**NAME**

scan_index_i – Class for Scanning B+tree Indexes

**SYNOPSIS**

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

class scan_index_i {
public:
    enum cmp_t { bad_cmp_t=badOp, eq=eqOp,
             gt=gtOp, ge=geOp, lt=ltOp, le=leOp };

    /* Logical-ID version */
    NORET               scan_index_i(
      const lvid_t&                 lvid,
      const serial_t&       stid,
      cmp_t                 c1,
      const cvec_t&                 bound1,
      cmp_t                         c2,
      const cvec_t&                 bound2,
      concurrency_t                 cc = t_cc_kvl);

    /* Physical-ID version */
    NORET               scan_index_i(
      const stid_t&                 stid,
      cmp_t                 c1,
      const cvec_t&                 bound1,
      cmp_t                         c2,
      const cvec_t&                 bound2,
      concurrency_t                 cc = t_cc_kvl);

    NORET               ~scan_index_i();

    rc_t                curr(
      vec_t*                        key,
      smsize_t&             klen,
      vec_t*                        el,
      smsize_t&             elen);

    rc_t                next(bool& eof)

    void                finish();
    bool                eof();
    rc_t                error_code();
};
```

**DESCRIPTION**

Class **scan_index_i** supports scanning a range in a B+ tree index. The scan is controlled by a **scan_index_i** object. Multiple scans can be open at one time. More information on indexes and key types is can be found **in the SSM interface document.**

**scan_index_i(lvid, stid, c1, bound1, c2, bound2, cc)**

The **scan_index_i** constructor is used to initialize a scan. The *lvid* and *stid* parameters specify the index to be scanned. The *bound1* and *bound2* parameters specify the keys marking the beginning and end of the scan, respectively. The *c1* and *c2 parameters specify how comparisons* should be made with their corresponding bounds. Valid values are:

```
eq: Only keys equal to the bound will be returned.
    Valid for c1 or c2.
gt: Only keys greater than the bound will be returned.
    Valid only for c1.
ge: Only keys greater than or equal to the bound will
    be returned.  Valid only for c1.
lt: Only keys less than the bound will be returned.
    Valid only for c2.
le: Only keys less than or equal to the bound will
    be returned.  Valid only for c2.
```

The *cc* parameter specifies the granularity of locks acquired for concurrency control. See **enum(ssm)** for a description of the values. Here are the effects of all valid values for file scan:

t_cc_none:
The file is IS locked, but no locks are obtained on any pages or entries in the file.

t_cc_kvl:
The file is IS locked and the keys of the index entries are locked. Next-key locking provides phantom protection.

t_cc_modkvl:
The file is IS locked and the keys of the index entries are locked. No next-key locking is done. The only permissible scans are those where both bounds are **eq.**

t_cc_im:
The value in an entry is treated as a (physical) record identifier, and that record's lock is obtained. Next-record locking provides phantom protection. This locking protocol not useful for VASs that use logical identifiers.

t_cc_file:
The file is SH locked, so no finer-granularity locks are obtained.

**~scan_index_i()**

The destructor frees all resources used by the scan.

**curr(key, klen, el, elen)**

The **curr** method copies out the current key and element. They are copied to the memory addressed by the *key* and *el* vectors. The *klen* parameter will be set to the length of the key copied out. The *elen* parameter will be set to the length of the element copied out.

**next(eof)**

The **next** method advances the scan to the next key/element pair. If the upper bound of the scan has been reached, *eof* will be set to **true.**

**finish()**

The **finish** method frees all resources used by the scan.

**eof()**

If the upper bound of the scan has been reached, the **eof** method will return **true.**

**error_code()**

The **error_code** method will return any error code generated by the other scan member methods. For more information on errors, see ERRORS section below.

**Updates While Scanning**

A common question is *what is the effect of changes to an index made by a transaction that is also scanning the index?* It is not safe to change anything in the file while scanning. Instead, a list of changes should be made during the scan and only performed after the scan is complete.

## ERRORS

A **scan_index_i** object remembers if an error has occured while constructing the scan or while scanning. An error that occurs in constructing the scan (such as having a bad index ID), can be detected by calling **error_code.** Alternatively, the error can be detected on the first call to **next** which will return the remembered error code. Therefore, if an error occurs while constructing or scanning, repeated calls to next will all return the first error code and no progress will be made on the scan.

## 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

**intro(ssm), btree(ssm), scan_file_i(ssm), scan_rt_i(ssm)**