NAME

generate_new_lvid, create_vol, destroy_vol, add_logical_id_index, has_logical_id_index, get_volume_quota, print_lid_index, vol_root_index, get_du_statistics − Class ss_m Methods for Volume Management

SYNOPSIS

```
#include <sm_vas.h>  // which includes sm.h

static rc_t                  generate_new_lvid(lvid_t& lvid);
static rc_t             create_vol(
    const char*             device_name,
    const lvid_t&           lvid,
    uint4                   quota_KB,
    bool                    skip_raw_init = false,
    vid_t                   local_vid = vid_t::null);
static rc_t             destroy_vol(const lvid_t& lvid);
static rc_t             add_logical_id_index(
    const lvid_t&                lvid,
    uint4                   reserved_local,
    uint4                   reserved_remote);
static rc_t             has_logical_id_index(
    const lvid_t&                lvid,
    bool&                   has_index);
static rc_t                   get_volume_quota(
    const lvid_t&                 lvid,
    smksize_t&                    quota_KB,
    smksize_t&                    quota_used_KB);
static rc_t             print_lid_index(const lvid_t& lvid);

/* Logical-ID version */
static rc_t                   vol_root_index(
    const lvid_t&                 v,
    serial_t&                     liid);

/* Phsical-ID version */
static rc_t                   vol_root_index(
    const vid_t&                        v,
    stid_t&                             iid);
static rc_t                   vol_root_index(
    const vid_t&                        v,
    stid_t&                             iid);


// Volume space utilization statistics

static rc_t             get_du_statistics(
    lvid_t                  vid,
    sm_du_stats_t&                du,
    bool                    audit = TRUE);
static rc_t             get_du_statistics(
    const lvid_t&                vid,
    const serial_t&              serial,
    sm_du_stats_t&               du,
```

```
    bool                    audit = TRUE);
/* Physical-ID version */
static rc_t             get_du_statistics(
    vid_t                   vid,
    sm_du_stats_t&                du,
    bool                    audit = TRUE);
static rc_t             get_du_statistics(
    const stid_t&                 stid,
    sm_du_stats_t&                du,
    bool                    audit = TRUE);
```

DESCRIPTION

The above class **ss_m** methods manage volumes.

Volumes are a logical unit of storage that are mapped to devices, which are physical units of storage (corresponding to disks or disk partitions).

A volume is identified uniquely and persistently by a logical volume ID (lvid_t). Volumes can be used whenever the device they are located on is mounted by the SSM. Volumes have a quota. The sum of the quotas of all the volumes on a device cannot exceed the device quota. Volumes are located on devices. Device management methods are described in **device(ssm).**

The basic steps to begin using a new volume are:

format_dev():
initialize the device

mount_dev(): allow use of the device

generate_new_lvid: generate a unique ID for the volume

create_vol: create a volume on the device

add_logical_id_index: add logical ID facility to the volume

VOLUMES INITIALIZATION METHODS

generate_new_lvid(lvid)

The **generate_new_lvid** method generates a universally unique volume id and returns it via *lvid.* Currently, the ID is generated using the network address of the server combined with a timestamp.

create_vol(device_name, lvid, quota_KB, skip_raw_init, local_vid)

The **create_vol** method create_vol creates and formats a new volume on a device. When a volume is stored on a raw device, formatting it involves the time consuming step of zero-ing every page. This is necessary for correct operation of recovery. In some situations (during testing, for example), this zeroing is unnecessary. In this case, setting *skip_raw_init* to **true** disables the zeroing. Creating a volume make the volume available for use. The *local_vid* parameter is only meant to be a temporary hack for those VASs using the physical ID version of the SSM interface. Local_vid is used to specify the local handle that should be when a volume is mounted. The default value vid_t::null indicates that the SSM can use any number it wants to use. **Note:** currently there is a limit of one volume per device.

destroy_vol(lvid)

The **destroy_vol** method destroys a volume on a device. After a **destroy_vol** the device remains mounted and another volume can be created on the device.

add_logical_id_index(lvid, reserved_local, reserved_remote)

The **add_logical_id_index** method sets up the logical ID index on volume *lvid* and should be called after **create_vol.** The logical ID index is used to map **logical ID serial numbers,** type **serial_t,** to physical locations on the volume or to IDs on other volumes. The *reserved_local* parameter reserves a certain number of intra-volume (local) serial numbers. The *reserved_remote* parameter reserves inter-volume serial numbers. The reserved serial numbers will not be allocated by any calls which generate serial numbers and therefore can be used for other things by the VAS.

has_logical_id_index(lvid, has_index)

The **has_logical_id_index** method sets *has_index* to **true** if volume *lvid* contains a logical ID index.

print_lid_index(lvid)

The **print_lid_index** method is a debugging function that prints the logical ID index to standard output.

ROOT INDEX METHODS

The root index of a volume is a special B+tree index available on every volume. It can be used to store hooks (roots) into the data on a volume. A common use of a this index is to associate a string name with a record, index or file ID containing information about the contents of the volume. For example, in a database system, this might be the ID for the catalog. The index is accessed just like any other B+tree index. See **btree(ssm)** for more information. **Note:** keys with the prefix "SSM_RESERVED" are reserved for use by the SSM.

vol_root_index(lvid, serial)

The **vol_root_index** method returns (in *serial)*
the serial number (logical ID) of the **root index** for volume *lvid.*

SPACE UTILIZATION METHODS

The following methods provide disk space utilization statistics for volumes, files, and indexes.

get_volume_quota(lvid, quota_KB, quota_used_KB)

The **get_volume_quota** method returns the quota (in K-bytes) in *quota_KB* and the amount of the quota allocated in *quota_used_KB,* for volume *lvid.*

get_du_statistics(lvid, du, audit)

The **get_du_statistics** method gathers space utilization statistics for volume *lvid.* The use of "du" stems from similarity, in purpose, to the "du" command found on some operating systems. The statistics are returned in the *du* parameter. When the *audit* parameter is set to **true,**
the entire volume is share (SH) locked and the statistics are audited for correctness. The error code **fcINTERNAL** will be returned at the first sign of an auditing problem. If **fcINTERNAL** is

returned it indicates either there is a problem with the integrity of the volumes data structures (possibly indicating inaccessible garbage) or there is a bug in the auditing code. When the *audit* parameter is set to **false,** only an intention-share (IS) locks are obtained on the volume and all files and indexes. Therefore the statistics gathering methods may not see a consistent version of the volume as things can be changing while statistics are gathered.

get_du_statistics(lvid, serial, du, audit)

The **get_du_statistics** method gathers space utilization statistics for a specific file or index indicated by the logical ID: *lvid, serial.* The statistics are returned in the *du* parameter. The *audit* parameter works as described in the previous methods except that it is the index or file that is SH locked when *audit* is **true.**

ERRORS

All of the above methods return a **w_rc_t** error code.

See **errors(ssm)** for more information on error handling.

TRANSACTION ISSUES

Many of the above methods cannot be run within the scope of a transaction. The reason for this restriction is to avoid the implication that rolling back (aborting) the transaction would rollback the effect of the method.

TODO

EXAMPLES

TODO

VERSION

This manual page applies to Version 2.0 of the Shore Storage Manager.

SPONSORSHIP

The Shore project is sponsored by the Advanced Research Project Agency, ARPA order number 018 (formerly 8230), monitored by the U.S. Army Research Laboratory under contract DAAB07-91-C-Q518. Further funding for this work was provided by DARPA through Rome Research Laboratory Contract No. F30602-97-2-0247.

SEE ALSO

intro(ssm), device(ssm).