NAME

ss_m, set_shutdown_flag, setup_options – Class ss_m Methods for Initialization

SYNOPSIS

```
#include <sm_vas.h>

class ss_m;

static void ss_m::set_shutdown_flag(bool clean);
            ss_m::ss_m();
            ss_m(ss_m::LOG_WARN_CALLBACK_FUNC callback = 0);

static rc_t ss_m::setup_options(option_group_t*);
            ss_m::~ss_m();
```

DESCRIPTION

These methods of class *ss_m* control initializing and shutting down the Shore storage manager (SSM). Constructing an instance of *ss_m* starts the SSM. Destroying the instance causes the SSM to shut down.

During construction, the SSM follows the steps discussed in the Recovery section of **The Shore Storage Manager Programming Interface.**

setup_options(option_group_t *)

The **setup_options** method adds storage manager specific options to the *option_group_t.* These options must be initialized before the **ss_m** constructor is called.

ss_m(ss_m::LOG_WARN_CALLBACK_FUNC callback = 0)

The **ss_m** constructor initializes all SSM data structures, and performs recovery based on the current log. Only one instance of **ss_m** may be in existence at any one time (this is enforced by the constructor). The single optional argument is a callback function that is called when the the active transactions have used so much log space that there is a threat of running out of log. The details of this procedure are in the section LOG_WARN_CALLBACK_FUNC, below. If no such callback function is provided, when the threshold is exceeded the SSM returns to the calling function with the error value *RC(E_LOGSPACEWARN).*

Part of SSM initialization includes allocating a buffer pool. The buffer pool is located in shared memory, so the operating system must have shared-memory support to accommodate the size of the buffer pool. If insufficient shared memory is available, the SSM prints a message indicating how much shared memory it is trying to acquire, and exits.

set_shutdown_flag(clean)

The **set_shutdown_flag** method can be used to simulate a crash. If *clean* is set to **false,** the SSM will not flush any buffers when **~ss_m()** is called. If *clean* is set to **true,** all data pages and logs are flushed to disk, and no recovery processing will be needed when the SSM is restarted. This is the normal operation of the storage manager.

˜ss_m()

The **ss_m** destructor flushes all buffers in the buffer pool to disk (unless **set_shutdown_flag(clean)** was used to defeat this) and frees all the resources used by the SSM.

LOG_WARN_CALLBACK_FUNC

At all times there must be enough log space left to abort a transaction. In the SSM, log space is finite (the SSM does not archive parts of log on tertiary storage), so the value-added server must assist in keeping adequate log space available. The run-time option *sm_log_warn* determines a threshold (in percentage of the log) at which the threat of running out of log exists. When all the active transactions in the system together have used that much of the log, the SSM issues a callback to the value-added server, which then chooses a victim (transaction) to abort.

The callback function has the following type, which is in the *ss_m* namespace:

```
typedef w_rc_t (*LOG_WARN_CALLBACK_FUNC) (
    xct_i* iterator,
    xct_t*& victim,
    w_base_t::base_stat_t curr,
    w_base_t::base_stat_t thresh
);
```

The first argument, *iterator*, iterates over the transactions in the system. Its methods **xct_t* xct_i::next()** and **xct_t* xct_i::curr()** return pointers to transaction data structures.

The second argument, *victim*, is where the resulting chosen victim is returned.

The arguments *curr* and *thresh* are simply advisory information: the current number of bytes of log used by all active transactions in the system, and the threshold that was exceeded before the callback was made.

The callback function must analyze data structures that are internal to the storage manager, so the source code for this function must include the definitions of these data structures. To accomplish this, the following macros and inclusions are required:

```
#define SM_LEVEL 1
#define SM_SOURCE
#define XCT_C
#include "sm_int_1.h"
#include "e_error_def_gen.h"

/* Define your callback function here: */
w_rc_t out_of_log_space (
      xct_i* iterator ,
      xct_t *& victim,
      w_base_t::base_stat_t curr,
      w_base_t::base_stat_t thresh
)
{
      /* this function must return one of three states:
       *
       * a valid xct_t* in victim AND the w_rc_t value
       *  RC(E_USERABORT) (in which case, victim is aborted
       *  by the SSM after the calling  SSM method completes)
       *
       * OR
       *
```

```
             * the w_rc_t value RCOK (in which case victim is ignored)
             *
             * OR
             *
             * the any other w_rc_t value (in which case victim is ignored,
             *  but the error code is returned to the caller of the
             *  calling SSM method, and the SSM method is not applied)
             */

            ...

            return RCOK;
    }
```

The detection of threat of log-space overrun occurs whenever any value-added server thread calls a SSM method that might generate any log. The detection and callback occur at the beginning of the method call; subsequent aborting of the victim occurs upon exit from the SSM method.

It is important that the callback function not return the same victim more than once, so the callback function must take precautions to save state in a thread-safe manner.

If no such callback is provided in the SSM constructor, when the threshold is exceeded the SSM returns to the calling function with the error value *RC(E_LOGSPACEWARN)*.

ERRORS

Failure to properly construct/destruct the SSM will result in a **fatal** error that will print a message and exit the program.

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

intro(ssm), volume(ssm), options(common), transaction(ssm)