

NAME

bulkld_index, create_assoc, create_index, destroy_assoc, destroy_index, find_assoc, print_index – Class ss_m Methods for B+Tree Index Operations

SYNOPSIS

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

static rc_t create_index(
    const lvid_t&      lvid,
    ndx_t             ntype,
    store_property_t   property,
    const char*        key_desc,
    concurrency_t      cc,
    uint              size_kb_hint, // hint on final size in KB
    serial_t&          liid);

// for backward-compatibility:
static rc_t create_index(
    const lvid_t&      lvid,
    ndx_t             ntype,
    store_property_t   property,
    const char*        key_desc,
    uint              size_kb_hint, // hint on final size in KB
    serial_t&          liid);

static rc_t destroy_index(
    const lvid_t&      lvid,
    const serial_t&    liid);

static rc_t bulkld_index(
    const lvid_t&      lvid,
    const serial_t&    liid,
    const lvid_t&      s_lvid,
    const serial_t&    s_lfid,
    sm_du_stats_t&     stats);

static rc_t bulkld_index(
    const lvid_t&      lvid,
    const serial_t&    liid,
    sort_stream_i&     sorted_stream,
    sm_du_stats_t&     stats);

static rc_t print_index(
    const lvid_t&      lvid,
    const serial_t&    liid);

static rc_t create_assoc(
    const lvid_t&      lvid,
    const serial_t&    liid,
    const vec_t&        key,
    const vec_t&        el);

static rc_t destroy_assoc(
    const lvid_t&      lvid,
```

```

        const serial_t&      liid,
        const vec_t&         key,
        const vec_t&         el);

static rc_t                  destroy_all_assoc(
    const lvid_t&            lvid,
    const serial_t&          liid,
    const vec_t&             key,
    int&                     num_removed);

static rc_t                  find_assoc(
    const lvid_t&            lvid,
    const serial_t&          liid,
    const vec_t&             key,
    void*                    el,
    ssize_t&                 elen,      // if you don't want the result,
                                      // make this 0 on input
    bool&                    found);

```

DESCRIPTION

The above class **ss_m** methods manipulate B+tree indexes. See **The Shore Storage Manager Programming Interface** for a more detailed description of B+tree indexes.

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 vector pointing to the key portion of an index entry.

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

create_index(lvid, ntype, property, key_desc, size_kb_hint, liid)

The **create_index** methods 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. Valid values for the *ntype* parameter are **t_btree**, indicating a B+tree allowing entries with duplicate keys, and **t_uni_btree**, indicating a B+tree only allowing entries with unique keys. The *property* parameter specifies whether the index is temporary. See **enum(ssm)** for more information on **ss_m::store_property_t**.

The *key_desc* parameter is a string describing the the type of the keys to be stored in the index. The syntax of *key_desc* is as follows:

```

<key_type_str>      ::= <key_type>* <v_key_type>
<key_type>          ::= <type> <len>
<v_key_type>        ::= <type> <var> <len>
<type>              ::= 'i' | 'u' | 'f' | 'b'
<var>               ::= '*' | NULL
<len>               ::= [1-9][0-9]*

```

A <key_type> contains a type ('i' for integer, 'u' for unsigned, 'f' for float, 'b' for binary), and a length. A <v_key_type>, which is the last part of <key_type_str>, can contain an optional

indicator ('*') for variable length field. A <key_type_str> is composed of multiple <key_type>, and a <v_key_type>; i.e. only the last field can be variable length.

For example the key_desc "i4f8b*1000" specifies a key that contains:

1. a 4 byte integer
2. an 8 byte float (double)
3. a variable length binary field that could be as long as as a 1000 bytes.

The SSM applies a function to the key values, the result of which is a string of bytes that can be lexicographically compared, and yield the correct order. The SSM contains such functions for keys of the following types: signed and unsigned integer keys of length 1, 2, or 4, floating point keys of length 4 or 8. Byte strings keys are not interpreted; they are stored as presented to the SSM.

The *size_kb_hint* parameter allows the caller to give a hint about the final size of the index (in K-bytes). This helps the SM determine where to allocate space for the index. For now, the hint's main purpose is to determine whether the index will fit on one page or on many. A value of 0 should be used if it is known the index will be small or if the final size of the index is unknown. Otherwise a value larger than **ss_m::page_sz/1000** should be used.

create_index(lvid, ntype, property, cc, key_desc, size_kb_hint, liid)

This form of **create_index** allows you to associate with the index, a locking protocol other than *t_cc_kvl*. See **enum(ssm)** for more information on **ss_m::concurrency_t**.

See the "ROOT INDEX METHODS" section of **volume(ssm)** for information on how to keep track of the indexes on a volume.

destroy_index(lvid, liid)

The **destroy_index** methods 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_index(lvid, liid, s_lvid, s_lfid, stats)

This **bulkld_index** method bulk loads the **empty** index, identified by *lvid* and *liid*. The entries to load must be 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 and the body contains the element (value) associated with the key. Statistics for the newly loaded index are returned in *stats*, specifically in the *btree* field.

bulkld_index(lvid, liid, sorted_stream, stats)

This **bulkld_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.

print_index(lvid, liid)

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

create_assoc(lvid, liid, key, el)

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

destroy_assoc(lvid, liid, key, el)

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

destroy_all_assoc(lvid, liid, key, num_removed)

The **destroy_all_assoc** method destroys all entries with *key* as a key. The number of entries removed is returned in *num_removed*.

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

The **find_assoc** method finds *key* in the index 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. *Elen* will be set to the length actually written. If *key* is found, then *found* will be set to **true**. A more comprehensive lookup facility, allowing range searches, is available from the class *scan_index_i* described in **scan_index_i(ssm)**

ERRORS

All of the above methods return a **w_rc_t** error code. If an error occurs during a methods that is updating persistent data (the create, destroy, and bulk load methods 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 1.1 of the Shore software.

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.

COPYRIGHT

Copyright © 1994, 1995, 1996, 1997, Computer Sciences Department, University of WisconsinMadison. All Rights Reserved.

SEE ALSO

scan_index_i(ssm), sort_stream_i(ssm) intro(ssm),