

3.2.1 OSD Handle

NAME

osd_handle_t

SYNOPSIS

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

DESCRIPTION

osd_handle_t is a API private value that is passed to the client. The client cannot make any assumptions about the contents of osd_handle_t data type.

3.2.2 OSD Request

NAME

osd_req_t

SYNOPSIS

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

DESCRIPTION

osd_req_t is an index into the data structure pointer that is setup upon client's request. The client cannot make any assumptions about the contents of osd_req_t data type.

3.2.3 OSD Result

NAME

osd_result_t

SYNOPSIS

```
#include <osd.h>
typedef struct osd_result {
      uint8_t     result_code;
      uint8_t     err_type;
      uint16_t    service_action;
      uint64_t    partition_id;
      uint64_t    object_id;
      uint16_t    bad_field_offset;
      void        *errp;
} osd_result_t;
```

DESCRIPTION

osd_result_t is the data structure that contains the result of a failed/incomplete OSD request. It is meant to be used with osd_get_result() API.

Notes: Possible result_code values are defined in Section 3.1.2. errp points to sense data if err_type is OSD_ERRTYPE_SENSE and result_code is OSD_CHECK_CONDITION. The sense data is defined in SPC-3 Section 4.5. OSD specific descriptors are defined in OSD Spec (osd-r10) Section 4.14.2.

3.3 Functions

The API functions are broadly classified into four categories.

- Setup request
- Submit request
- Result request
- Free request

Another API which should be used to OPEN the device is discussed in Section ? of the document. This API uses already existing Solaris mechanisms that are not introduced as part of this project.

3.3.1 Setup OSD Request

The 'Setup' APIs should be used by the client as an initial step in the I/O operation. Before using these APIs, the client needs to obtain the osd_handle by opening the device. See Section ?, for device opening details All APIs return osd_req_t, a driver private structure pointer. See Example in Section ?, for a typical use case.

3.3.1.1 Setup INQUIRY command

```
NAME
      osd_setup_inquiry

SYNOPSIS
      #include <osd.h>
      osd_req_t osd_setup_inquiry(osd_handle_t oh,
          uint64_t len, void *inq_data, uint8_t inq_page)

PARAMETERS
      oh                OSD Handle that was given when device is opened
      len               Length of the inquiry data buffer that is sent
      inq_data          Pointer to the buffer to store inquiry data
      page              Inquiry page for which the request is made

DESCRIPTION
      This API is used to setup an INQUIRY command to OSD device.

RETURN_VALUES
      OSD_SUCCESS       Successful setup
      OSD_FAILURE       General failure
```

3.3.1.2 Setup LIST command

```
NAME
      osd_setup_list

SYNOPSIS
      #include <osd.h>
      osd_req_t osd_setup_list(osd_handle_t oh,
          uint64_t partition_id, uint64_t list_len,
```

```
            uint64_t obj_id, void *list_data);

PARAMETERS
      oh                OSD Handle that was given when device is opened
      partition_id      Partition ID on the OSD device
      len               Length of the list to be retrieved
      obj_id            Object ID on the OSD device
      list_data         Pointer to the buffer to store list data

DESCRIPTION
      This API is used to setup a LIST command to OSD device.

RETURN VALUES
      OSD_SUCCESS       Successful setup
      OSD_FAILURE       General failure
```

3.3.1.3 Setup FORMAT OSD command

```
NAME
      osd_setup_format_osd

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

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

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

RETURN VALUES
      OSD_SUCCESS       Successful setup
      OSD_FAILURE       General failure
```

3.3.1.4 Setup CREATE PARTITION command

```
NAME
      osd_setup_create_partition

SYNOPSIS
     #include <osd.h>
      osd_req_t osd_setup_create_partition(osd_handle_t ohp,
          uint64_t requested_partition_id);

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

DESCRIPTION
```

This API is used to setup a CREATE PARTITION command to OSD device.

RETURN VALUES

OSD_SUCCESS	Successful setup
OSD_FAILURE	General failure

3.3.1.5 Setup REMOVE PARTITION command

NAME

osd_setup_remove_partition

SYNOPSIS

```
#include <osd.h>
osd_req_t osd_setup_remove_partition(osd_handle_t ohp,
    uint64_t partition_id);
```

PARAMETERS

oh	OSD Handle
partition_id	Partition ID to be removed on the OSD device

DESCRIPTION

This API is used to setup a REMOVE PARTITION command to OSD device.

RETURN VALUES

OSD_SUCCESS	Successful setup
OSD_FAILURE	General failure

3.3.1.6 Setup CREATE command

NAME

osd_setup_create_object

SYNOPSIS

```
#include <osd.h>
osd_req_t osd_setup_create_object(osd_handle_t oh,
    uint64_t partition_id, uint64_t requested_object_id,
    uint16_t num_objs);
```

PARAMETERS

oh	OSD 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 API is used to setup a CREATE (object) command to OSD device.

RETURN VALUES

OSD_SUCCESS	Successful setup
OSD_FAILURE	General failure

3.3.1.7 Setup REMOVE command

NAME

osd_setup_remove_object

SYNOPSIS

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

PARAMETERS

oh	OSD 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 API is used to setup a REMOVE (object) command to OSD device.

RETURN VALUES

OSD_SUCCESS	Successful setup
OSD_FAILURE	General failure

3.3.1.8 Setup CREATE AND WRITE command

NAME

osd_setup_create_and_write

SYNOPSIS

```
#include <osd.h>
osd_req_t osd_setup_create_and_write(osd_handle_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 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 API is used to setup a CREATE AND WRITE (object) command to OSD device.

RETURN VALUES

OSD_SUCCESS	Successful setup
OSD_FAILURE	General failure

3.3.1.9 Setup WRITE command

NAME

osd_setup_write

SYNOPSIS

```
#include <osd.h>
osd_req_t osd_setup_write(osd_handle_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 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 API is used to setup a WRITE command the OSD device.

RETURN_VALUES

OSD_SUCCESS	Successful setup
OSD_FAILURE	General failure

3.3.1.10 Setup WRITE pagelist command

NAME

osd_setup_write_pagelist

SYNOPSIS

```
#include <osd.h>
osd_req_t osd_setup_write_pagelist(osd_handle_t oh,
    uint64_t partition_id, uint64_t object_id, uint64_t len,
    uint64_t starting_byte_addr, int data_offset,
    int pagelist_count, page_t *pagelist);
```

PARAMETERS

oh	OSD 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)
data_offset	Offset into page list from where data needs to be mapped
pagelist_count	Number of page_t structures setup in pagelist argument

```
      pagelist              Pointer to the pagelist structure

DESCRIPTION
      This API is used to setup a WRITE command the OSD device,
      where pages are given for buf mapping.

RETURN_VALUES
      OSD_SUCCESS       Successful setup
      OSD_FAILURE       General failure
```

3.3.1.11 Setup READ command

```
NAME
      osd_setup_read

SYNOPSIS
      #include <osd.h>
      osd_req_t osd_setup_read(osd_handle_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 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 API is used to setup a READ command to OSD device.

RETURN_VALUES
      OSD_SUCCESS       Successful setup
      OSD_FAILURE       General failure
```

3.3.1.12 Setup READ pagelist command

```
NAME
      osd_setup_read_pagelist

SYNOPSIS
      #include <osd.h>
      osd_req_t osd_setup_read_pagelist(osd_handle_t oh,
          uint64_t partition_id, uint64_t object_id, uint64_t len,
          uint64_t starting_byte_addr, int data_offset,
          int pagelist_count, page_t *pagelist);

PARAMETERS
      oh                    OSD 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 read should commence in the specified object relative to 1st byte(byte 0)
data_offset	Offset into page list from where data needs to be mapped
pagelist_count	Number of page_t structures setup in pagelist argument
pagelist	Pointer to the pagelist structure

DESCRIPTION

This API is used to setup a READ command the OSD device, where pages are present in buf mapping.

RETURN VALUES

OSD_SUCCESS	Successful setup
OSD_FAILURE	General failure

3.3.1.13 Setup APPEND command

NAME

osd_setup_append

SYNOPSIS

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

PARAMETERS

oh	OSD Handle
partition_id	Partition ID on the OSD device
object_id	Object ID to be written 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 API is used to setup an APPEND command the OSD device.

RETURN VALUES

OSD_SUCCESS	Successful setup
OSD_FAILURE	General failure

3.3.1.14 Setup SET ATTRIBUTES command (1 attribute in PAGE format)

NAME

osd_setup_set_1page_attr

SYNOPSIS

```
#include <osd.h>
osd_req_t osd_setup_set_1page_attr(osd_handle_t oh,
```

```
            uint64_t partition_id, uint64_t object_id, uint32_t attr_page,
            uint32_t attr_num, uint32_t attr_len, char *attr_val);
```

PARAMETERS

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

This API is used to setup a SET ATTRIBUTES cmd. Specifally used to set a single attribute with value less than (or equal to) 18 bytes. The attribute value is stored directly in the CDB, without having to use a Data-out buffer.

RETURN_VALUES

OSD_SUCCESS	Successful setup
OSD_FAILURE	General failure

NOTES

If attr_len > 18, the command is terminated with OSD_CHECK_CONDITION with Sense key set to ILLEGAL REQUEST and additional sense code set to INVALID FIELD IN CDB.

3.3.1.15 Setup SET/GET ATTRIBUTES

NAME

osd_setup_set_get_page_attr

SYNOPSIS

```
#include <osd.h>
osd_req_t osd_setup_set_get_1page_attr(osd_handle_t oh,
    uint64_t partition_id, uint64_t object_id, uint8_t priority,
    uint32_t get_attr_page, uint32_t get_attr_alloc_len,
    uint32_t *ret_attr,
    uint32_t set_attr_page, uint32_t set_attr_num,
    uint32_t set_attr_len, uint32_t *set_attr_val);
```

PARAMETERS

	oh	OSD Handle
	partition_id	Partition ID on the OSD device
	object_id	Object ID to lookup
	priority	GET/SET operation to be performed first
+	get_attr_page	Atribute page number to be retrieved
+	get_attr_alloc_len	Memory allocated for attribute value being retrieved.
+	ret_attr	Pointer to place the retrieved attr value
+	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

Please see NOTES section for limitations.
This API is used to setup GET ATTRIBUTES AND/OR SET ATTRIBUTES in PAGE format. Specifically used to SET a single attribute and/or GET a SINGLE attribute. The priority field determines which operation needs to be performed FIRST when both GET and SET operations are requested. If get_attr_alloc_len is not 0 and the command completes successfully, the retrieved attribute will be available in ret_attr location.

\+ If arguments indicated with + sign are zero or NULL the API assumes that a following osd_add_* API call will setup the request completely. If arguments are non-zero, the API performs the operation and subsequent osd_add_* calls will fail the setup.

RETURN_VALUES

OSD_SUCCESS	Successful setup
OSD_FAILURE	General failure

NOTES

Although, the API describes simultaneous GET/SET operations, due to limitation in the current SCSI framework, only one operation per direction is supported at this time. Clients can use this API as long as they have either set_attr_len or get_attr_alloc_len as 0. When bi-directional support is introduced, this limitation will be removed and the API can be used as described.

3.3.1.16 Add SET PAGE ATTRIBUTES to a Request

NAME

osd_add_set_page_attr_to_req

SYNOPSIS

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

Please see NOTES section for limitations.
This API is used to ADD SET ATTRIBUTES in PAGE format to an OSD OSD request that has already been setup.

RETURN_VALUES

OSD_SUCCESS	Successful setup

```
      OSD_FAILURE       General failure

NOTES
      Note that the 1st argument is a osd_req_t, unlike osd_handle_t that
      is present in osd_setup_XXX() APIs.

      Currently, since bi-directional commands are not supported, the API
      will fail if a GET request was previously setup corresponding to
this osd_req_t.
```

3.3.1.17 Add GET PAGE ATTRIBUTES to a Request

```
NAME
      osd_add_get_page_attr_to_req

SYNOPSIS
     #include <osd.h>
      osd_req_t osd_add_get_page_attr_to_req(osd_req_t req,
          uint32_t get_attr_page, uint32_t get_attr_alloc_len);

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.

DESCRIPTION
      Please see NOTES section for limitations.
      This API is used to ADD GET ATTRIBUTES in PAGE format to an OSD
      OSD request that has already been setup.

RETURN_VALUES
      OSD_SUCCESS       Successful setup
      OSD_FAILURE       General failure

NOTES
      Note that the 1st argument is a osd_req_t, unlike osd_handle_t that
      is present in osd_setup_XXX() APIs.

      Currently, since bi-directional commands are not supported, the API
      will fail if a SET request was previously setup corresponding to
      this osd_req_t.
```

3.3.1.18 Extract PAGE ATTRIBUTES from a completed Request

```
NAME
      osd_extract_page_attr_from_req

SYNOPSIS
     #include <osd.h>
      int osd_extract_page_attr_from_req(osd_req_t req,
          uint32_t *attr_page, uint32_t attr_alloc_len,
          uint32_t *attr_val);
```

PARAMETERS

req	OSD Request that is already setup using one of the setup APIs.
attr_page	Atribute page number to be retrieved. If this is NULL, 1st page is retrieved. To retrieve subsequent attributes, the client should give the previous page that was retrieved.
attr_alloc_len	Memory allocated for attribute value being retrieved.
attr	Pointer to place the retrieved attr value

DESCRIPTION

Please see NOTES section for limitations.
This API is used to EXTRACT ATTRIBUTES in PAGE format from an OSD request that completed.

RETURN VALUES

OSD_SUCCESS	Successful setup
OSD_FAILURE	General failure

NOTES

Note that the 1st argument is a osd_req_t, unlike osd_handle_t that is present in osd_setup_XXX() APIs.

Currently, since bi-directional commands are not supported, the API will fail if a SET request was previously setup corresponding to this osd_req_t.

3.3.1.19 Setup SET/GET ATTRIBUTES (Multiple attributes LIST format)

NAME

osd_setup_set_get_list_attr

SYNOPSIS

```
#include <osd.h>
osd_req_t osd_setup_set_get_list_attr(osd_handle_t oh,
    uint64_t partition_id, uint64_t object_id, uint8_t priority,
    uint32_t get_attr_list_len, uint32_t *get_attr_list_off,
    uint32_t get_attr_alloc_len, uint32_t *get_attr_off,
    uint32_t set_attr_list_len, uint32_t *set_attr_list_off);
```

PARAMETERS

oh	OSD Handle
partition_id	Partition ID on the OSD device
object_id	Object ID to lookup
priority	GET/SET operation to be performed first
get_attr_list_len	Atribute page number to be retrieved
get_attr_list_off	Memory allocated for attribute value being retrieved.
get_attr_alloc_len	Page number of attribute value to Set
get_attr_off	Attribute number with attribute page
set_attr_list_len	Length of attribute value in bytes
set_attr_list_off	Value of attribute to set.

DESCRIPTION

Please see NOTES section for limitations.
This API is used to setup GET ATTRIBUTES AND/OR SET ATTRIBUTES in LIST format. Specifically used to SET or GET attribute lists.
The priority field determines which operation needs to be performed FIRST when both GET and SET operations are requested.

RETURN VALUES

OSD_SUCCESS	Successful setup
OSD_FAILURE	General failure

NOTES

Although, the API describes simultaneous GET/SET operations, due to limitation in the current SCSI framework, only one operation per direction is supported at this time. Clients can use this API as long as they have either set_attr_len or get_attr_alloc_len as 0. When bi-directional support is introduced, this limitation will be removed and the API can be used as described.

Additionally, this API requires the knowledge of LIST format in the OSD spec to the client. This will be modified (by adding more LIST operation) APIs that aid the client in the setup.

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 *), void *ct_priv,
    uint32_t timeout, uint8_t retries);
```

PARAMETERS

req	osd_req_t private data as returned by osd_setup_XXX APIs
done	routine to callback into client after completion
ct_priv	Client private data that is sent as 2nd argument to callback routine for async requests. Synchronous callers must use NULL.
timeout	Seconds after which the driver can timeout the request (Async requests' argument is ignored)
nretries	Number of retries allowed by the driver.

DESCRIPTION

The osd_submit_req() API should be used following the osd_setup_xxx API by the client. This API sends the command to the underlying transport. The osd_setup_xxx() APIs only form the CDBs as per the OSD specification. If a callback routine is provided, it will be invoked by the driver on the completion of the request. Kernel callers MUST register a callback. User mode clients MUST send NULL, and the command blocks until completion in user mode.

```
RETURN_VALUES
      OSD_SUCCESS             Successful completion
      OSD_FAILURE             General failure
```

3.3.3 Get Result of OSD Request

```
NAME
      osd_result_t

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

PARAMETERS
      req         osd_req_t private data that was used to submit the request
      result      osd_result_t structure with the result of the request

DESCRIPTION
      This API would be used by the client to obtain the result of a
      submitted request that is failed or incomplete. Successful requests
      just give OSD_SUCCESS as return value for osd_submit_req() API.

      Result structure defined as osd_result_t.
     #include <osd.h>
     typedef struct osd_result {
           uint8_t     result_code;
           uint8_t     err_type;
           uint16_t    service_action;
           uint64_t    partition_id;
           uint64_t    object_id;
           uint16_t    bad_field_offset;
           void        *errp;
     } osd_result_t;

      Result_Code (result_code) Definitions:
      OSD_SUCCESS             Successful completion
      OSD_FAILURE             General failure
      OSD_CHECK_CONDITION     A SCSI check condition occurred
                               while performing the request.
      OSD_INVALID             An invalid parameter was detected
                              in the request.
      OSD_TOOBIG              The requested data transfer is
                              too large for the system.
      OSD_BADREQUEST          The requested operation cannot be
                              performed due to an unrecoverable
                              error in the request
      OSD_LUN_ERROR           The requested operation cannot be
                              performed due to an unrecoverable
                              error in the LUN
      OSD_LUN_FAILURE         The LUN has become permanently
                              unusable or inaccessible
      OSD_BUSY                The requested operation cannot be
                              performed at this time
```

```
        OSD_RESERVATION_CONFLICT A SCSI Reservation conflict was
                                  encountered at the LUN
        OSD_RESIDUAL             One or more data transfers for the
                                 request have a nonzero residual
        OSD_NORESOURCES          Insufficient resources available
                                 to complete the request

        Error Type (err_type) Definitions:
        OSD_ERRTYPE_NONE         No error info available;
                                 errp is undefined
        OSD_ERRTYPE_SENSE         SCSI sense data available; errp contains
                                 sense data. This may be set when err_type
                                 is OSD_CHECK_CONDITION.
        OSD_ERRTYPE_RESID        One or more nonzero transfer residuals;
                                 residuals; errp points to an osd_resid_t
                                 struct (TBD). This is set when err_type is
                                 set to the OSD_RESIDUAL code.
RETURN VALUES
        OSD_SUCCESS              Successful completion
        OSD_FAILURE              General failure
```

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 private data that was used to submit the request

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

RETURN VALUES
      OSD_SUCCESS              Successful completion.
```

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 file pathname. This API is not a standard way to retrieve the device handle and could be used by user-mode clients, if required.

```
int osd_handle_by_name(char *pathname, int flags, cred_t *cr,
    osd_handle_t &oh);
```

Notes:
pathname - absolute path to the device (/dev/osd/..)
flags - FEXCL, FNDELAY, FREAD, FWRITE
cr - Pointer to the credentials structure.
oh - pointer an osd_handle_t that will be filled in by a successful open.

3.4.2 CLOSE

API to Close the device.

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

3.5 Miscellaneous Utility APIs

The following are a few general purpose APIs 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 API returns the maximum allowed DMA transfer size (in bytes) so that client can limit the I/O to the specified limit.

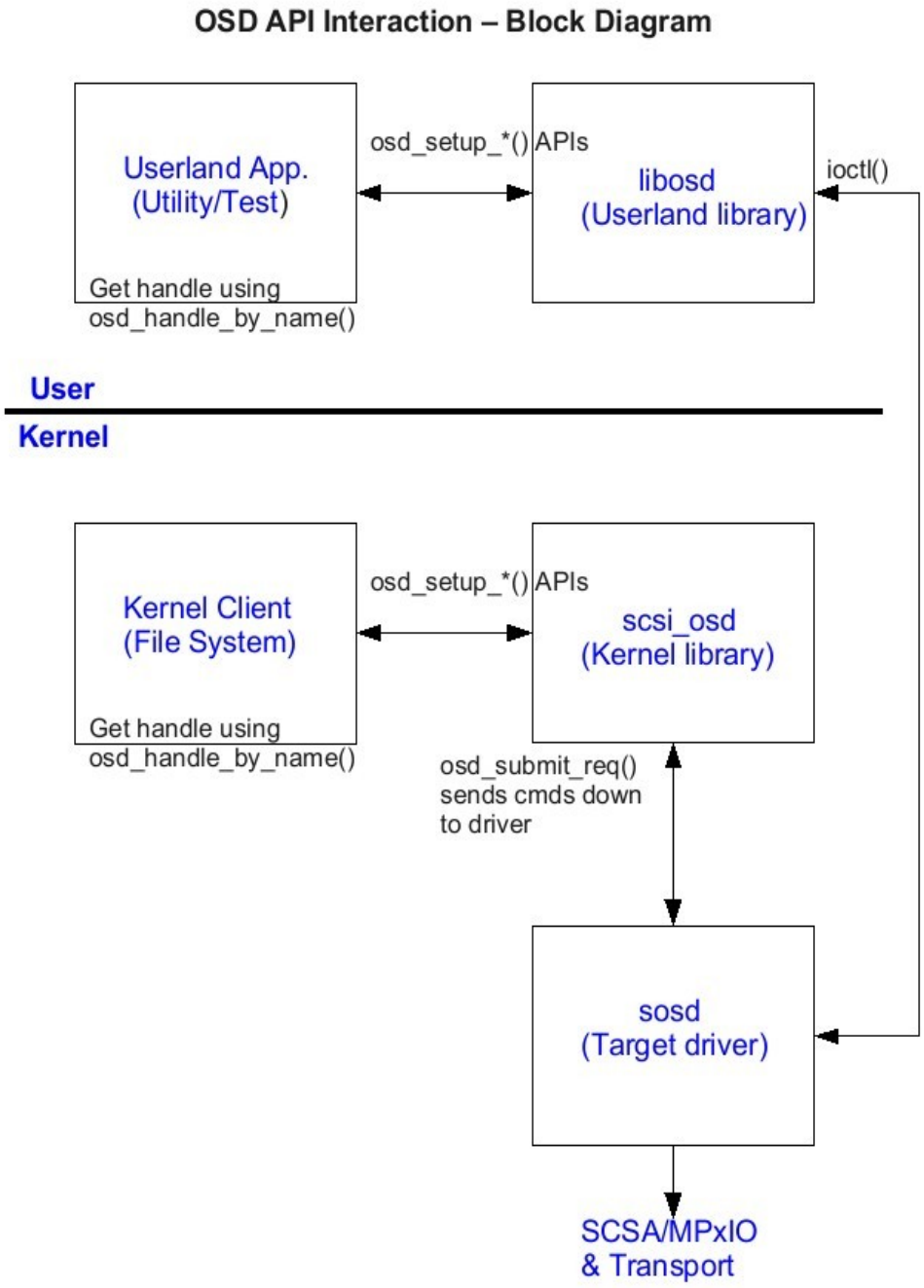
```
int osd_get_max_dma_size(osd_handle_t oh);
```

3.6 Driver, Library and Loadable Module Interaction

Commands sent from a user mode client are processed synchronously without the requirement for a callback routine. Asynchrnous processing with a client callback, followed by an API to fetch the result is done only in kernel mode.

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

<TBD: Details of libosd – Userland version of the same API>


OSD API Interaction – Block Diagram
Userland App.
(Utility/Test)
Get handle using
osd_handle_by_name()
osd_setup_*() APIs
libosd
(Userland library)
ioctl()
User
Kernel
Kernel Client
(File System)
Get handle using
osd_handle_by_name()
osd_setup_*() APIs
scsi_osd
(Kernel library)
osd_submit_req()
sends cmds down
to driver
sosd
(Target driver)
SCSA/MPxIO
& Transport

3.7 API Use Case (Kernel mode)

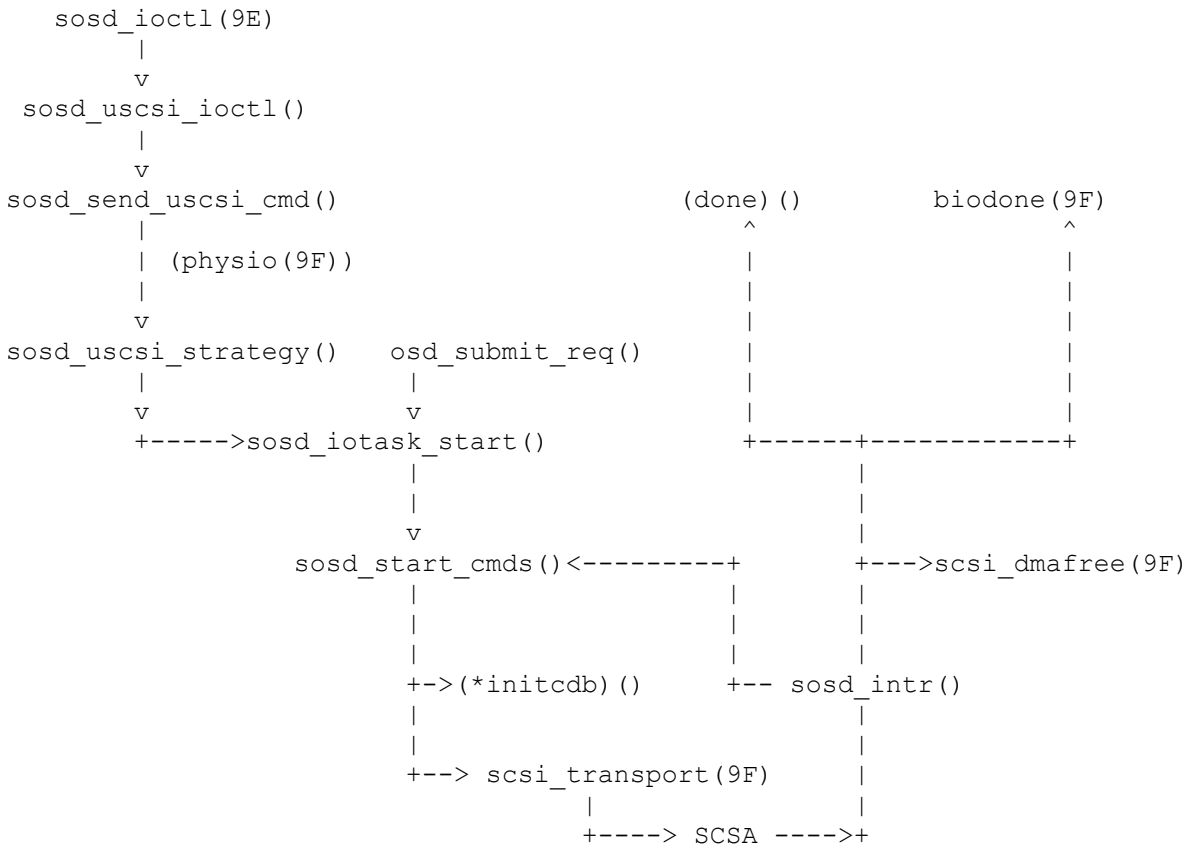
```
int done(osd_req_t *req, void *ct_priv);

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              *ct_priv;
      cred_t            *cred;
      ldi_ident_t       li;
      ldi_handle_t      lh;
      dev_t             devp;
      osd_req_t         req;
      osd_result_t      *res;
      osd_handle_t      handle;


      /* Get handle using device pathname */
      rval = osd_handle_by_name(
          "/dev/osd/osd010000144F1D837C00002A00479FAC04,root",
          FREAD | FWRITE, CRED(), &handle);

      /*
       * 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, ct_priv) != OSD_SUCCESS) {
            syslog(LOG_ERR, "osd_submit_req error: Command failed.\n", err);
      }

}
```

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

      res = kmem_zalloc(sizeof (osd_result_t), KM_SLEEP);

      err = osd_get_result(req, res);

      if (err == OSD_SUCCESS) && (res->result_code != OSD_SUCCESS) {
            /* Process the result data to find out what went wrong */
             if (res->err_type == OSD_CHECK_CONDITION) {
                  /* Interpret res->errp sense data and retry setup+submit */
            }
      }

      kmem_free(res, sizeof (osd_result_t));

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

      return (0);
}
```

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 would be created along with a symbolic link in the /dev/osd directory. This puts the theatrical limit of OSD luns supported by a single node to 2 ^ 18 .

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 an 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/osd2r02.pdf
- OpenSolaris web page of OSD project: http://opensolaris.org/os/project/osd/