C++ Software transactional Memory

Zoltan Fuzesi

C00197361 IT Carlow

Supervisor : Joe Kehoe

Wed Mar 7 2018 21:22:57

Contents

1 OSTM C++ Software Transactional Memory

1.1 Object Based Software Transactional Memory.

OSTM is a polymorphic solution to store and manage shared memory spaces within c++ programming context. You can store and managed any kind of object in transactional environment as a shared and protected memory space.

1.1.1 Brief. Download the zip file from the provided link in the web-site, that contains the libostm.so, TM.h, TX.h, OSTM.h files.

Unzip the archive file to the desired destination possibly where in you program is stored.

1.1.2 Step 1: Download the archive file.

1.1.3 Step 2: Unzip in the target destination.

1.1.4 Step 3: Copy the shared library (libostm.so) to the operating system folder where the other shared library are stored.

It will be different destination folder on different platforms. (Linux, Windows, Mac OS) `More Information`

1.1.5 Step 4: Achieve the required class hierarchy between the OSTM library and your own class structure.

Details and instruction of class hierarchy requirements can be found on the web-site. www.serversite.info/ostm

1.1.6 Step 5: Create an executable file as you linking together the TM.h, TX.h, OSTM.h files with your own files.

1.1.7 Step 6: Now your application use transactional environment, that guarantees the consistency between object transactions.

1.1.8 Step 7: Run the application.

2 README

Usage of the STM library on Linux.
In order to use the O_STM library with any C++ application, it need to be placed to the operation system /usr/lib directory.

1. Copy lib_o_stm.so file to /usr/lib: sudo cp lib_o_stm.so /usr/lib
2. Include the TM.h TX.h and the OSTM.h files in your application.
3. Create Makefile :

Makefile.mk Documentation<br>

EXE =Test
CC = g++
PROGRAM = app
CFLAGS =-std=c++14 -pthread
CFILES = main.cpp AIB.cpp ULSTER.cpp BOA.cpp UNBL.cpp SWBPLC.cpp
HFILES = TM.h TX.h OSTM.h AIB.h ULSTER.h BOA.h UNBL.h SWBPLC.h

all:

Rule for SHARED linking

:
∗.cpp -I -L /usr/lib/lib_o_stm.so -o
clean:
rm -f ∗.o

1. Run the application/executable file : ./Test

3 Class Index

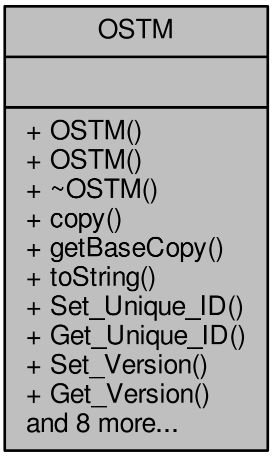
3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

4 Class Documentation

4.1 OSTM Class Reference

Collaboration diagram for OSTM:



Public Member Functions

- OSTM ()

    OSTM Constructor.
- OSTM (int _version_number_, int _unique_id_)

    OSTM Custom Constructor.
- virtual ∼OSTM ()

    De-constructor.
- virtual void copy (std::shared_ptr< OSTM > from, std::shared_ptr< OSTM > to)

    OSTM required virtual method for deep copy.
- virtual std::shared_ptr< OSTM > getBaseCopy (std::shared_ptr< OSTM > object)

    OSTM required virtual method for returning a pointer that is copy of the original pointer.
- virtual void toString ()

    OSTM required virtual method for display object.
- void Set_Unique_ID (int uniqueID)

    setter for unique id
- int Get_Unique_ID () const

    getter for unique id
- void Set_Version (int version)

    setter for version number
- int Get_Version () const

    getter for version number
- void increase_VersionNumber ()

    commit time increase version number to child object
- bool Is_Can_Commit () const

    return boolean

- void Set_Can_Commit (bool canCommit)

    set boolean
- void Set_Abort_Transaction (bool abortTransaction)

    set boolean
- bool Is_Abort_Transaction () const

    return boolean
- void lock_Mutex ()

    object unique lock, locks mutex
- void unlock_Mutex ()

    object unique lock, unlocks mutex
- bool is_Locked ()

    object unique lock, try locks mutex return boolean value depends on the lock state

4.1.1 Detailed Description

Definition at line 17 of file OSTM.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 OSTM::OSTM ( )

OSTM Constructor.

Default constructor.

Parameters

version	indicates the version number of the inherited child pointer
uniqueID	is a unique identifier assigned to every object registered in OSTM library
canCommit	boolean
abort_Transaction	boolean

Definition at line 20 of file OSTM.cpp.

```
{
    this->version = ZERO;
    this->uniqueID = Get_global_Unique_ID_Number(); //++global_Unique_ID_Number;
    this->canCommit = true;
    this->abort_Transaction = false;
}
```

4.1.2.2 OSTM::OSTM (int *_version_number_*, int *_unique_id_*)

OSTM Custom Constructor.

Custom Constructor Used for copy object.

Parameters

version	indicates the version number of the inherited child pointer
uniqueID	is a unique identifier assigned to every object registered in OSTM library
canCommit	boolean
abort_Transaction	boolean

Definition at line 36 of file OSTM.cpp.

```
{
   // std::cout << "OSTM COPY CONSTRUCTOR" << global_Unique_ID_Number << std::endl;
    this->uniqueID = _unique_id_;
    this->version = _version_number_;
    this->canCommit = true;
    this->abort_Transaction = false;
}
```

4.1.2.3 OSTM::∼OSTM ( ) [virtual]

De-constructor.

De-constructor

Definition at line 48 of file OSTM.cpp.

```
            {
    //std::cout << "[OSTM DELETE]" << std::endl;
}
```

4.1.3 Member Function Documentation

4.1.3.1 int OSTM::Get_Unique_ID ( ) const

getter for unique id

Parameters

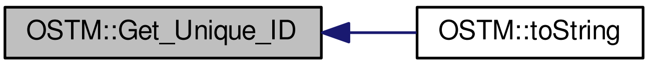
uniqueID	int

Definition at line 73 of file OSTM.cpp.

Referenced by toString().

```
{
    return uniqueID;
}
```

Here is the caller graph for this function:



4.1.3.2 int OSTM::Get_Version ( ) const

getter for version number

Parameters

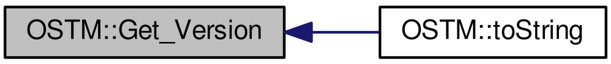
version	int

Definition at line 89 of file OSTM.cpp.

Referenced by toString().

```
{
    return version;
}
```

Here is the caller graph for this function:



4.1.3.3 void OSTM::increase_VersionNumber ( )

commit time increase version number to child object

Parameters

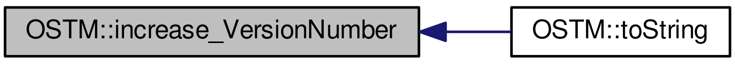
version	int

Definition at line 97 of file OSTM.cpp.

Referenced by toString().

```
{
    this->version += 1;
}
```

Here is the caller graph for this function:



4.1.3.4 bool OSTM::Is_Abort_Transaction ( ) const

return boolean

NOT USED YET.

Parameters

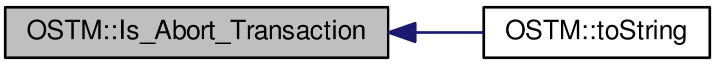
abort_Transaction	boolean

Definition at line 126 of file OSTM.cpp.

Referenced by toString().

```
{
    return abort_Transaction;
}
```

Here is the caller graph for this function:



4.1.3.5 bool OSTM::Is_Can_Commit ( ) const

return boolean

NOT USED YET.

Parameters

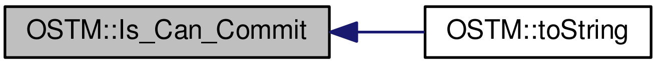
canCommit	boolean

Definition at line 112 of file OSTM.cpp.

Referenced by toString().

```
{
    return canCommit;
}
```

Here is the caller graph for this function:



4.1.3.6 bool OSTM::is_Locked ( )

object unique lock, try locks mutex return boolean value depends on the lock state

Parameters

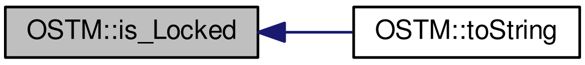
mutex	std::mutex

Definition at line 147 of file OSTM.cpp.

Referenced by toString().

```
00147                    {
00148     return this->mutex.try_lock();
00149 }
```

Here is the caller graph for this function:



4.1.3.7 void OSTM::lock_Mutex ( )

object unique lock, locks mutex

Parameters

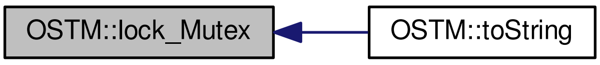
mutex	std::mutex

Definition at line 133 of file OSTM.cpp.

Referenced by toString().

```
00133                   {
00134     this->mutex.lock();
00135 }
```

Here is the caller graph for this function:



4.1.3.8 void OSTM::Set_Abort_Transaction (bool *abortTransaction*)

set boolean

NOT USED YET.

Parameters

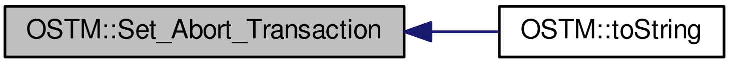
abort_Transaction	boolean

Definition at line 119 of file OSTM.cpp.

Referenced by toString().

```
00119                                                    {
00120     this->abort_Transaction = abortTransaction;
00121 }
```

Here is the caller graph for this function:



4.1.3.9 void OSTM::Set_Can_Commit (bool *canCommit*)

set boolean

NOT USED YET.

Parameters

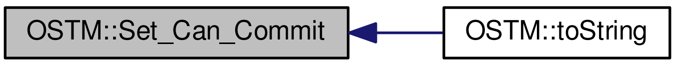
canCommit	boolean

Definition at line 105 of file OSTM.cpp.

Referenced by toString().

```
00105                                            {
00106     this->canCommit = canCommit;
00107 }
```

Here is the caller graph for this function:



4.1.3.10 void OSTM::Set_Unique_ID (int *uniqueID*)

setter for unique id

Parameters

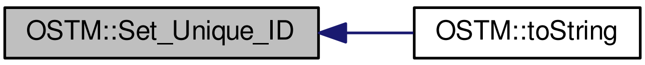
uniqueID	int

Definition at line 66 of file OSTM.cpp.

Referenced by toString().

```
00066                                        {
00067     this->uniqueID = uniqueID;
00068 }
```

Here is the caller graph for this function:



4.1.3.11 void OSTM::Set_Version (int *version*)

setter for version number

Parameters

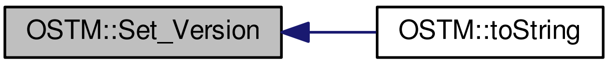
version	int

Definition at line 81 of file OSTM.cpp.

Referenced by toString().

```
00082 {
00083     this->version = version;
00084 }
```

Here is the caller graph for this function:



4.1.3.12 void OSTM::unlock_Mutex ( )

object unique lock, unlocks mutex

Parameters

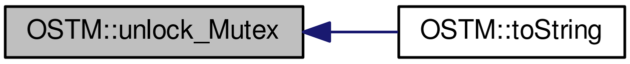
mutex	std::mutex

Definition at line 140 of file OSTM.cpp.

Referenced by toString().

```
00140                         {
00141     this->mutex.unlock();
00142 }
```

Here is the caller graph for this function:

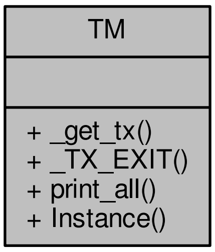


The documentation for this class was generated from the following files:

- OSTM.h
- OSTM.cpp

4.2 TM Class Reference

Collaboration diagram for TM:



Public Member Functions

- std::shared_ptr< TX > const _get_tx ()

  _get_tx std::shared_ptr<TX>, returning a shared pointer with the transaction
- void _TX_EXIT ()

  _TX_EXIT void, the thread calls the ostm_exit function in the transaction, and clear all elements from the shared global collection associated with the main process
- void print_all ()

  ONLY FOR TESTING print_all void, print out all object key from txMAP collection.

Static Public Member Functions

- static TM & Instance ()

  Scott Meyer's Singleton creation, what is thread safe.

4.2.1 Detailed Description

Definition at line 49 of file TM.h.

4.2.2 Member Function Documentation

4.2.2.1 std::shared_ptr< TX > const TM::_get_tx ( )

_get_tx std::shared_ptr<TX>, returning a shared pointer with the transaction

_get_tx std::shared_ptr<TX>, return a shared_ptr with the Transaction object, if TX not exists then create one, else increasing the nesting level std::mutex, protect shared collection from critical section

Parameters

guard	std::lock_guard, locks the register_Lock mutex, unlock automatically when goes out of the scope

Definition at line 77 of file TM.cpp.

```
{
    std::lock_guard<std::mutex> guard(get_Lock);

    std::map<std::thread::id, std::shared_ptr<TX>>::iterator it = txMap.find(std::this_thread::get_id());
    if(it == txMap.end())
    {
       registerTX();
       it = txMap.find(std::this_thread::get_id());

    } else {
        it->second->_increase_tx_nesting();
    }
    //it = txMap.find(std::this_thread::get_id());


    return it->second;

}
```

4.2.2.2 void TM::_TX_EXIT ( )

_TX_EXIT void, the thread calls the ostm_exit function in the transaction, and clear all elements from the shared global collection associated with the main process

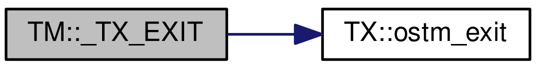
_TX_EXIT void, the thread calls the ostm_exit function in the transaction, and clear all elements from the shared global collection associated with the main process tx TX, local object to function in transaction

Definition at line 100 of file TM.cpp.

References TX::ostm_exit().

```
                  {
    TX tx(std::this_thread::get_id());
    pid_t ppid = getppid();
    std::map<pid_t, std::map< std::thread::id, int >>::iterator process_map_collection_Iterator =
TM::process_map_collection.find(ppid);
    if (process_map_collection_Iterator != TM::process_map_collection.end()) {

        for (auto current = process_map_collection_Iterator->second.begin(); current !=
process_map_collection_Iterator->second.end(); ++current) {
            /*
             * Delete all transaction associated with the actual main process

             */
            txMap.erase(current->first);
        }
        TM::process_map_collection.erase(ppid);

    }
    tx.ostm_exit();
}
```

Here is the call graph for this function:



4.2.2.3 TM & TM::Instance ( ) [static]

Scott Meyer's Singleton creation, what is thread safe.

Instance TM, return the same singleton object to any process.

Parameters

_instance	TM, static class reference to the instance of the Transaction Manager class
_instance	ppid, assigning the process id whoever created the Singleton instance

Definition at line 27 of file TM.cpp.

```
00027                 {
00028     static TM _instance;
00029     _instance._tm_id = getpid();
00030 
00031     return _instance;
00032 }
```

4.2.2.4 void TM::print_all ( )

ONLY FOR TESTING print_all void, print out all object key from txMAP collection.

ONLY FOR TESTING print_all void, prints all object in the txMap

Definition at line 120 of file TM.cpp.

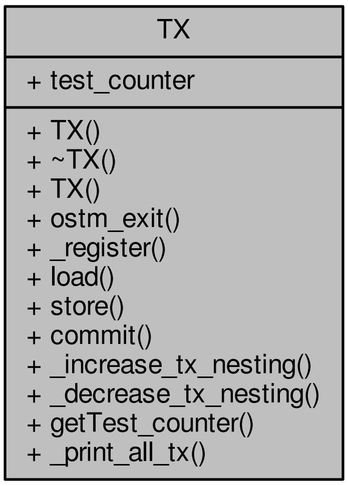
```
00120                  {
00121     get_Lock.lock();
00122     for (auto current = txMap.begin(); current != txMap.end(); ++current) {
00123         std::cout << "KEY : " << current->first << std::endl;
00124     }
00125     get_Lock.unlock();
00126 }
```

The documentation for this class was generated from the following files:

- TM.h
- TM.cpp

4.3 TX Class Reference

Collaboration diagram for TX:



Public Member Functions

- TX (std::thread::id id)

    Constructor.
- ∼TX ()

    De-constructor.
- TX (const TX &orig)

    Default copy constructor.
- void ostm_exit ()

    Delete all map entries associated with the main process.
- void _register (std::shared_ptr< OSTM > object)

    Register OSTM pointer into STM library.
- std::shared_ptr< OSTM > load (std::shared_ptr< OSTM > object)

    load std::shared_ptr<OSTM>, returning an std::shared_ptr<OSTM> copy of the original pointer, to work with during transaction life time
- void store (std::shared_ptr< OSTM > object)

    Store transactional changes.
- bool commit ()

    Commit transactional changes.
- void _increase_tx_nesting ()

    Add TX nesting level by one.
- void _decrease_tx_nesting ()

    Remove TX nesting level by one.
- int getTest_counter ()

    getTest_counter TESTING ONLY!!! returning the value of the test_counter stored, number of rollbacks
- void _print_all_tx ()

Static Public Attributes

- static int test_counter = 0

Friends

- class TM

4.3.1 Detailed Description

Definition at line 24 of file TX.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 TX::TX (std::thread::id *id*)

Constructor.

Parameters

transaction_Number	int, to store associated thread
_tx_nesting_level	int, to store and indicate nesting level of transactions within transaction

Definition at line 31 of file TX.cpp.

```
                      {
    this->transaction_Number = id;
    this->_tx_nesting_level = 0;
}
```

4.3.3 Member Function Documentation

4.3.3.1 void TX::_decrease_tx_nesting ( )

Remove TX nesting level by one.

_decrease_tx_nesting decrease the value stored in _tx_nesting_level by one, when outer transactions commiting

Parameters

_tx_nesting_level	int

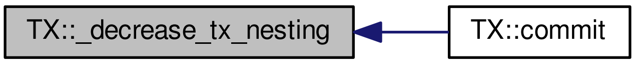
Definition at line 316 of file TX.cpp.

Referenced by commit().

```
                                  {
```

```
   // std::cout << "[this->_tx_nesting_level] = " << this->_tx_nesting_level << std::endl;
    this->_tx_nesting_level -= 1;
;
}
```

Here is the caller graph for this function:



4.3.3.2 void TX::_increase_tx_nesting ( )

Add TX nesting level by one.

_increase_tx_nesting increase the value stored in _tx_nesting_level by one, indicate that the transaction nested

Parameters

_tx_nesting_level	int

Definition at line 307 of file TX.cpp.

```
                               {

    this->_tx_nesting_level += 1;
    // std::cout << "[this->_tx_nesting_level] = " << this->_tx_nesting_level << std::endl;
}
```

4.3.3.3 void TX::_print_all_tx ( )

ONLY FOR TESTING CHECK THE MAP AFTER THREAD EXIT AND ALL SHOULD BE DELETED!!!!!!!

Definition at line 346 of file TX.cpp.

```
                        {

    std::cout << "[PRINTALLTHREAD]" << std::endl;
    std::map< int, std::shared_ptr<OSTM> >::iterator it;
    /*
     * All registered thread id in the TX global

     */
     pid_t ppid = getppid();
    std::map<pid_t, std::map< int, int >>::iterator process_map_collection_Iterator =
TX::process_map_collection.find(ppid);
    if (process_map_collection_Iterator != TX::process_map_collection.end()) {

        for (auto current = process_map_collection_Iterator->second.begin(); current !=
process_map_collection_Iterator->second.end(); ++current) {
            it = working_Map_collection.find(current->first);
            if(it != working_Map_collection.end()){
                std::cout << "[Unique number ] : " <<it->second->Get_Unique_ID() << std::endl;
            }


        }

    }
}
```

4.3.3.4 void TX::_register (std::shared_ptr< OSTM > *object*)

Register OSTM pointer into STM library.

register void, receives an std::shared_ptr<OSTM> that point to the original memory space to protect from reca conditions

Parameters

working_Map_collection	std::map, store all the std::shared_ptr<OSTM> pointer in the transaction
main_Process_Map_collection	std::map, store all std::shared_ptr<OSTM> from all transaction, used to lock and compare the objects
process_map_collection	std::map, store all std::shared_ptr<OSTM> unique ID from all transaction, used to delete all pointers used by the main process, from all transaction before the program exit.
std::lock_guard	use register_Lock(mutex) shared lock between all transaction
ppid	int, store main process number

Definition at line 104 of file TX.cpp.

```
                                              {
    /*
     * MUST USE SHARED LOCK TO PROTECT SHARED GLOBAL MAP/COLLECTION

     */
    std::lock_guard<std::mutex> guard(TX::register_Lock);

    /*
     * Check for null pointer !

     * Null pointer can cause segmentation fault!!!

     */
    if(object == nullptr){
        throw std::runtime_error(std::string("[RUNTIME ERROR : NULL POINTER IN REGISTER FUNCTION]") );
    }

    pid_t ppid = getppid();
    std::map<pid_t, std::map< int, int >>::iterator process_map_collection_Iterator =
TX::process_map_collection.find(ppid);
    if (process_map_collection_Iterator == TX::process_map_collection.end()) {
        /*
         * Register main process/application to the global map

         */
        std::map< int, int >map =  get_thread_Map();
        TX::process_map_collection.insert({ppid, map});
        /*
         * Get the map if registered first time

         */
        process_map_collection_Iterator = TX::process_map_collection.find(ppid);
    }
    std::map<int, std::shared_ptr<OSTM>>::iterator main_Process_Map_collection_Iterator =
TX::main_Process_Map_collection.find(object->Get_Unique_ID());
    if (main_Process_Map_collection_Iterator == TX::main_Process_Map_collection.end()) {
        /*
         * Insert to the GLOBAL MAP

         */
        TX::main_Process_Map_collection.insert({object->Get_Unique_ID(), object});
        /*
         * Insert to the GLOBAL MAP as a helper to clean up at end of main process

         */
        process_map_collection_Iterator->second.insert({object->Get_Unique_ID(), 1});
    }


    std::map< int, std::shared_ptr<OSTM> >::iterator working_Map_collection_Object_Shared_Pointer_Iterator
= working_Map_collection.find(object->Get_Unique_ID());
    if (working_Map_collection_Object_Shared_Pointer_Iterator == working_Map_collection.end()) {

```

```
00147         working_Map_collection.insert({object->Get_Unique_ID(), object->getBaseCopy(object)});
00148     }
00149
00150 }
```

4.3.3.5 bool TX::commit ( )

Commit transactional changes.

commit bool, returns boolean value TRUE/FALSE depends on the action taken within the function

Parameters

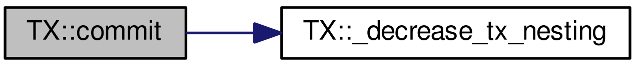
working_Map_collection	std::map, store all the std::shared_ptr<OSTM> pointer in the transaction
main_Process_Map_collection	std::map, store all std::shared_ptr<OSTM> from all transaction, used to lock and compare the objects
can_Commit	bool, helps to make decision that the transaction can commit or rollback

Definition at line 202 of file TX.cpp.

References _decrease_tx_nesting(), and test_counter.

```
00202                  {
00203
00204     bool can_Commit = true;
00205
00206     /*
00207      * Dealing with nested transactions first

00208      */
00209     if (this->_tx_nesting_level > 0) {
00210         _decrease_tx_nesting();
00211         return true;
00212     }
00213
00214     std::map< int, std::shared_ptr<OSTM> >::iterator working_Map_collection_Object_Shared_Pointer_Iterator;
00215
00216     std::map<int, std::shared_ptr<OSTM>>::iterator main_Process_Map_collection_Iterator;
00217     for (working_Map_collection_Object_Shared_Pointer_Iterator = working_Map_collection.begin();
      working_Map_collection_Object_Shared_Pointer_Iterator != working_Map_collection.end();
      working_Map_collection_Object_Shared_Pointer_Iterator++) {
00218
00219             main_Process_Map_collection_Iterator = TX::main_Process_Map_collection.find(
      working_Map_collection_Object_Shared_Pointer_Iterator->second->Get_Unique_ID());
00220             /*
00221              * Throws runtime error if object can not find

00222              */
00223             if(main_Process_Map_collection_Iterator == TX::main_Process_Map_collection.end())
00224             {
00225                 throw std::runtime_error(std::string("[RUNTIME ERROR : CAN'T FIND OBJECT COMMIT FUNCTION]")
      );
00226             }
00227
00228         /*
00229          * Busy wait WHILE object locked by other thread

00230          */
00231         while(!(main_Process_Map_collection_Iterator->second)->is_Locked());
00232
00233         if (main_Process_Map_collection_Iterator->second->Get_Version() >
      working_Map_collection_Object_Shared_Pointer_Iterator->second->Get_Version()) {
00234
00235             working_Map_collection_Object_Shared_Pointer_Iterator->second->Set_Can_Commit(false);
00236             can_Commit = false;
00237             break;
00238         } else {
00239
00240             working_Map_collection_Object_Shared_Pointer_Iterator->second->Set_Can_Commit(true);
00241         }
00242     }
00243     if (!can_Commit) {
        TX::test_counter += 1;
        for (working_Map_collection_Object_Shared_Pointer_Iterator = working_Map_collection.begin();
working_Map_collection_Object_Shared_Pointer_Iterator != working_Map_collection.end();
working_Map_collection_Object_Shared_Pointer_Iterator++) {

            main_Process_Map_collection_Iterator  = TX::main_Process_Map_collection.find(
working_Map_collection_Object_Shared_Pointer_Iterator->second->Get_Unique_ID());
            (working_Map_collection_Object_Shared_Pointer_Iterator->second)->copy(
working_Map_collection_Object_Shared_Pointer_Iterator->second, main_Process_Map_collection_Iterator->second);

        }

        _release_object_lock();

        return false;
    } else {
        /*
         * Commit changes

         */
        for (working_Map_collection_Object_Shared_Pointer_Iterator = working_Map_collection.begin();
working_Map_collection_Object_Shared_Pointer_Iterator != working_Map_collection.end();
working_Map_collection_Object_Shared_Pointer_Iterator++) {

                main_Process_Map_collection_Iterator = TX::main_Process_Map_collection.find((
working_Map_collection_Object_Shared_Pointer_Iterator->second)->Get_Unique_ID());
                if (main_Process_Map_collection_Iterator != TX::main_Process_Map_collection.end()) {

                    (main_Process_Map_collection_Iterator->second)->copy(
main_Process_Map_collection_Iterator->second, working_Map_collection_Object_Shared_Pointer_Iterator->second);
                    main_Process_Map_collection_Iterator->second->increase_VersionNumber();


                } else {
                    throw std::runtime_error(std::string("[RUNTIME ERROR : CAN'T FIND OBJECT COMMIT
 FUNCTION]"));

                }
        }


        _release_object_lock();
        this->th_exit();
        return true;
    }
}//Commit finish
```

Here is the call graph for this function:



4.3.3.6 std::shared_ptr< OSTM > TX::load (std::shared_ptr< OSTM > *object*)

load std::shared_ptr<OSTM>, returning an std::shared_ptr<OSTM> copy of the original pointer, to work with during transaction life time

Register OSTM pointer into STM library

Parameters

working_Map_collection	std::map, store all the std::shared_ptr<OSTM> pointer in the transaction

Definition at line 155 of file TX.cpp.

```
00155                                                         {
00156 
00157     std::map< int, std::shared_ptr<OSTM> >::iterator working_Map_collection_Object_Shared_Pointer_Iterator;
00158     /*
00159      * Check for null pointer !

00160      * Null pointer can cause segmentation fault!!!

00161      */
00162     if(object == nullptr){
00163         throw std::runtime_error(std::string("[RUNTIME ERROR : NULL POINTER IN LOAD FUNCTION]") );
00164     }
00165 
00166         working_Map_collection_Object_Shared_Pointer_Iterator = working_Map_collection.find(object->
      Get_Unique_ID());
00167 
00168     if (working_Map_collection_Object_Shared_Pointer_Iterator != working_Map_collection.end()) {
00169 
00170         return working_Map_collection_Object_Shared_Pointer_Iterator->second->getBaseCopy(
      working_Map_collection_Object_Shared_Pointer_Iterator->second);
00171 
00172     } else { throw std::runtime_error(std::string("[RUNTIME ERROR : NO OBJECT FOUND LOAD FUNCTION]") );}
00173 }
```

4.3.3.7 void TX::ostm_exit ( )

Delete all map entries associated with the main process.

ostm_exit void, clear all elements from the shared global collections associated with the main process

Parameters

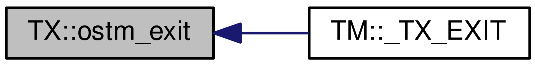
main_Process_Map_collection	std::map, store all std::shared_ptr<OSTM> from all transaction shared between multiple processes
process_map_collection	std::map, store all unique id from all transaction within main process DO NOT CALL THIS METHOD EXPLICITLY!!!!!! WILL DELETE ALL PROCESS ASSOCIATED ELEMENTS!!!!

Definition at line 72 of file TX.cpp.

Referenced by TM::_TX_EXIT().

```
00072                     {
00073     std::map<int, std::shared_ptr<OSTM>>::iterator main_Process_Map_collection_Iterator;
00074 
00075     pid_t ppid = getppid();
00076     std::map<pid_t, std::map< int, int >>::iterator process_map_collection_Iterator =
      TX::process_map_collection.find(ppid);
00077     if (process_map_collection_Iterator != TX::process_map_collection.end()) {
00078 
00079         for (auto current = process_map_collection_Iterator->second.begin(); current !=
      process_map_collection_Iterator->second.end(); ++current) {
00080             main_Process_Map_collection_Iterator = TX::main_Process_Map_collection.find(current->first);
00081 
00082             if (main_Process_Map_collection_Iterator != TX::main_Process_Map_collection.end()){
00083                 /*
00084                  * Delete element from shared main_Process_Map_collection by object unique key value,
       shared_ptr will destroy automatically

00085                  */
00086                 TX::main_Process_Map_collection.erase(main_Process_Map_collection_Iterator->first);
00087             }
00088         }
00089         /*
00090          * Delete from Process_map_collection, Main process exits delete association with library

00091          */
00092         TX::process_map_collection.erase(process_map_collection_Iterator->first);
00093     }
00094 }
```

Here is the caller graph for this function:



4.3.3.8 void TX::store (std::shared_ptr< OSTM > *object*)

Store transactional changes.

store void, receive an std::shared_ptr<OSTM> object to store the changes within the transaction, depends the user action

Parameters

working_Map_collection	std::map, store all the std::shared_ptr<OSTM> pointer in the transaction

Definition at line 178 of file TX.cpp.

```
                                            {
    /*
     * Check for null pointer !

     * Null pointer can cause segmentation fault!!!

     */
    if(object == nullptr){
        throw std::runtime_error(std::string("[RUNTIME ERROR : NULL POINTER IN STORE FUNCTION]") );
    }

    std::map< int, std::shared_ptr<OSTM> >::iterator working_Map_collection_Object_Shared_Pointer_Iterator;

    working_Map_collection_Object_Shared_Pointer_Iterator = working_Map_collection.find(object->
Get_Unique_ID());
    if (working_Map_collection_Object_Shared_Pointer_Iterator != working_Map_collection.end()) {

        working_Map_collection_Object_Shared_Pointer_Iterator->second = object;

    } else { std::cout << "[ERROR STORE]" << std::endl; }
}
```

4.3.4 Friends And Related Function Documentation

4.3.4.1 friend class TM [friend]

Only TM Transaction Manager can create instance of TX Transaction

Definition at line 70 of file TX.h.

4.3.5 Member Data Documentation

4.3.5.1 int TX::test_counter = 0 [static]

Parameters

test_counter	int ONLY FOR TESTING!!!
static	Global counter for rollback

Definition at line 78 of file TX.h.

Referenced by commit(), and getTest_counter().

The documentation for this class was generated from the following files:

- TX.h
- TX.cpp