NAME

bulkld_md_index, create_md_assoc, create_md_index, destroy_md_assoc, destroy_md_index, find_md_assoc, print_md_index, draw_rtree rtree_stats − Class ss_m Methods for R*tree (multi-dimensional) Index Operations

SYNOPSIS

```
#include <sm_vas.h>  // includes sm.h (where they are declared)

    /* Logical-ID version */
static rc_t                 create_md_index(const lvid_t&   lvid,
            ndx_t                   ntype,
            store_property_t              property,
            serial_t&               liid,
            int2                    dim=2);

    /* Physical-ID version */
static rc_t                 create_md_index(vid_t           vid,
            ndx_t                   ntype,
            store_property_t              property,
            stid_t&                       stid,
            int2_t                        dim = 2,
            const serial_t&               logical_id=serial_t::null);

    /* Logical-ID version */
static rc_t                 destroy_md_index(const lvid_t&  lvid,
            const serial_t&               liid);
    /* Physical-ID version */
static rc_t                 destroy_md_index(const stid_t& iid);


    /* Logical-ID version */
static rc_t                 bulkld_md_index(const lvid_t&   lvid,
            const serial_t&               liid,
            const lvid_t&                 s_lvid,
            const serial_t&               s_liid,
            sm_du_stats_t&                stats,
            int2                    hff=65,
            int2                    hef=120,
            nbox_t*                       universe=NULL);

    /* Physical-ID version */
static rc_t                 bulkld_md_index(const stid_t&   stid,
            const stid_t&                 source,
            sm_du_stats_t&                stats,
            int2_t                        hff=75,
            int2_t                        hef=120,
            nbox_t*                       universe=NULL);

    // version of above that takes multiple input files
static rc_t                 bulkld_md_index(const stid_t&   stid,
            int                     nsrcs,
            const stid_t*                 source,
            sm_du_stats_t&                stats,
```

```
            int2_t                      hff=75,
            int2_t                      hef=120,
            nbox_t*                     universe=NULL);


    /* Logical-ID version */
static rc_t                 bulkld_md_index(const lvid_t&   lvid,
            const serial_t&             liid,
            sort_stream_i&              sorted_stream,
            sm_du_stats_t&              stats,
            int2                    hff=65,
            int2                    hef=120,
            nbox_t*                     universe=NULL);

    /* Physical-ID version */
static rc_t                 bulkld_md_index(const stid_t&   stid,
            sort_stream_i&              sorted_stream,
            sm_du_stats_t&              stats,
            int2_t                      hff=75,
            int2_t                      hef=120,
            nbox_t*                     universe=NULL);


    /* Logical-ID version */
static rc_t                 create_md_assoc(const lvid_t&   lvid,
            const serial_t&             liid,
            const nbox_t&               key,
            const vec_t&                el);

    /* Physical-ID version */
static rc_t                 create_md_assoc(stid_t          stid,
            const nbox_t&               key,
            const vec_t&                el);


    /* Logical-ID version */
static rc_t                 destroy_md_assoc(const lvid_t&  lvid,
            const serial_t&             liid,
            const nbox_t&               key,
            const vec_t&                el);

    /* Physical-ID version */
static rc_t                 destroy_md_assoc(stid_t   stid,
            const nbox_t&               key,
            const vec_t&                el);


    /* Logical-ID version */
static rc_t                 find_md_assoc(const lvid_t&     lvid,
            const serial_t&             liid,
            const nbox_t&               key,
            void*                   el,
            smsize_t&               elen,
```

```
            bool&                   found);

    /* Physical-ID version */
static rc_t                 find_md_assoc(stid_t           stid,
            const nbox_t&               key,
            void*                   el,
            smsize_t&               elen,
            bool&                   found);


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

    /* Physical-ID version */
static rc_t                 print_md_index(stid_t stid);


    /* Logical-ID version */
static rc_t                 draw_rtree(const lvid_t&  lvid,
            const serial_t&             liid);
    /* Physical-ID version */
static rc_t                 draw_rtree(const stid_t& stid, ostream &);


    /* Logical-ID version */
static rc_t                 rtree_stats(const lvid_t& lvid,
            const serial_t&             liid,
            rtree_stats_t&              stat,
            uint2                   size = 0,
            uint2*                  ovp = NULL,
            bool                    audit = false);

    /* Physical-ID version */
static rc_t                 rtree_stats(const stid_t& stid,
            rtree_stats_t&              stat,
            uint2_t                     size = 0,
            uint2_t*                    ovp = NULL,
            bool                    audit = false);
```

DESCRIPTION

The above class **ss_m** methods all deal with manipulating multi-dimensional (md) indexes. So far, the only type of multi-dimension index provided by the SSM is the R*tree. See **The Shore Storage Manager Programming Interface** for more information on R*trees.

Common Parameters

There are a number of common parameters for these methods:

lvid Logical volume ID of volume containing an index.

liid Logical index ID, the serial number of an index.

key A n-dimensional box, **nbox_t(common),** that is the key portion of an index entry.

el A vector pointing to the element portion of an index entry.

create_md_index(lvid, ntype, property, liid, dim)

The **create_md_index** method creates a new B+tree index on the volume *lvid,* and returns its serial number in *liid.* The *ntype* parameter specifies the type of implementation used for the index. The only valid value for the *ntype* parameter is **t_rtree,** indicating an R*tree. The *property* parameter specifies whether the index is temporary or not. See **enum(ssm)** for more information on **store_property_t.** The *dim* parameter specifies the number of dimensions for the index. **Note:** only 2 dimensions are currently supported.

destroy_md_index(lvid, liid)

The **destroy_index** method destroys the index and deallocates all space used by it. The space is not available for reuse until the transaction destroying the index commits.

bulkld_md_index(lvid, liid, s_lvid, s_lfid, stats, hff, hef, universe)

This **bulkld_md_index** method bulk loads the **empty** index, identified by *lvid* and *liid.* The entries to load are located, in sorted order, in the file identified by *s_lvid* and *s_lfid.* The header of each record in the file contains the key (see **nbox_t(common)).** and the body contains the element (value) associated with the key. This file must have been sorted by **sort(ssm)** using the **t_spatial** key type to get a spatial linear order (Hilbert curve). Statistics for the newly loaded index are returned in *stats,* specifically in the *rtree field.*

The *hff* parameter is a heuristic fill factor and *hef* is a heuristic expansion factor. They are used to determine when an Rtree page should stop accepting new entries to reduce the degree of overlap during bulk loading. Since bulk loading requires some linear order to map 2-d keys to 1-d disk locations, there definitely will be some loss of spatial clustering. Thus, packing entries 100% to a rtree page could result in a very large overlap between leaf pages. These heuristics parameters are designed to minimize this problem. It is recommended that the default values always be used. The default values are chosen to be the best on the average case. But they not guaranteed best for the worst case (skewed data).

The VA universe parameter is a "box" specifying the boundaries of the "space" containing the loaded keys. A more compact index can be built if this parameter is provided, but it is not necessary.

bulkld_md_index(lvid, liid, sorted_stream, stats, hff, hef, universe)

This **bulkld_md_index** method is identical to the one above except that rather than getting entries from a file, the entries come from *sorted_stream.* **Note:** this method has not been extensively tested and may change in the future. See **sort_stream_i(ssm)** for more information.

create_md_assoc(lvid, liid, key, el)

The **create_md_assoc** method adds a new entry associating *key* with the element (value) *el.*

destroy_md_assoc(lvid, liid, key, el)

The **destroy_md_assoc** method destroys the entry associating *key* with the element (value) *el.*

find_md_assoc(lvid, liid, key, el, elen, found)

The **find_assoc** method finds *key* in the index and and writes the associated element (only the first one found) to the address specified by *el.* At most *elen* bytes will be written. If the element is not needed, set *elen* to 0. If *key* is found, then *found* will be set to **true.** A more comprehensive lookup facility, allowing range searches, is available from the class .FN scan_rt_i described in .SA scan_rt_i(ssm)

print_md_index(lvid, liid)

The **print_md_index** method is prints the contents of the index. It is meant to be a debugging utility.

draw_rtree(lvid, liid)

The **draw_rtree** method generates a "gremlin" file for visualizing an R*-tree graphically. This method is an unsupported debugging utility.

rtree_stats(lvid, liid, stat, size, ovp, audit)

The **rtree_stats** method is an unsupported debugging utility for gathering more detailed statistics on an R*tree. The *stats* parameter is filled with the regular Rtree stats gathered by **ss_m::get_du_statistics().** The *ovp* parameter is an array that will be filled with overlap percentage for each level of the R*tree. The *size* parameter is the size of the array. If the *audit* parameter is **true,** the stats structure is audited.

ERRORS

All of the above methods return a **w_rc_t** error code. If an error occurs during a method that is updating persistent data (the create, destroy, and bulk load method will update data) then the index could be in an inconsistent state. The caller then has the choice of aborting the transaction or rolling back to the nearest save-point (see **transaction(ssm)**).

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

EXAMPLES

To Do.

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.

COPYRIGHT

SEE ALSO

scan_rt_i(ssm) sort_stream_i(ssm) intro(ssm)