NAME

sort_stream_i − Sorting Utility Class

SYNOPSIS

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

class sort_stream_i {
  public:

    sort_stream_i();
    sort_stream_i(const key_info_t& k, const sort_parm_t& s, uint est_rec_sz=0);
    ~sort_stream_i();

    // initialize the sort_stream
    void      init(const key_info_t& k, const sort_parm_t& s, uint est_rec_sz=0);
    // close the sort stream (release any resource held)
    void      finish();
    // put <key, elem> pair into the sort stream
    rc_t      put(const cvec_t& key, const cvec_t& elem);
    // fetch next pair in sorted order
    rc_t      get_next(vec_t& key, vec_t& elem, bool& eof);
    // detect if the stream is empty
    bool      is_empty();
    // detect if the stream is sorted or not
    bool      is_sorted()
};


struct key_info_t {
    enum key_type_t  { t_char=0, t_int, t_float, t_string, t_spatial };
    enum where_t     { t_hdr=0, t_body };

    key_type_t  type;           // key type
    nbox_t    universe;   // for spatial object only
    bool      derived;    // if true, the key must be the only item in rec
                          // header, and the header will not be copied to
                          // the result record (allow user to store derived
                          // key temporarily for sorting purpose).

    // following applies to file sort only
    where_t   where;      // where the key resides
    uint4     offset;           // offset from the begin
    uint4     len;       // key length

    key_info_t() {
      type = t_int;
      where = t_body;
      offset = 0;
      len = sizeof(int);
      derived = FALSE;
    }
};


//
```

```
// sort parameter
//
struct sort_parm_t {
    uint2    run_size;             // size for each run (# of pages)
    vid_t    vol;          // volume for files
    bool     unique;       // result unique ?
    bool     ascending;            // ascending order ?
    bool     destructive;  // destroy the input file at the end ?
    sm_store_property_t property; // temporary file ?

    sort_parm_t() : run_size(10), unique(false), ascending(true),
                  destructive(false), property(t_regular) {}
};
```

DESCRIPTION

Class **sort_stream_i** class is used for sorting a stream of records. After creating an instance of **sort_stream_i,** you can keep putting <key, element> pairs into the stream and will save all the records to a temporary persistent store, sort them and return them in a sorted order via calls to **get_next.** The temporary store is destroyed automatically upon completion.

To create a **sort_stream_i** instance, you need to supply a **key_info_t** parameter, which includes information about the key type See **btree(ssm)** for a description of key types.

A **sort_parm_t** parameter is needed to provide information on the run size, temporary file volume. Besides, estimated record length will help the sort code to allocate the right amount of resources for the sort.

Note that **sort_stream** exists only during the put and fetch, after the last pair is fetched through **get_next()** the stream is destroyed.

ERRORS

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.

COPYRIGHT

SEE ALSO

btree(ssm), file(ssm)