CppUnit test STM

Zoltan Fuzesi

C00197361 IT Carlow

STM Library reliability test

Contents

1 Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

2 Class Index

2.1 Class List

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

3 File Index

3.1 File List

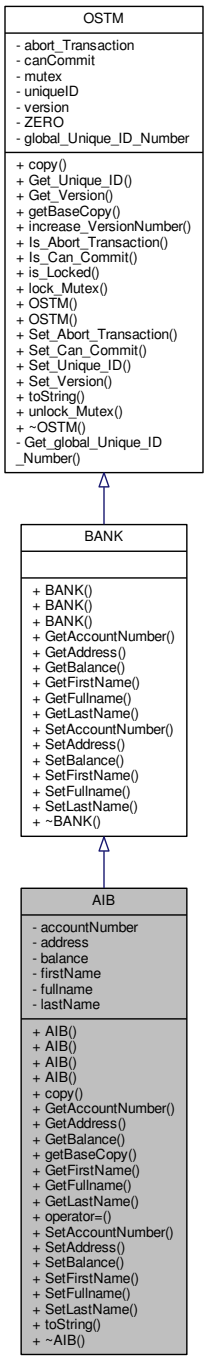
Here is a list of all files with brief descriptions:

4 Class Documentation

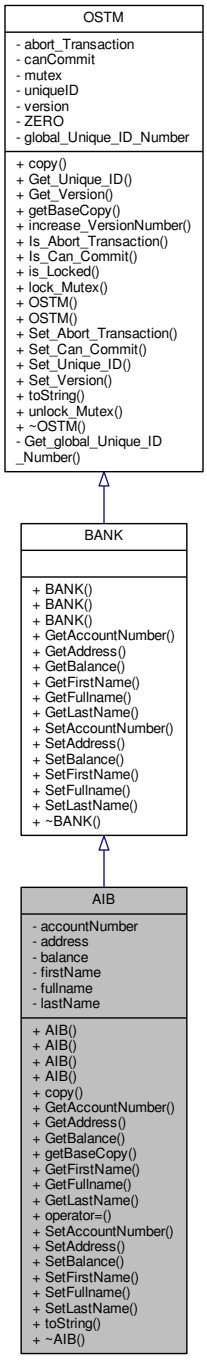
4.1 AIB Class Reference

```
#include <AIB.h>
```

Inheritance diagram for AIB:



Collaboration diagram for AIB:



Public Member Functions

- AIB ()
- AIB (int accountNumber, double balance, std::string firstName, std::string lastName, std::string address)
- AIB (std::shared_ptr< BANK > obj, int _version, int _unique_id)
- AIB (const AIB &orig)
- virtual void copy (std::shared_ptr< OSTM > to, std::shared_ptr< OSTM > from)

OSTM required virtual method for deep copy.

- virtual int GetAccountNumber () const
- virtual std::string GetAddress () const
- virtual double GetBalance () const
- 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 std::string GetFirstName () const
- virtual std::string GetFullname () const
- virtual std::string GetLastName () const
- AIB operator= (const AIB &orig)
- virtual void SetAccountNumber (int accountNumber)
- virtual void SetAddress (std::string address)
- virtual void SetBalance (double balance)
- virtual void SetFirstName (std::string firstName)
- virtual void SetFullname (std::string fullname)
- virtual void SetLastName (std::string lastName)
- virtual void toString ()

  OSTM required virtual method for display object.
- virtual ~AIB ()

Private Attributes

- int accountNumber
- std::string address
- double balance
- std::string firstName
- std::string fullname
- std::string lastName

4.1.1 Detailed Description

Definition at line 18 of file AIB.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 AIB::AIB ( ) [inline]

Definition at line 23 of file AIB.h.

References accountNumber, address, balance, firstName, fullname, and lastName.

Referenced by AIB(), and getBaseCopy().

```
00023          : BANK()
00024     {
00025         this->accountNumber = 0;
00026         this->balance = 50;
00027         this->firstName = "Joe";
00028         this->lastName = "Blog";
00029         this->address = "High street, Carlow";
00030         this->fullname = firstName + " " + lastName;
00031
00032     };
```

4.1.2.2 AIB::AIB (int *accountNumber,* double *balance,* std::string *firstName,* std::string *lastName,* std::string *address*) [inline]

Definition at line 36 of file AIB.h.

References accountNumber, address, balance, firstName, fullname, and lastName.

```
00036                                                                                                                            :
      BANK()
00037     {
00038         this->accountNumber = accountNumber;
00039         this->balance = balance;
00040         this->firstName = firstName;
00041         this->lastName = lastName;
00042         this->address = address;
00043         this->fullname = firstName + " " + lastName;
00044     };
```

4.1.2.3 AIB::AIB (std::shared_ptr< BANK > *obj,* int *_version,* int *_unique_id*) [inline]

Definition at line 48 of file AIB.h.

References accountNumber, address, AIB(), balance, firstName, fullname, and lastName.

```
00048                                                                                  : BANK(_version, _unique_id)
00049     {
00050 
00051         this->accountNumber = obj->GetAccountNumber();
00052         this->balance = obj->GetBalance();
00053         this->firstName = obj->GetFirstName();
00054         this->lastName = obj->GetLastName();
00055         this->address = obj->GetAddress();
00056         this->fullname = obj->GetFirstName() + " " + obj->GetLastName();
00057 
00058     };
```

Here is the call graph for this function:

4.1.2.4 AIB::AIB (const AIB & *orig*)

Definition at line 14 of file AIB.cpp.

```
00014                          {
00015 }
```

4.1.2.5 AIB::~AIB ( ) [virtual]

Definition at line 17 of file AIB.cpp.

Referenced by operator=().

```
00017           {
00018 }
```

4.1.3 Member Function Documentation

4.1.3.1 void AIB::copy (std::shared_ptr< OSTM > *from*, std::shared_ptr< OSTM > *to*) [virtual]

OSTM required virtual method for deep copy.

Reimplemented from OSTM.

Definition at line 37 of file AIB.cpp.

References OSTM::Set_Unique_ID().

Referenced by operator=().

```
00037                                                                   {
00038
00039     std::shared_ptr<AIB> objTO = std::dynamic_pointer_cast<AIB>(to);
00040     std::shared_ptr<AIB> objFROM = std::dynamic_pointer_cast<AIB>(from);
00041     objTO->Set_Unique_ID(objFROM->Get_Unique_ID());
00042     objTO->Set_Version(objFROM->Get_Version());
00043     objTO->SetAccountNumber(objFROM->GetAccountNumber());
00044     objTO->SetBalance(objFROM->GetBalance());
00045 }
```

Here is the call graph for this function:

4.1.3.2 int AIB::GetAccountNumber ( ) const [virtual]

Reimplemented from BANK.

Definition at line 75 of file AIB.cpp.

References accountNumber.

Referenced by operator=(), and toString().

```
00075                                   {
00076     return accountNumber;
00077 }
```

4.1.3.3 std::string AIB::GetAddress ( ) const [virtual]

Reimplemented from BANK.

Definition at line 59 of file AIB.cpp.

References address.

Referenced by operator=().

```
00059                                   {
00060     return address;
00061 }
```

4.1.3.4 double AIB::GetBalance ( ) const [virtual]

Reimplemented from BANK.

Definition at line 67 of file AIB.cpp.

References balance.

Referenced by operator=(), and toString().

```
00067                                 {
00068     return balance;
00069 }
```

4.1.3.5 std::shared_ptr< OSTM > AIB::getBaseCopy (std::shared_ptr< OSTM > *object*) [virtual]

OSTM required virtual method for returning a pointer that is copy of the original pointer.

Reimplemented from OSTM.

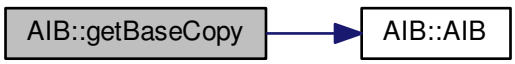
Definition at line 24 of file AIB.cpp.

References AIB().

Referenced by operator=().

```
00025 {
00026
00027     std::shared_ptr<BANK> objTO = std::dynamic_pointer_cast<BANK>(object);
00028     std::shared_ptr<BANK> obj(new AIB(objTO, object->Get_Version(),object->Get_Unique_ID()));
00029     std::shared_ptr<OSTM> ostm_obj = std::dynamic_pointer_cast<OSTM>(obj);
00030     return ostm_obj;
00031 }
```

Here is the call graph for this function:



4.1.3.6 std::string AIB::GetFirstName ( ) const [virtual]

Reimplemented from BANK.

Definition at line 91 of file AIB.cpp.

References firstName.

Referenced by operator=(), and toString().

```
00091                                   {
00092     return firstName;
00093 }
```

4.1.3.7 std::string AIB::GetFullname ( ) const [virtual]

Reimplemented from BANK.

Definition at line 99 of file AIB.cpp.

References fullname.

Referenced by operator=().

```
00099                                   {
00100     return fullname;
00101 }
```

4.1.3.8 std::string AIB::GetLastName ( ) const [virtual]

Reimplemented from BANK.

Definition at line 83 of file AIB.cpp.

References lastName.

Referenced by operator=(), and toString().

```
00083                                   {
00084     return lastName;
00085 }
```

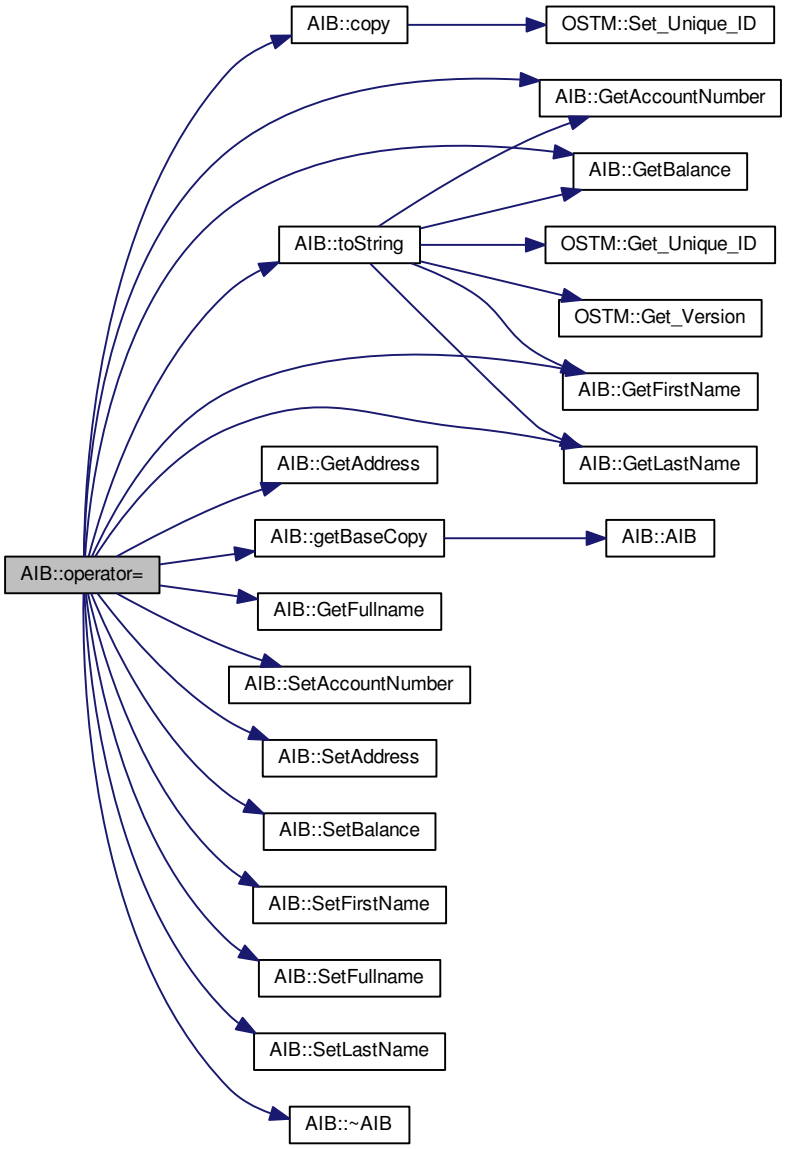
4.1.3.9 AIB AIB::operator= (const AIB & *orig*) [inline]

Definition at line 66 of file AIB.h.

References accountNumber, address, balance, copy(), firstName, fullname, GetAccountNumber(), GetAddress(), GetBalance(), getBaseCopy(), GetFirstName(), GetFullname(), GetLastName(), lastName, SetAccountNumber(), SetAddress(), SetBalance(), SetFirstName(), SetFullname(), SetLastName(), toString(), and ~AIB().

```
00066 {};
```

Here is the call graph for this function:



4.1.3.10 void AIB::SetAccountNumber (int *accountNumber*) [virtual]

Reimplemented from BANK.

Definition at line 71 of file AIB.cpp.

References accountNumber.

Referenced by operator=().

```
00071                                          {
00072     this->accountNumber = accountNumber;
00073 }
```

4.1.3.11 void AIB::SetAddress (std::string *address*) [virtual]

Reimplemented from BANK.

Definition at line 55 of file AIB.cpp.

References address.

Referenced by operator=().

```
00055                                      {
00056     this->address = address;
00057 }
```

4.1.3.12 void AIB::SetBalance (double *balance*) [virtual]

Reimplemented from BANK.

Definition at line 63 of file AIB.cpp.

References balance.

Referenced by operator=().

```
00063                                    {
00064     this->balance = balance;
00065 }
```

4.1.3.13 void AIB::SetFirstName (std::string *firstName*) [virtual]

Reimplemented from BANK.

Definition at line 87 of file AIB.cpp.

References firstName.

Referenced by operator=().

```
00087                                          {
00088     this->firstName = firstName;
00089 }
```

4.1.3.14 void AIB::SetFullname (std::string *fullname*) [virtual]

Reimplemented from BANK.

Definition at line 95 of file AIB.cpp.

References fullname.

Referenced by operator=().

```
00095                                       {
00096     this->fullname = fullname;
00097 }
```

4.1.3.15 void AIB::SetLastName (std::string *lastName*) [virtual]

Reimplemented from BANK.

Definition at line 79 of file AIB.cpp.

References lastName.

Referenced by operator=().

```
00079                                       {
00080     this->lastName = lastName;
00081 }
```

4.1.3.16 void AIB::toString ( ) [virtual]

OSTM required virtual method for display object.

Reimplemented from OSTM.

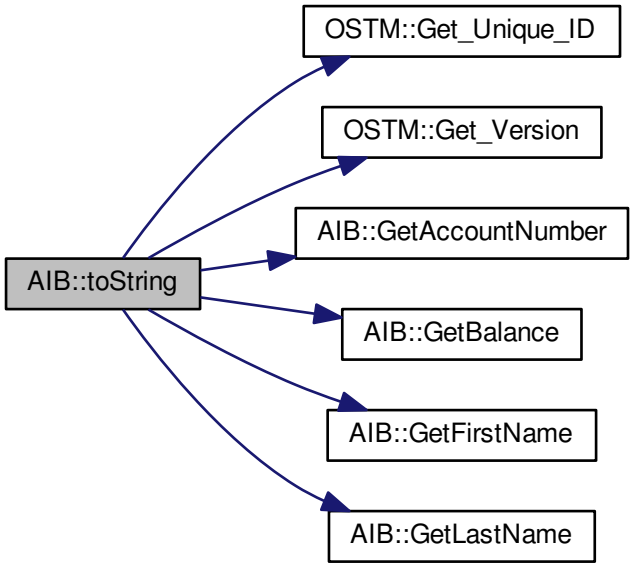
Definition at line 50 of file AIB.cpp.

References OSTM::Get_Unique_ID(), OSTM::Get_Version(), GetAccountNumber(), GetBalance(), GetFirstName(), and GetLastName().

Referenced by operator=().

```
00051 {
00052     std::cout << "\nAIB BANK" << "\nUnique ID : " << this->Get_Unique_ID() << "\nInt account :
       " << this->GetAccountNumber() << "\nDouble value : " << this->
      GetBalance() << "\nFirst name: " << this->GetFirstName() << "\nLast name : " <<
      this->GetLastName()  << "\nVersion number : " << this->Get_Version() << std::endl;
00053 }
```

Here is the call graph for this function:



4.1.4 Member Data Documentation

4.1.4.1 int AIB::accountNumber [private]

Definition at line 99 of file AIB.h.

Referenced by AIB(), GetAccountNumber(), operator=(), and SetAccountNumber().

4.1.4.2 std::string AIB::address [private]

Definition at line 101 of file AIB.h.

Referenced by AIB(), GetAddress(), operator=(), and SetAddress().

4.1.4.3 double AIB::balance [private]

Definition at line 100 of file AIB.h.

Referenced by AIB(), GetBalance(), operator=(), and SetBalance().

4.1.4.4 std::string AIB::firstName [private]

Definition at line 97 of file AIB.h.

Referenced by AIB(), GetFirstName(), operator=(), and SetFirstName().

4.1.4.5 std::string AIB::fullname [private]

Definition at line 96 of file AIB.h.

Referenced by AIB(), GetFullname(), operator=(), and SetFullname().

4.1.4.6 std::string AIB::lastName [private]

Definition at line 98 of file AIB.h.

Referenced by AIB(), GetLastName(), operator=(), and SetLastName().

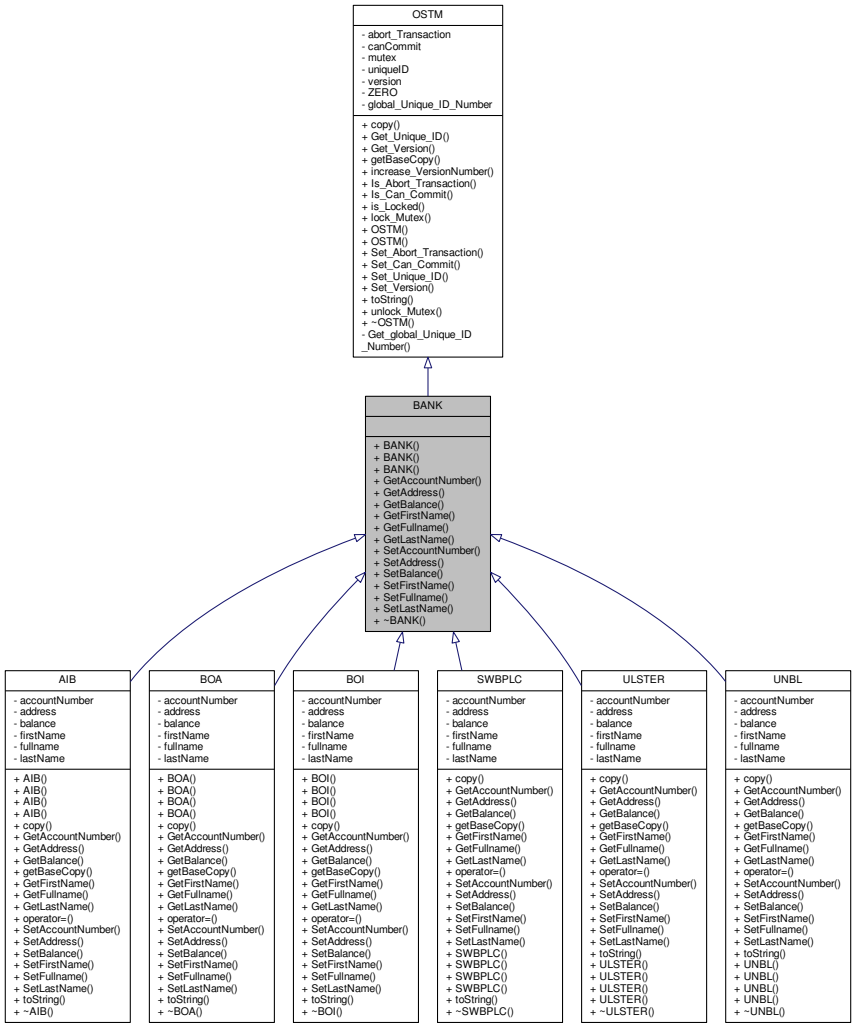
The documentation for this class was generated from the following files:

- AIB.h
- AIB.cpp

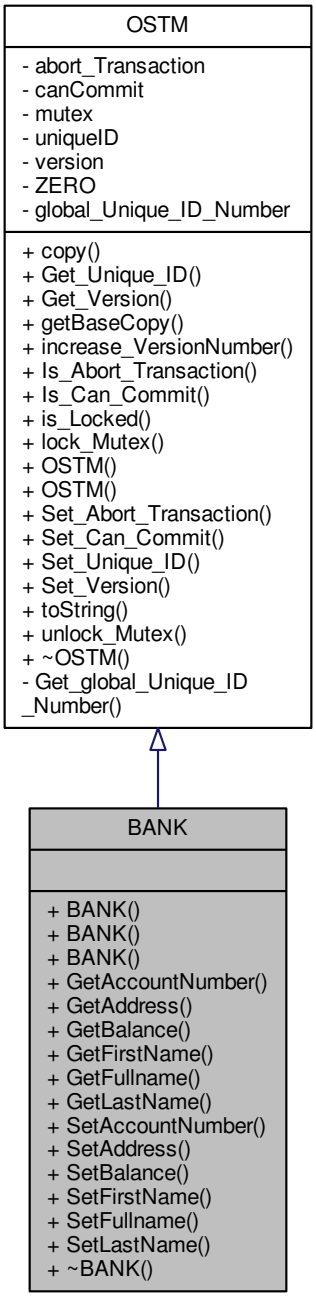
4.2 BANK Class Reference

`#include <BANK.h>`

Inheritance diagram for BANK:



Collaboration diagram for BANK:



Public Member Functions

- BANK ()
- BANK (int _version, int _unique_id)
- BANK (const BANK &orig)
- virtual int GetAccountNumber () const
- virtual std::string GetAddress () const

- virtual double GetBalance () const
- virtual std::string GetFirstName () const
- virtual std::string GetFullname () const
- virtual std::string GetLastName () const
- virtual void SetAccountNumber (int accountNumber)
- virtual void SetAddress (std::string address)
- virtual void SetBalance (double balance)
- virtual void SetFirstName (std::string firstName)
- virtual void SetFullname (std::string fullname)
- virtual void SetLastName (std::string lastName)
- virtual ~BANK ()

4.2.1 Detailed Description

Definition at line 16 of file BANK.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 BANK::BANK ( ) [inline]

Definition at line 23 of file BANK.h.

Referenced by BANK().

```
00023            : OSTM(){
00024
00025     };
```

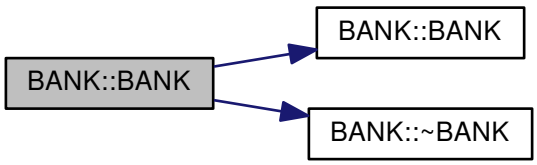
4.2.2.2 BANK::BANK (int *_version*, int *_unique_id*) [inline]

Definition at line 29 of file BANK.h.

References BANK(), and ~BANK().

```
00029                                        : OSTM(_version, _unique_id){
00030
00031     };
```

Here is the call graph for this function:



4.2.2.3 BANK::BANK (const BANK & *orig*)

Definition at line 11 of file BANK.cpp.

```
00011                           {
00012 }
```

4.2.2.4 BANK::~BANK ( ) [virtual]

Definition at line 14 of file BANK.cpp.

Referenced by BANK().

```
00014             {
00015 }
```

4.2.3 Member Function Documentation

4.2.3.1 virtual int BANK::GetAccountNumber ( ) const [inline], [virtual]

Reimplemented in AIB, BOA, BOI, SWBPLC, ULSTER, and UNBL.

Definition at line 49 of file BANK.h.

```
00049 {};
```

4.2.3.2 virtual std::string BANK::GetAddress ( ) const [inline], [virtual]

Reimplemented in AIB, BOA, BOI, SWBPLC, ULSTER, and UNBL.

Definition at line 45 of file BANK.h.

```
00045 {};
```

4.2.3.3 virtual double BANK::GetBalance ( ) const [inline], [virtual]

Reimplemented in AIB, BOA, BOI, SWBPLC, ULSTER, and UNBL.

Definition at line 47 of file BANK.h.

```
00047 {};
```

4.2.3.4 virtual std::string BANK::GetFirstName ( ) const [inline], [virtual]

Reimplemented in AIB, BOA, BOI, SWBPLC, ULSTER, and UNBL.

Definition at line 53 of file BANK.h.

```
00053 {};
```

4.2.3.5 virtual std::string BANK::GetFullname ( ) const [inline], [virtual]

Reimplemented in AIB, BOA, BOI, SWBPLC, ULSTER, and UNBL.

Definition at line 55 of file BANK.h.

```
00055 {};
```

4.2.3.6 virtual std::string BANK::GetLastName ( ) const [inline], [virtual]

Reimplemented in AIB, BOA, BOI, SWBPLC, ULSTER, and UNBL.

Definition at line 51 of file BANK.h.

```
00051 {};
```

4.2.3.7 virtual void BANK::SetAccountNumber (int *accountNumber*) [inline], [virtual]

Reimplemented in AIB, BOA, BOI, SWBPLC, ULSTER, and UNBL.

Definition at line 48 of file BANK.h.

```
00048 {};
```

4.2.3.8 virtual void BANK::SetAddress (std::string *address*) [inline], [virtual]

Reimplemented in AIB, BOA, BOI, SWBPLC, ULSTER, and UNBL.

Definition at line 44 of file BANK.h.

```
00044 {};
```

4.2.3.9 virtual void BANK::SetBalance (double *balance*) [inline], [virtual]

Reimplemented in AIB, BOA, BOI, SWBPLC, ULSTER, and UNBL.

Definition at line 46 of file BANK.h.

Referenced by MyTestCAse::_collection_bject_(), MyTestCAse::_complex_transfer_(), client::_complex_transfer↩_(), client::_nesting_(), MyTestCAse::_nesting_(), MyTestCAse::_one_account_transfer_(), MyTestCAse::_six_↩account_transfer_(), client::_six_account_transfer_(), client::_two_account_transfer_(), and MyTestCAse::_two_↩account_transfer_().

```
00046 {};
```

4.2.3.10 virtual void BANK::SetFirstName (std::string *firstName*) [inline], [virtual]

Reimplemented in AIB, BOA, BOI, SWBPLC, ULSTER, and UNBL.

Definition at line 52 of file BANK.h.

```
00052 {};
```

4.2.3.11 virtual void BANK::SetFullname (std::string *fullname*) [inline], [virtual]

Reimplemented in AIB, BOA, BOI, SWBPLC, ULSTER, and UNBL.

Definition at line 54 of file BANK.h.

```
00054 {};
```

4.2.3.12 virtual void BANK::SetLastName (std::string *lastName*) [inline], [virtual]

Reimplemented in AIB, BOA, BOI, SWBPLC, ULSTER, and UNBL.

Definition at line 50 of file BANK.h.

```
00050 {};
```

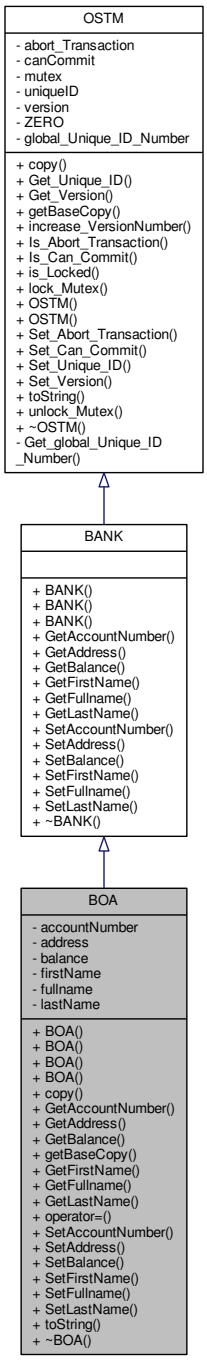
The documentation for this class was generated from the following files:

- BANK.h
- BANK.cpp

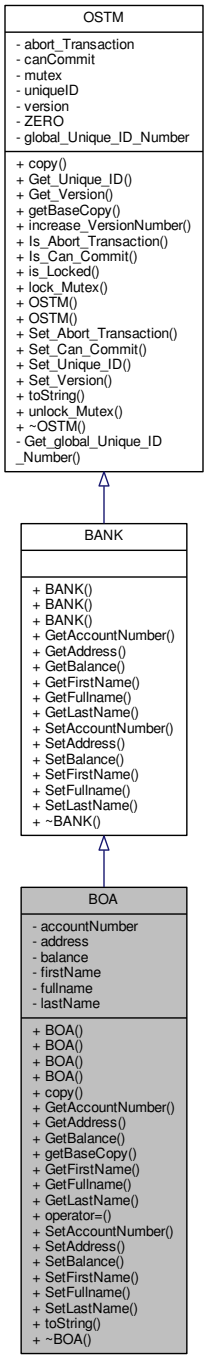
4.3 BOA Class Reference

```
#include <BOA.h>
```

Inheritance diagram for BOA:



Collaboration diagram for BOA:



Public Member Functions

- BOA ()
- BOA (int accountNumber, double balance, std::string firstName, std::string lastName, std::string address)
- BOA (std::shared_ptr< BANK > obj, int _version, int _unique_id)
- BOA (const BOA &orig)
- virtual void copy (std::shared_ptr< OSTM > to, std::shared_ptr< OSTM > from)

OSTM required virtual method for deep copy.

- virtual int GetAccountNumber () const
- virtual std::string GetAddress () const
- virtual double GetBalance () const
- 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 std::string GetFirstName () const
- virtual std::string GetFullname () const
- virtual std::string GetLastName () const
- BOA operator= (const BOA &orig)
- virtual void SetAccountNumber (int accountNumber)
- virtual void SetAddress (std::string address)
- virtual void SetBalance (double balance)
- virtual void SetFirstName (std::string firstName)
- virtual void SetFullname (std::string fullname)
- virtual void SetLastName (std::string lastName)
- virtual void toString ()

  OSTM required virtual method for display object.
- virtual ~BOA ()

Private Attributes

- int accountNumber
- std::string address
- double balance
- std::string firstName
- std::string fullname
- std::string lastName

4.3.1 Detailed Description

Definition at line 18 of file BOA.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 BOA::BOA ( ) [inline]

Definition at line 24 of file BOA.h.

References accountNumber, address, balance, firstName, fullname, and lastName.

Referenced by BOA(), and getBaseCopy().

```
00024          : BANK() {
00025        this->accountNumber = 0;
00026        this->balance = 50;
00027        this->firstName = "Joe";
00028        this->lastName = "Blog";
00029        this->address = "High street, Carlow";
00030        this->fullname = firstName + " " + lastName;
00031    };
```

4.3.2.2 BOA::BOA (int *accountNumber,* double *balance,* std::string *firstName,* std::string *lastName,* std::string *address*) [inline]

Definition at line 35 of file BOA.h.

References accountNumber, address, balance, firstName, fullname, and lastName.

```
00035                                                                                                        :
      BANK() {
00036         this->accountNumber = accountNumber;
00037         this->balance = balance;
00038         this->firstName = firstName;
00039         this->lastName = lastName;
00040         this->address = address;
00041         this->fullname = firstName + " " + lastName;
00042     };
```

4.3.2.3 BOA::BOA (std::shared_ptr< BANK > *obj,* int *_version,* int *_unique_id*) [inline]

Definition at line 46 of file BOA.h.

References accountNumber, address, balance, BOA(), firstName, fullname, and lastName.

```
00046                                                                   : BANK(_version, _unique_id) {
00047
00048         this->accountNumber = obj->GetAccountNumber();
00049         this->balance = obj->GetBalance();
00050         this->firstName = obj->GetFirstName();
00051         this->lastName = obj->GetLastName();
00052         this->address = obj->GetAddress();
00053         this->fullname = obj->GetFirstName() + " " + obj->GetLastName();
00054     };
```

Here is the call graph for this function:

4.3.2.4 BOA::BOA (const BOA & *orig*)

Definition at line 12 of file BOA.cpp.

```
00012                          {
00013 }
```

4.3.2.5 BOA::∼BOA ( ) [virtual]

Definition at line 15 of file BOA.cpp.

Referenced by operator=().

```
00015            {
00016 }
```

4.3.3 Member Function Documentation

4.3.3.1 void BOA::copy (std::shared_ptr< OSTM > *from*, std::shared_ptr< OSTM > *to*) [virtual]

OSTM required virtual method for deep copy.

Reimplemented from OSTM.

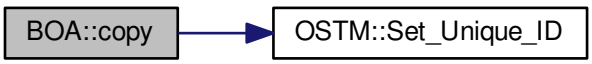
Definition at line 34 of file BOA.cpp.

References OSTM::Set_Unique_ID().

Referenced by operator=().

```
00034                                                                   {
00035
00036     std::shared_ptr<BOA> objTO = std::dynamic_pointer_cast<BOA>(to);
00037     std::shared_ptr<BOA> objFROM = std::dynamic_pointer_cast<BOA>(from);
00038     objTO->Set_Unique_ID(objFROM->Get_Unique_ID());
00039     objTO->Set_Version(objFROM->Get_Version());
00040     objTO->SetAccountNumber(objFROM->GetAccountNumber());
00041     objTO->SetBalance(objFROM->GetBalance());
00042
00043 }
```

Here is the call graph for this function:



4.3.3.2 int BOA::GetAccountNumber ( ) const [virtual]

Reimplemented from BANK.

Definition at line 74 of file BOA.cpp.

References accountNumber.

Referenced by operator=(), and toString().

```
00074                                   {
00075     return accountNumber;
00076 }
```

4.3.3.3 std::string BOA::GetAddress ( ) const [virtual]

Reimplemented from BANK.

Definition at line 58 of file BOA.cpp.

References address.

Referenced by operator=().

```
00058                                   {
00059     return address;
00060 }
```

4.3.3.4 double BOA::GetBalance ( ) const [virtual]

Reimplemented from BANK.

Definition at line 66 of file BOA.cpp.

References balance.

Referenced by operator=(), and toString().

```
00066                                   {
00067     return balance;
00068 }
```

4.3.3.5 std::shared_ptr< OSTM > BOA::getBaseCopy (std::shared_ptr< OSTM > *object*) [virtual]

OSTM required virtual method for returning a pointer that is copy of the original pointer.

Reimplemented from OSTM.

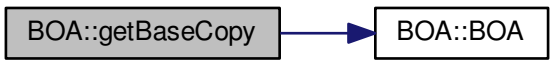
Definition at line 22 of file BOA.cpp.

References BOA().

Referenced by operator=().

```
00023 {
00024         std::shared_ptr<BANK> objTO = std::dynamic_pointer_cast<BANK>(object);
00025     std::shared_ptr<BANK> obj(new BOA(objTO,object->Get_Version(),object->Get_Unique_ID()));
00026         std::shared_ptr<OSTM> ostm_obj = std::dynamic_pointer_cast<OSTM>(obj);
00027     return ostm_obj;
00028 }
```

Here is the call graph for this function:



4.3.3.6 std::string BOA::GetFirstName ( ) const [virtual]

Reimplemented from BANK.

Definition at line 90 of file BOA.cpp.

References firstName.

Referenced by operator=(), and toString().

```
00090                                   {
00091     return firstName;
00092 }
```

4.3.3.7 std::string BOA::GetFullname ( ) const [virtual]

Reimplemented from BANK.

Definition at line 98 of file BOA.cpp.

References fullname.

Referenced by operator=().

```
00098                                   {
00099     return fullname;
00100 }
```

4.3.3.8 std::string BOA::GetLastName ( ) const [virtual]

Reimplemented from BANK.

Definition at line 82 of file BOA.cpp.

References lastName.

Referenced by operator=(), and toString().

```
00082                                   {
00083     return lastName;
00084 }
```

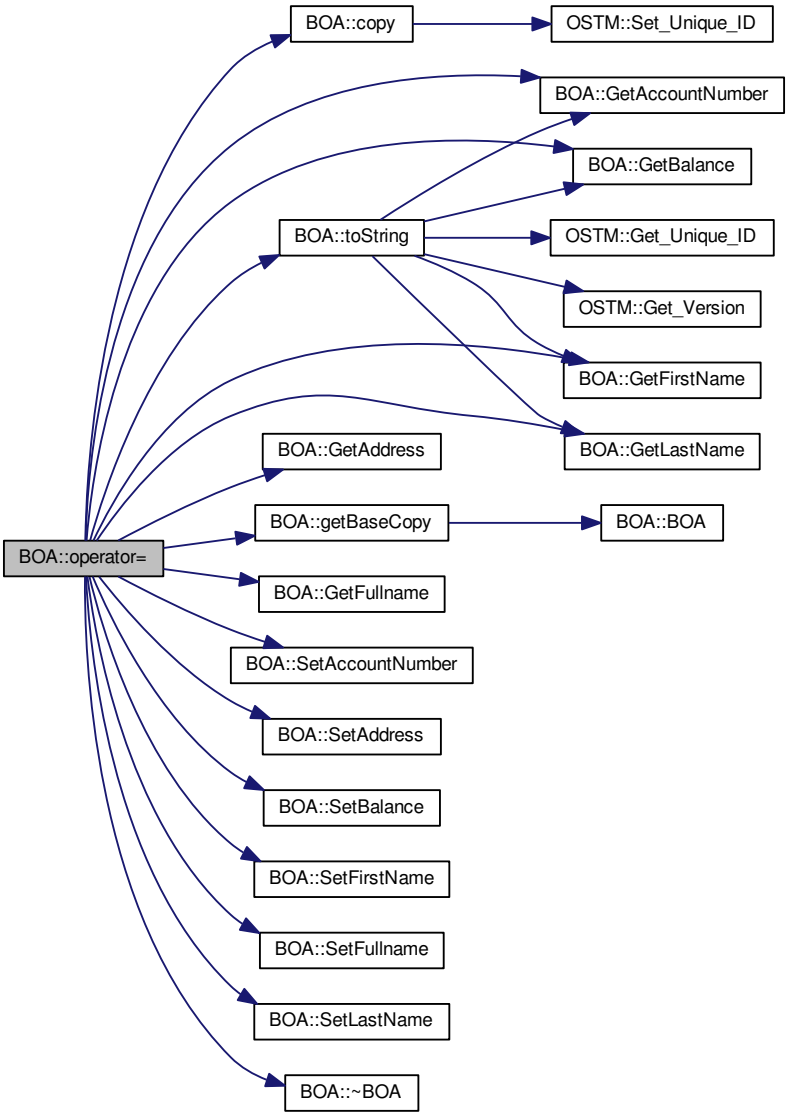
4.3.3.9 BOA BOA::operator= (const BOA & *orig*) [inline]

Definition at line 64 of file BOA.h.

References accountNumber, address, balance, copy(), firstName, fullname, GetAccountNumber(), GetAddress(), GetBalance(), getBaseCopy(), GetFirstName(), GetFullname(), GetLastName(), lastName, SetAccountNumber(), SetAddress(), SetBalance(), SetFirstName(), SetFullname(), SetLastName(), toString(), and ~BOA().

```
00064                                   {
00065     };
```

Here is the call graph for this function:



4.3.3.10 void BOA::SetAccountNumber (int *accountNumber*) [virtual]

Reimplemented from BANK.

Definition at line 70 of file BOA.cpp.

References accountNumber.

Referenced by operator=().

```
00070                                          {
00071     this->accountNumber = accountNumber;
00072 }
```

4.3.3.11 void BOA::SetAddress (std::string *address*) [virtual]

Reimplemented from BANK.

Definition at line 54 of file BOA.cpp.

References address.

Referenced by operator=().

```
00054                                     {
00055     this->address = address;
00056 }
```

4.3.3.12 void BOA::SetBalance (double *balance*) [virtual]

Reimplemented from BANK.

Definition at line 62 of file BOA.cpp.

References balance.

Referenced by operator=().

```
00062                                   {
00063     this->balance = balance;
00064 }
```

4.3.3.13 void BOA::SetFirstName (std::string *firstName*) [virtual]

Reimplemented from BANK.

Definition at line 86 of file BOA.cpp.

References firstName.

Referenced by operator=().

```
00086                                        {
00087     this->firstName = firstName;
00088 }
```

4.3.3.14 void BOA::SetFullname (std::string *fullname*) [virtual]

Reimplemented from BANK.

Definition at line 94 of file BOA.cpp.

References fullname.

Referenced by operator=().

```
00094                                       {
00095     this->fullname = fullname;
00096 }
```

4.3.3.15 void BOA::SetLastName (std::string *lastName*) [virtual]

Reimplemented from BANK.

Definition at line 78 of file BOA.cpp.

References lastName.

Referenced by operator=().

```
00078                                       {
00079     this->lastName = lastName;
00080 }
```

4.3.3.16 void BOA::toString ( ) [virtual]

OSTM required virtual method for display object.

Reimplemented from OSTM.

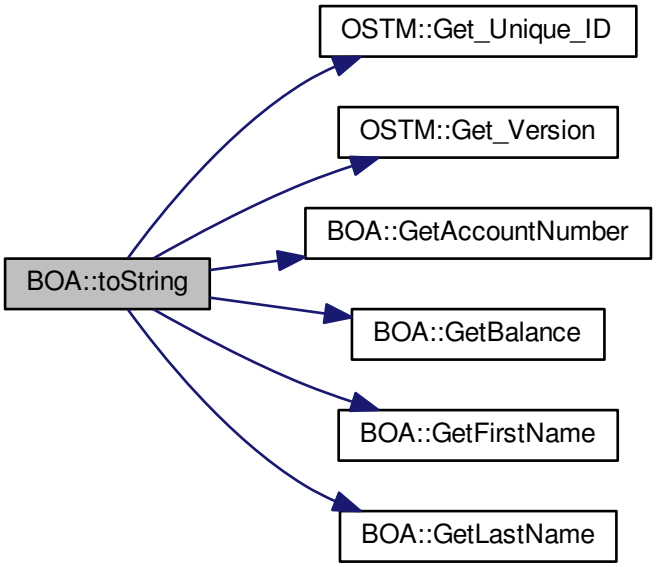
Definition at line 48 of file BOA.cpp.

References OSTM::Get_Unique_ID(), OSTM::Get_Version(), GetAccountNumber(), GetBalance(), GetFirstName(), and GetLastName().

Referenced by operator=().

```
00049 {
00050     // std::cout << "\nUnique ID : " << this->GetUniqueID() << "\nInt value : " << this->GetV_int() <<
       "\nDouble value : " << this->GetV_double() << "\nFloat value : " << this->GetV_float() << "\nString value : " <<
       this->GetV_string()  << "\nVersion number : " << this->GetVersion() << "\nLoad Counter : "<<
       this->GetLoadCounter() << "\nWrite Counter : "<< this->GetWriteCounter() << std::endl;
00051      std::cout << "\nBOA BANK" << "\nUnique ID : " << this->Get_Unique_ID() << "\nInt account
       : " << this->GetAccountNumber() << "\nDouble value : " << this->
      GetBalance() << "\nFirst name: " << this->GetFirstName() << "\nLast name : " <<
      this->GetLastName()  << "\nVersion number : " << this->Get_Version() << std::endl;
00052 }
```

Here is the call graph for this function:



4.3.4 Member Data Documentation

4.3.4.1 int BOA::accountNumber [private]

Definition at line 98 of file BOA.h.

Referenced by BOA(), GetAccountNumber(), operator=(), and SetAccountNumber().

4.3.4.2 std::string BOA::address [private]

Definition at line 100 of file BOA.h.

Referenced by BOA(), GetAddress(), operator=(), and SetAddress().

4.3.4.3 double BOA::balance [private]

Definition at line 99 of file BOA.h.

Referenced by BOA(), GetBalance(), operator=(), and SetBalance().

4.3.4.4 std::string BOA::firstName [private]

Definition at line 96 of file BOA.h.

Referenced by BOA(), GetFirstName(), operator=(), and SetFirstName().

4.3.4.5 std::string BOA::fullname [private]

Definition at line 95 of file BOA.h.

Referenced by BOA(), GetFullname(), operator=(), and SetFullname().

4.3.4.6 std::string BOA::lastName [private]

Definition at line 97 of file BOA.h.

Referenced by BOA(), GetLastName(), operator=(), and SetLastName().

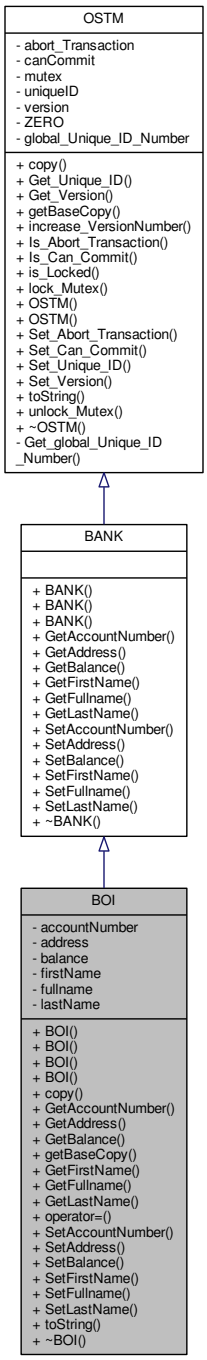
The documentation for this class was generated from the following files:

- BOA.h
- BOA.cpp

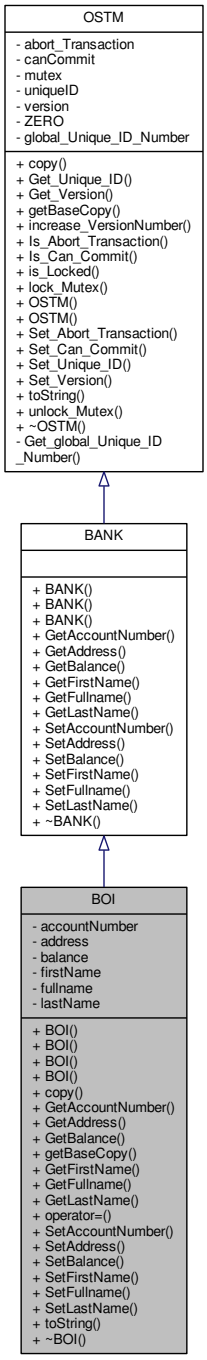
4.4 BOI Class Reference

```
#include <BOI.h>
```

Inheritance diagram for BOI:



Collaboration diagram for BOI:



Public Member Functions

- BOI ()
- BOI (int accountNumber, double balance, std::string firstName, std::string lastName, std::string address)
- BOI (std::shared_ptr< BOI > obj, int _version, int _unique_id)
- BOI (const BOI &orig)
- virtual void copy (std::shared_ptr< OSTM > to, std::shared_ptr< OSTM > from)

OSTM required virtual method for deep copy.

- virtual int GetAccountNumber () const
- virtual std::string GetAddress () const
- virtual double GetBalance () const
- 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 std::string GetFirstName () const
- virtual std::string GetFullname () const
- virtual std::string GetLastName () const
- BOI operator= (const BOI &orig)
- virtual void SetAccountNumber (int accountNumber)
- virtual void SetAddress (std::string address)
- virtual void SetBalance (double balance)
- virtual void SetFirstName (std::string firstName)
- virtual void SetFullname (std::string fullname)
- virtual void SetLastName (std::string lastName)
- virtual void toString ()

  OSTM required virtual method for display object.
- virtual ~BOI ()

Private Attributes

- int accountNumber
- std::string address
- double balance
- std::string firstName
- std::string fullname
- std::string lastName

4.4.1 Detailed Description

Definition at line 19 of file BOI.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 BOI::BOI ( ) [inline]

Definition at line 24 of file BOI.h.

References accountNumber, address, balance, firstName, fullname, and lastName.

Referenced by BOI(), and getBaseCopy().

```
00024          : BANK()
00025     {
00026         this->accountNumber = 0;
00027         this->balance = 50;
00028         this->firstName = "Joe";
00029         this->lastName = "Blog";
00030         this->address = "High street, Carlow";
00031         this->fullname = firstName + " " + lastName;
00032
00033     }
```

4.4.2.2 BOI::BOI (int *accountNumber,* double *balance,* std::string *firstName,* std::string *lastName,* std::string *address*) [inline]

Definition at line 37 of file BOI.h.

References accountNumber, address, balance, firstName, fullname, and lastName.

```
00037                                                                                                           :
      BANK()
00038     {
00039         this->accountNumber = accountNumber;
00040         this->balance = balance;
00041         this->firstName = firstName;
00042         this->lastName = lastName;
00043         this->address = address;
00044         this->fullname = firstName + " " + lastName;
00045     };
```

4.4.2.3 BOI::BOI (std::shared_ptr< BOI > *obj,* int *_version,* int *_unique_id*) [inline]

Definition at line 49 of file BOI.h.

References accountNumber, address, balance, BOI(), firstName, fullname, and lastName.

```
00049                                                                     : BANK(_version, _unique_id)
00050     {
00051         this->accountNumber = obj->GetAccountNumber();
00052         this->balance = obj->GetBalance();
00053         this->firstName = obj->GetFirstName();
00054         this->lastName = obj->GetLastName();
00055         this->address = obj->GetAddress();
00056         this->fullname = obj->GetFirstName() + " " + obj->GetLastName();
00057     };
```

Here is the call graph for this function:

4.4.2.4 BOI::BOI (const BOI & *orig*)

Definition at line 15 of file BOI.cpp.

```
00015                         {
00016 }
```

4.4.2.5 BOI::∼BOI ( ) [virtual]

Definition at line 12 of file BOI.cpp.

Referenced by operator=().

```
00012             {
00013 }
```

4.4.3 Member Function Documentation

4.4.3.1 void BOI::copy (std::shared_ptr< OSTM > *from*, std::shared_ptr< OSTM > *to*) [virtual]

OSTM required virtual method for deep copy.

Reimplemented from OSTM.

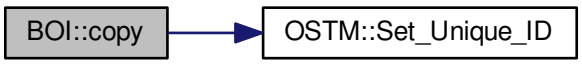
Definition at line 35 of file BOI.cpp.

References OSTM::Set_Unique_ID().

Referenced by operator=().

```
00035                                                                   {
00036
00037     std::shared_ptr<BOI> objTO = std::dynamic_pointer_cast<BOI>(to);
00038     std::shared_ptr<BOI> objFROM = std::dynamic_pointer_cast<BOI>(from);
00039     objTO->Set_Unique_ID(objFROM->Get_Unique_ID());
00040     objTO->Set_Version(objFROM->Get_Version());
00041     objTO->SetAccountNumber(objFROM->GetAccountNumber());
00042     objTO->SetBalance(objFROM->GetBalance());
00043 }
```

Here is the call graph for this function:



4.4.3.2 int BOI::GetAccountNumber ( ) const [virtual]

Reimplemented from BANK.

Definition at line 73 of file BOI.cpp.

References accountNumber.

Referenced by operator=(), and toString().

```
00073                                   {
00074     return accountNumber;
00075 }
```

4.4.3.3 std::string BOI::GetAddress ( ) const [virtual]

Reimplemented from BANK.

Definition at line 57 of file BOI.cpp.

References address.

Referenced by operator=().

```
00057                                   {
00058     return address;
00059 }
```

4.4.3.4 double BOI::GetBalance ( ) const [virtual]

Reimplemented from BANK.

Definition at line 65 of file BOI.cpp.

References balance.

Referenced by operator=(), and toString().

```
00065                                 {
00066     return balance;
00067 }
```

4.4.3.5 std::shared_ptr< OSTM > BOI::getBaseCopy (std::shared_ptr< OSTM > *object*) [virtual]

OSTM required virtual method for returning a pointer that is copy of the original pointer.

Reimplemented from OSTM.

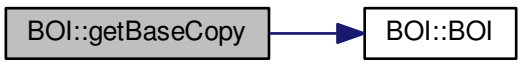
Definition at line 22 of file BOI.cpp.

References BOI().

Referenced by operator=().

```
00023 {
00024
00025     std::shared_ptr<BOI> objTO = std::dynamic_pointer_cast<BOI>(object);
00026     std::shared_ptr<BOI> obj(new BOI(objTO,object->Get_Version(),object->Get_Unique_ID()));
00027     std::shared_ptr<OSTM> ostm_obj = std::dynamic_pointer_cast<OSTM>(obj);
00028     return ostm_obj;
00029 }
```

Here is the call graph for this function:



4.4.3.6 std::string BOI::GetFirstName ( ) const [virtual]

Reimplemented from BANK.

Definition at line 89 of file BOI.cpp.

References firstName.

Referenced by operator=(), and toString().

```
00089                                   {
00090     return firstName;
00091 }
```

4.4.3.7 std::string BOI::GetFullname ( ) const [virtual]

Reimplemented from BANK.

Definition at line 97 of file BOI.cpp.

References fullname.

Referenced by operator=().

```
00097                                   {
00098     return fullname;
00099 }
```

4.4.3.8 std::string BOI::GetLastName ( ) const [virtual]

Reimplemented from BANK.

Definition at line 81 of file BOI.cpp.

References lastName.

Referenced by operator=(), and toString().

```
00081                                   {
00082     return lastName;
00083 }
```

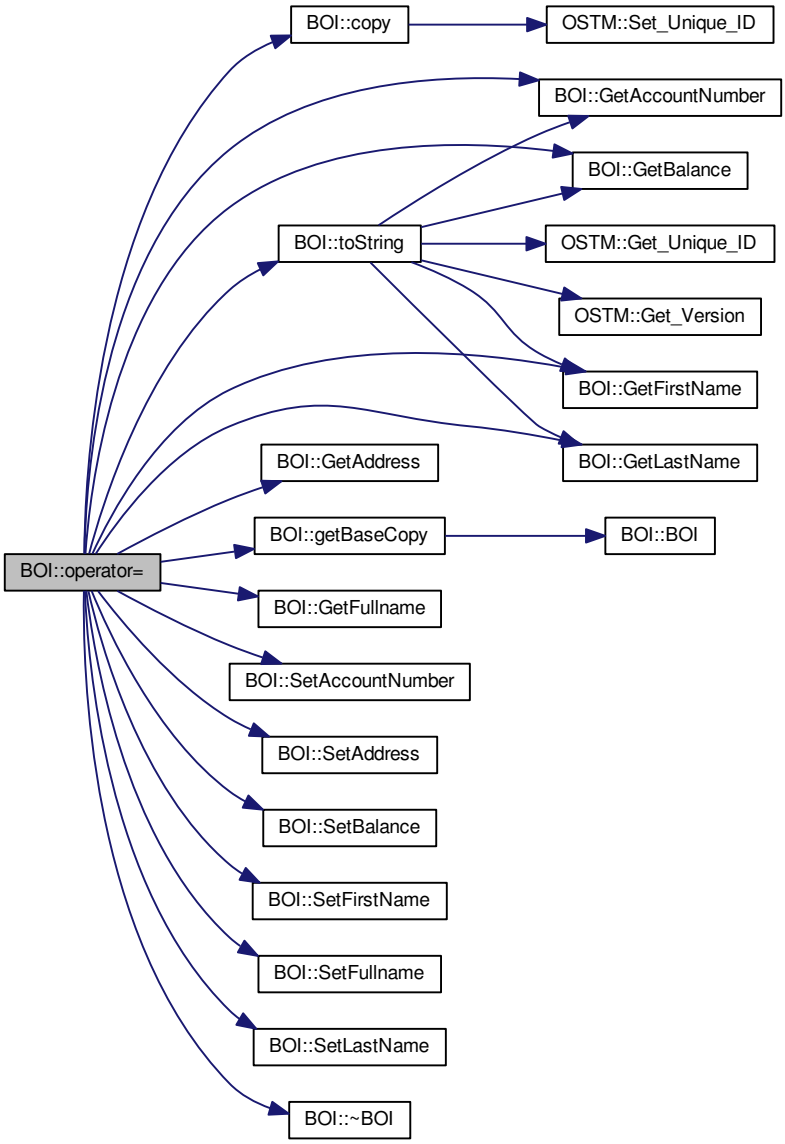
4.4.3.9 BOI BOI::operator= (const BOI & *orig*) [inline]

Definition at line 65 of file BOI.h.

References accountNumber, address, balance, copy(), firstName, fullname, GetAccountNumber(), GetAddress(), GetBalance(), getBaseCopy(), GetFirstName(), GetFullname(), GetLastName(), lastName, SetAccountNumber(), SetAddress(), SetBalance(), SetFirstName(), SetFullname(), SetLastName(), toString(), and ~BOI().

```
00065 {};
```

Here is the call graph for this function:



4.4.3.10 void BOI::SetAccountNumber (int *accountNumber*) [virtual]

Reimplemented from BANK.

Definition at line 69 of file BOI.cpp.

References accountNumber.

Referenced by operator=().

```
00069                                          {
00070     this->accountNumber = accountNumber;
00071 }
```

4.4.3.11 void BOI::SetAddress (std::string *address*) [virtual]

Reimplemented from BANK.

Definition at line 53 of file BOI.cpp.

References address.

Referenced by operator=().

```
00053                                       {
00054     this->address = address;
00055 }
```

4.4.3.12 void BOI::SetBalance (double *balance*) [virtual]

Reimplemented from BANK.

Definition at line 61 of file BOI.cpp.

References balance.

Referenced by operator=().

```
00061                                     {
00062     this->balance = balance;
00063 }
```

4.4.3.13 void BOI::SetFirstName (std::string *firstName*) [virtual]

Reimplemented from BANK.

Definition at line 85 of file BOI.cpp.

References firstName.

Referenced by operator=().

```
00085                                          {
00086     this->firstName = firstName;
00087 }
```

4.4.3.14 void BOI::SetFullname (std::string *fullname*) [virtual]

Reimplemented from BANK.

Definition at line 93 of file BOI.cpp.

References fullname.

Referenced by operator=().

```
00093                                       {
00094     this->fullname = fullname;
00095 }
```

4.4.3.15 void BOI::SetLastName (std::string *lastName*) [virtual]

Reimplemented from BANK.

Definition at line 77 of file BOI.cpp.

References lastName.

Referenced by operator=().

```
00077                                       {
00078     this->lastName = lastName;
00079 }
```

4.4.3.16 void BOI::toString ( ) [virtual]

OSTM required virtual method for display object.

Reimplemented from OSTM.

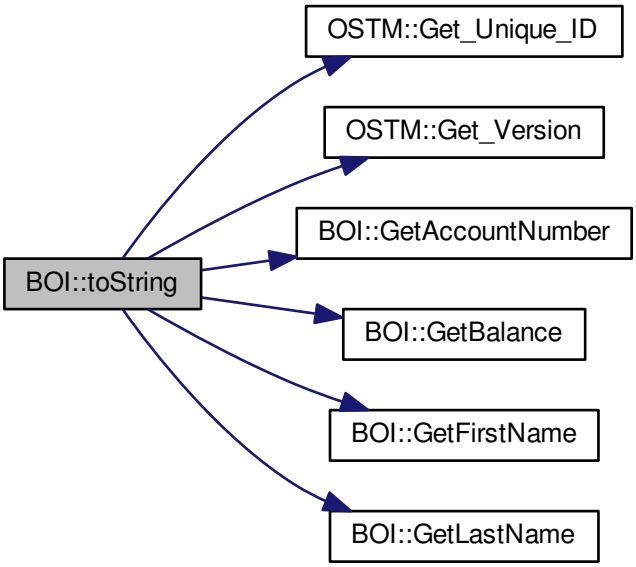
Definition at line 49 of file BOI.cpp.

References OSTM::Get_Unique_ID(), OSTM::Get_Version(), GetAccountNumber(), GetBalance(), GetFirstName(), and GetLastName().

Referenced by operator=().

```
00050 {
00051    std::cout << "\nBOI BANK" << "\nUnique ID : " << this->Get_Unique_ID() << "\nInt account :
      " << this->GetAccountNumber() << "\nDouble value : " << this->
      GetBalance() << "\nFirst name: " << this->GetFirstName() << "\nLast name : " <<
      this->GetLastName()  << "\nVersion number : " << this->Get_Version() << std::endl;
00052 }
```

Here is the call graph for this function:



4.4.4 Member Data Documentation

4.4.4.1 int BOI::accountNumber [private]

Definition at line 99 of file BOI.h.

Referenced by BOI(), GetAccountNumber(), operator=(), and SetAccountNumber().

4.4.4.2 std::string BOI::address [private]

Definition at line 101 of file BOI.h.

Referenced by BOI(), GetAddress(), operator=(), and SetAddress().

4.4.4.3 double BOI::balance [private]

Definition at line 100 of file BOI.h.

Referenced by BOI(), GetBalance(), operator=(), and SetBalance().

4.4.4.4 std::string BOI::firstName [private]

Definition at line 97 of file BOI.h.

Referenced by BOI(), GetFirstName(), operator=(), and SetFirstName().

4.4.4.5 std::string BOI::fullname [private]

Definition at line 96 of file BOI.h.

Referenced by BOI(), GetFullname(), operator=(), and SetFullname().

4.4.4.6 std::string BOI::lastName [private]

Definition at line 98 of file BOI.h.

Referenced by BOI(), GetLastName(), operator=(), and SetLastName().

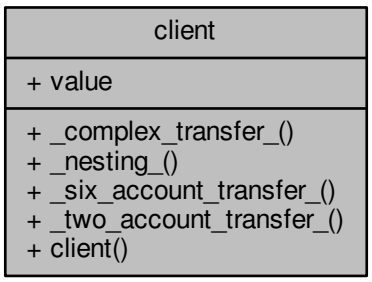
The documentation for this class was generated from the following files:

- BOI.h
- BOI.cpp

4.5 client Class Reference

```
#include <client.h>
```

Collaboration diagram for client:



Public Member Functions

- void _complex_transfer_ (std::shared_ptr< OSTM > _from_, std::shared_ptr< OSTM > _from_two_, std↩ ::vector< std::shared_ptr< OSTM >> _customer_vec, TM &_tm, double _amount)
- void _nesting_ (std::shared_ptr< OSTM > _to_, std::shared_ptr< OSTM > _from_, TM &tm, double _↩ amount)
- void _six_account_transfer_ (std::shared_ptr< OSTM > _to_, std::shared_ptr< OSTM > _from_one_, std↩ ::shared_ptr< OSTM > _from_two_, std::shared_ptr< OSTM > _from_three_, std::shared_ptr< OSTM > _from_four_, std::shared_ptr< OSTM > _from_five_, TM &_tm, double _amount)
- void _two_account_transfer_ (std::shared_ptr< OSTM > _to_, std::shared_ptr< OSTM > _from_, TM &tm, double _amount)
- client (int value)

Public Attributes

- int value = 0

4.5.1 Detailed Description

Definition at line 34 of file client.h.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 client::client (int *value*) [inline]

Definition at line 39 of file client.h.

References value.

```
00039 { this->value = value; };
```

4.5.3 Member Function Documentation

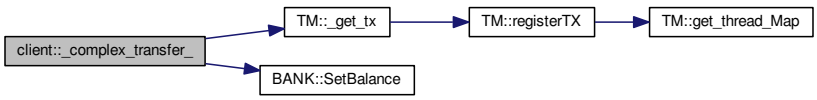
4.5.3.1 void client::_complex_transfer_ (std::shared_ptr< OSTM > *_from_*, std::shared_ptr< OSTM > *_from_two_*, std::vector< std::shared_ptr< OSTM >> *_customer_vec*, TM & *_tm*, double *_amount*) [inline]

Definition at line 236 of file client.h.

References TM::_get_tx(), and BANK::SetBalance().

```
00236
                                                    {
00237     std::shared_ptr<TX> tx = _tm._get_tx();
00238     /* Register the two single account*/
00239     tx->_register(_from_);
00240     tx->_register(_from_two_);
00241     /* Declare required pointers */
00242     std::shared_ptr<OSTM> _FROM_OSTM_ONE_, _FROM_OSTM_TWO_, _TO_OSTM_;
00243     std::shared_ptr<BANK> _FROM_, _FROM_TWO_, _TO_;
00244
00245     bool done = false;
00246     try {
00247         while (!done) {
00248             for (auto&& obj : _customer_vec) {
00249                 /* Register customers accounts from the collection (vector) */
00250                 tx->_register(obj);
00251                 /* From std::shared_ptr<OSTM> to std::shared_ptr<BANK> to access the virtual methods  */
00252                 _FROM_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_));
00253                 _FROM_TWO_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_two_));
00254                 _TO_ = std::dynamic_pointer_cast<BANK> (tx->load(obj));
00255                 /* Make changes with the objects */
00256                 _FROM_->SetBalance(_FROM_->GetBalance() - _amount);
00257                 _FROM_TWO_->SetBalance(_FROM_TWO_->GetBalance() - _amount);
00258                 _TO_->SetBalance(_TO_->GetBalance() + (_amount * 2));
00259                 /* From std::shared_ptr<BANK> to std::shared_ptr<OSTM> to store the memory spaces */
00260                 _FROM_OSTM_ONE_ = std::dynamic_pointer_cast<OSTM> (_FROM_);
00261                 _FROM_OSTM_TWO_ = std::dynamic_pointer_cast<OSTM> (_FROM_TWO_);
00262                 _TO_OSTM_ = std::dynamic_pointer_cast<OSTM> (_TO_);
00263                 /* Store changes */
00264                 tx->store(_FROM_OSTM_ONE_);
00265                 tx->store(_FROM_OSTM_TWO_);
00266                 tx->store(_TO_OSTM_);
00267             }
00268             /* Commit changes */
00269             done = tx->commit();
00270         }
00271     } catch (std::runtime_error& e) {
00272         std::cout << e.what() << std::endl;
00273     }
00274 }
```

Here is the call graph for this function:



4.5.3.2 void client::_nesting_ (std::shared_ptr< OSTM > *_to_*, std::shared_ptr< OSTM > *_from_*, TM & *tm*, double *_amount*) [inline]

Definition at line 89 of file client.h.

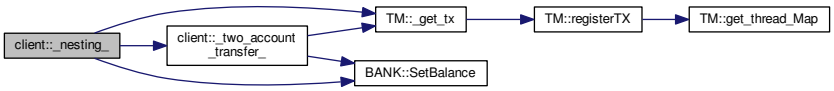
References TM::_get_tx(), _two_account_transfer_(), and BANK::SetBalance().

Referenced by MyTestCAse::nested_transaction_object_test().

```
00089                                                                                               {
00090     std::shared_ptr<TX> tx = tm._get_tx();
00091     /*
00092      * Register the two single account
00093      */
00094     tx->_register(_to_);
00095     tx->_register(_from_);
00096     /*
00097      * Declare required pointers
00098      */
00099     std::shared_ptr<BANK> _TO_BANK_, _FROM_BANK_;
00100     std::shared_ptr<OSTM> _TO_OSTM_, _FROM_OSTM_;
00101 
00102 
00103     bool done = false;
00104     try {
00105         while (!done) {
00106             /*
00107              * From std::shared_ptr<OSTM> to std::shared_ptr<BANK> to access the virtual methods
00108              */
00109             _TO_BANK_ = std::dynamic_pointer_cast<BANK> (tx->load(_to_));
00110             _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_));
00111             /*
00112              * Make changes with the objects
00113              */
00114             _TO_BANK_->SetBalance(_TO_BANK_->GetBalance() + _amount);
00115             _FROM_BANK_->SetBalance(_FROM_BANK_->GetBalance() - _amount);
00116             /*
00117              * From std::shared_ptr<BANK> to std::shared_ptr<OSTM> to store the memory spaces
00118              */
00119             _TO_OSTM_ = std::dynamic_pointer_cast<OSTM> (_TO_BANK_);
00120             _FROM_OSTM_ = std::dynamic_pointer_cast<OSTM> (_FROM_BANK_);
00121             /*
00122              * Store changes
00123              */
00124             tx->store(_TO_OSTM_);
00125             tx->store(_FROM_OSTM_);
00126 
00127             /*
00128              * NESTED TRANSACTION
00129              */
00130             std::shared_ptr<TX> txTwo = tm._get_tx();
00131 
00132             bool nestedDone = false;
00133             while (!nestedDone) {
00134                 _TO_BANK_ = std::dynamic_pointer_cast<BANK> (txTwo->load(_to_));
00135                 _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (txTwo->load(_from_));
00136                 /*
00137                  * Make changes with the objects
00138                  */
00139                 _TO_BANK_->SetBalance(_TO_BANK_->GetBalance() + _amount);
00140                 _FROM_BANK_->SetBalance(_FROM_BANK_->GetBalance() - _amount);
00141                 /*
00142                  * From std::shared_ptr<BANK> to std::shared_ptr<OSTM> to store the memory spaces
00143                  */
```

```
                _TO_OSTM_ = std::dynamic_pointer_cast<OSTM> (_TO_BANK_);
                _FROM_OSTM_ = std::dynamic_pointer_cast<OSTM> (_FROM_BANK_);
                /*
                 * Store changes
                 */
                txTwo->store(_TO_OSTM_);
                txTwo->store(_FROM_OSTM_);
                /*
                 * NESTED TRANSACTION IN THE NESTED TRANSACTION
                 * _two_account_transfer_ function call
                 */
                _two_account_transfer_(_to_, _from_, tm, _amount);

                nestedDone = txTwo->commit();
            }

            /*
             * Commit changes
             */
            done = tx->commit();
        }
    } catch (std::runtime_error& e) {
        std::cout << e.what() << std::endl;
    }
}
```

Here is the call graph for this function:



4.5.3.3 void client::_six_account_transfer_ (std::shared_ptr< OSTM > *_to_*, std::shared_ptr< OSTM > *_from_one_*, std::shared_ptr< OSTM > *_from_two_*, std::shared_ptr< OSTM > *_from_three_*, std::shared_ptr< OSTM > *_from_four_*, std::shared_ptr< OSTM > *_from_five_*, TM & *_tm*, double *_amount*) [inline]

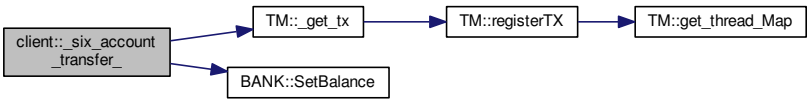
Definition at line 170 of file client.h.

References TM::_get_tx(), and BANK::SetBalance().

```
00170

                                {
00171     std::shared_ptr<TX> tx = _tm._get_tx();
00172     /*
00173      * Register the two single account
00174      */
00175     tx->_register(_to_);
00176     tx->_register(_from_one_);
00177     tx->_register(_from_two_);
00178     tx->_register(_from_three_);
00179     tx->_register(_from_four_);
00180     tx->_register(_from_five_);
00181
00182     /*
00183      * Required pointers to use in transaction
00184      */
00185     std::shared_ptr<OSTM> _TO_OSTM, _FROM_ONE_OSTM, _FROM_TWO_OSTM, _FROM_THREE_OSTM, _FROM_FOUR_OSTM,
      _FROM_FIVE_OSTM;
00186     std::shared_ptr<BANK> _TO_, _FROM_ONE_, _FROM_TWO_, _FROM_THREE_, _FROM_FOUR_, _FROM_FIVE_;
00187     try {
00188         bool done = false;
00189         while (!done) {
00190             /*
00191              * From std::shared_ptr<OSTM> to std::shared_ptr<BANK> to access the virtual methods
00192              */
00193             _TO_ = std::dynamic_pointer_cast<BANK> (tx->load(_to_));
00194             _FROM_ONE_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_one_));
00195             _FROM_TWO_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_two_));
```

```
            _FROM_THREE_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_three_));
            _FROM_FOUR_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_four_));
            _FROM_FIVE_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_five_));
            /*
             * Make changes with the objects
             */
            _TO_->SetBalance(_TO_->GetBalance() + (_amount * 5));
            _FROM_ONE_->SetBalance(_FROM_ONE_->GetBalance() - _amount);
            _FROM_TWO_->SetBalance(_FROM_TWO_->GetBalance() - _amount);
            _FROM_THREE_->SetBalance(_FROM_THREE_->GetBalance() - _amount);
            _FROM_FOUR_->SetBalance(_FROM_FOUR_->GetBalance() - _amount);
            _FROM_FIVE_->SetBalance(_FROM_FIVE_->GetBalance() - _amount);
            /*
             * From std::shared_ptr<BANK> to std::shared_ptr<OSTM> to store the memory spaces
             */
            _TO_OSTM = std::dynamic_pointer_cast<OSTM> (_TO_);
            _FROM_ONE_OSTM = std::dynamic_pointer_cast<OSTM> (_FROM_ONE_);
            _FROM_TWO_OSTM = std::dynamic_pointer_cast<OSTM> (_FROM_TWO_);
            _FROM_THREE_OSTM = std::dynamic_pointer_cast<OSTM> (_FROM_THREE_);
            _FROM_FOUR_OSTM = std::dynamic_pointer_cast<OSTM> (_FROM_FOUR_);
            _FROM_FIVE_OSTM = std::dynamic_pointer_cast<OSTM> (_FROM_FIVE_);
            /*
             * Store changes
             */
            tx->store(_TO_OSTM);
            tx->store(_FROM_ONE_OSTM);
            tx->store(_FROM_TWO_OSTM);
            tx->store(_FROM_THREE_OSTM);
            tx->store(_FROM_FOUR_OSTM);
            tx->store(_FROM_FIVE_OSTM);
            /*
             * Commit changes
             */
            done = tx->commit();
        }
    } catch (std::runtime_error& e) {
        std::cout << e.what() << std::endl;
    }
}
```

Here is the call graph for this function:



4.5.3.4 void client::_two_account_transfer_ (std::shared_ptr< OSTM > *_to_*, std::shared_ptr< OSTM > *_from_*, TM & *tm*, double *_amount*) [inline]

Definition at line 41 of file client.h.

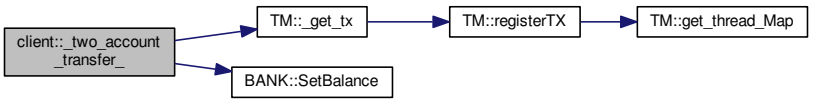
References TM::_get_tx(), and BANK::SetBalance().

Referenced by _nesting_(), MyTestCAse::two_object_transfer_complete(), and MyTestCAse::two_object_transfer↩_state_change().

```
{

    std::shared_ptr<TX> tx = tm._get_tx();
    /*
     * Register the two single account
     */
    tx->_register(_to_);
    tx->_register(_from_);
    /*
     * Declare required pointers
```

```
00051      */
00052     std::shared_ptr<BANK> _TO_BANK_, _FROM_BANK_;
00053     std::shared_ptr<OSTM> _TO_OSTM_, _FROM_OSTM_;
00054
00055     bool done = false;
00056     try {
00057         while (!done) {
00058             /*
00059              * From std::shared_ptr<OSTM> to std::shared_ptr<BANK> to access the virtual methods
00060              */
00061             _TO_BANK_ = std::dynamic_pointer_cast<BANK> (tx->load(_to_));
00062             _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_));
00063             /*
00064              * Make changes with the objects
00065              */
00066             _TO_BANK_->SetBalance(_TO_BANK_->GetBalance() + _amount);
00067             _FROM_BANK_->SetBalance(_FROM_BANK_->GetBalance() - _amount);
00068             /*
00069              * From std::shared_ptr<BANK> to std::shared_ptr<OSTM> to store the memory spaces
00070              */
00071             _TO_OSTM_ = std::dynamic_pointer_cast<OSTM> (_TO_BANK_);
00072             _FROM_OSTM_ = std::dynamic_pointer_cast<OSTM> (_FROM_BANK_);
00073             /*
00074              * Store changes
00075              */
00076             tx->store(_TO_OSTM_);
00077             tx->store(_FROM_OSTM_);
00078
00079             /*
00080              * Commit changes
00081              */
00082             done = tx->commit();
00083         }
00084     } catch (std::runtime_error& e) {
00085         std::cout << e.what() << std::endl;
00086     }
00087 }
```

Here is the call graph for this function:



4.5.4 Member Data Documentation

4.5.4.1 int client::value = 0

Definition at line 38 of file client.h.

Referenced by client().

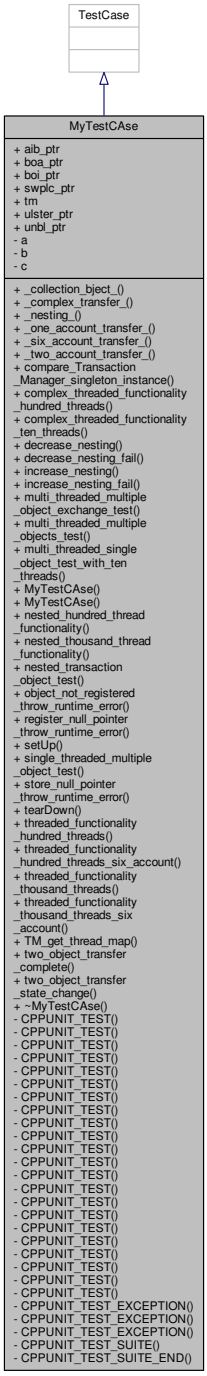
The documentation for this class was generated from the following file:

- client.h

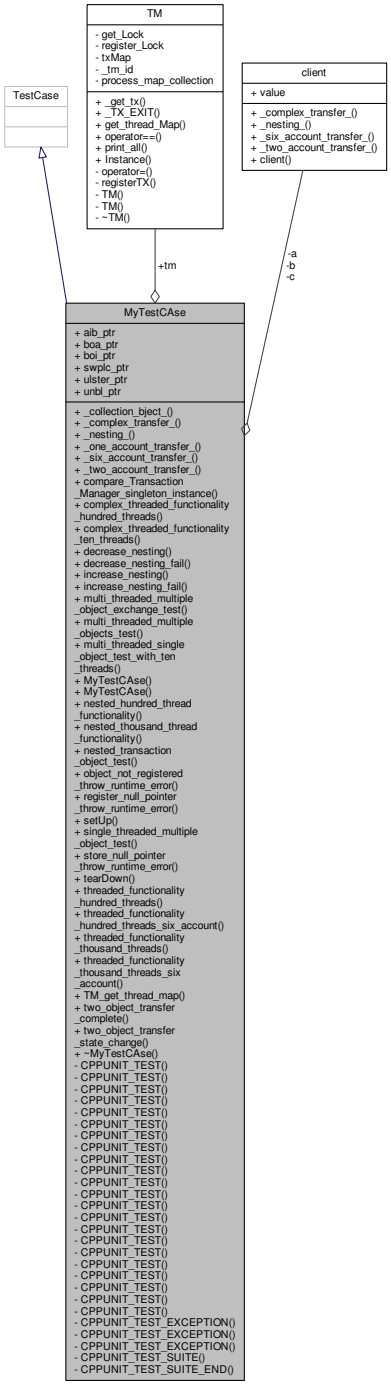
4.6 MyTestCAse Class Reference

`#include <MyTestCAse.h>`

Inheritance diagram for MyTestCAse:



Collaboration diagram for MyTestCAse:



Public Member Functions

- void _collection_bject_ (std::vector< std::shared_ptr< OSTM >> _customer_vec, TM &_tm, double _amount, int loop)

  Test with a vector collection.
- void _complex_transfer_ (std::shared_ptr< OSTM > _from_, std::shared_ptr< OSTM > _from_two_, std::vector< std::shared_ptr< OSTM >> _customer_vec, TM &_tm, double _amount)

This function use two single objects and a collection of objects in the transaction. The two single object transfer 1 - 1 (2) unit to every object in the collection.

- void _nesting_ (std::shared_ptr< OSTM > _to_, std::shared_ptr< OSTM > _from_, TM &_tm, double _amount)

  Testing nested transaction.
- void _one_account_transfer_ (std::shared_ptr< OSTM > _to_, TM &_tm, double _amount)

  one object in the trasaction
- void _six_account_transfer_ (std::shared_ptr< OSTM > _to_, std::shared_ptr< OSTM > _from_one_, std::shared_ptr< OSTM > _from_two_, std::shared_ptr< OSTM > _from_three_, std::shared_ptr< OSTM > _from_four_, std::shared_ptr< OSTM > _from_five_, TM &_tm, double _amount)

  Testing the transactions between six object.
- void _two_account_transfer_ (std::shared_ptr< OSTM > _to_, std::shared_ptr< OSTM > _from_, TM &tm, double _amount)

  Testin transaction between two pointer.
- void compare_Transaction_Manager_singleton_instance ()

  This testing function comparing the Transaction Manager to make sure the application using a Singleton object.
- void complex_threaded_functionality_hundred_threads ()
- void complex_threaded_functionality_ten_threads ()

  Testing the library consistent behavior This test transfer 1 - 1 unit by 10 threads = 10 ∗ 1 = 10 to evey object in the collection (-600) by single objects After transfer from account has -10 ∗ 600 for both single objects and + 2 unit ∗ 10 to every objects.
- void decrease_nesting ()

  Testing the nesting decrease function.
- void decrease_nesting_fail ()

  Testing the nesting decrease function to make sure the variable state changing.
- void increase_nesting ()

  Testing the nesting increase function.
- void increase_nesting_fail ()

  Testing the nesting increase function to make sure the variable state changing.
- void multi_threaded_multiple_object_exchange_test ()

  Design Manual document based tests Implementations.
- void multi_threaded_multiple_objects_test ()

  1. *Multi-threaded multiple Objects test.*
- void multi_threaded_single_object_test_with_ten_threads ()

  1. *Multi-threaded single Object test with 10 threads.*
- MyTestCAse ()
- MyTestCAse (const MyTestCAse &orig)
- void nested_hundred_thread_functionality ()

  Testing the library consistent behavior Nested threaded function : 3 level of nesting, every thread transfer 3 unit from one object to the another object so, at end of the 100 thransaction the from object transfer 100 ∗ 3 (300) to the another object 500 - 300 = 200 AND 500 + 300 = 800.
- void nested_thousand_thread_functionality ()

  Testing the library consistent behavior Nested threaded function : 3 level of nesting, every thread transfer 3 unit from one object to the another object so, at end of the 100 thransaction the from object transfer 1000 ∗ 3 (3000) to the another object 500 - 3000 = -2500 AND 500 + 3000 = 3500.
- void nested_transaction_object_test ()

  Testing the library consistent behavior This test calls the nested function, where every thread transfer 3 unit in the nested transactions. Because this test in not threaded, the 3 level deep nesting transfer 3∗20 = 60 from one object to the another object. 500 - 60 = 440 AND 500 + 60 = 560.
- void object_not_registered_throw_runtime_error ()

  The library function throws runtime error if the client application tries to load a working pointer from the library of a not registered pointer by the client application. Runtime error should be thrown.
- void register_null_pointer_throw_runtime_error ()
- void setUp ()

- void single_threaded_multiple_object_test ()

  1. *Single-threaded multiple object test.*
- void store_null_pointer_throw_runtime_error ()

  The test function throws runtime error if the client application tries to store the changed working pointer as a null pointer. Runtime error should be thrown.
- void tearDown ()
- void threaded_functionality_hundred_threads ()
- void threaded_functionality_hundred_threads_six_account ()

  Testing the library consistent behavior This test transfer 1 unit by 100 threads from five account to one account After transfer from account has - 100∗1 from the accounts, and to account has +100∗5.
- void threaded_functionality_thousand_threads ()

  Testing the library consistent behavior This test transfer 1 unit by 1000 threads = 1000 ∗1 = 100 After transfer from account has -1000, and to account has +1000.
- void threaded_functionality_thousand_threads_six_account ()

  Testing the library consistent behavior This test transfer 1 unit by 1000 threads from five account to one account After transfer from account has - 1000∗1 from the accounts, and to account has +1000∗5.
- void TM_get_thread_map ()

  This function testing the returned map from the Transaction Manager class.
- void two_object_transfer_complete ()

  Testing the library consistent behavior Transfer between two objects : the from object transfer 20 to the another object 500 - 20 = 480 AND 500 + 20 = 520.
- void two_object_transfer_state_change ()

  This function proves the objects states must change from the base values.
- virtual ∼MyTestCAse ()

Public Attributes

- std::shared_ptr< OSTM > aib_ptr
- std::shared_ptr< OSTM > boa_ptr
- std::shared_ptr< OSTM > boi_ptr
- std::shared_ptr< OSTM > swplc_ptr
- TM & tm = TM::Instance()
- std::shared_ptr< OSTM > ulster_ptr
- std::shared_ptr< OSTM > unbl_ptr

Private Member Functions

- CPPUNIT_TEST (threaded_functionality_hundred_threads)
- CPPUNIT_TEST (threaded_functionality_thousand_threads)
- CPPUNIT_TEST (threaded_functionality_hundred_threads_six_account)
- CPPUNIT_TEST (threaded_functionality_thousand_threads_six_account)
- CPPUNIT_TEST (nested_hundred_thread_functionality)
- CPPUNIT_TEST (nested_thousand_thread_functionality)
- CPPUNIT_TEST (complex_threaded_functionality_hundred_threads)
- CPPUNIT_TEST (complex_threaded_functionality_ten_threads)
- CPPUNIT_TEST (two_object_transfer_complete)
- CPPUNIT_TEST (two_object_transfer_state_change)
- CPPUNIT_TEST (nested_transaction_object_test)
- CPPUNIT_TEST (multi_threaded_multiple_object_exchange_test)
- CPPUNIT_TEST (multi_threaded_single_object_test_with_ten_threads)
- CPPUNIT_TEST (single_threaded_multiple_object_test)
- CPPUNIT_TEST (multi_threaded_multiple_objects_test)

- CPPUNIT_TEST (increase_nesting)
- CPPUNIT_TEST (increase_nesting_fail)
- CPPUNIT_TEST (decrease_nesting)
- CPPUNIT_TEST (decrease_nesting_fail)
- CPPUNIT_TEST (two_object_transfer_state_change)
- CPPUNIT_TEST (compare_Transaction_Manager_singleton_instance)
- CPPUNIT_TEST (TM_get_thread_map)
- CPPUNIT_TEST_EXCEPTION (register_null_pointer_throw_runtime_error, std::runtime_error)
- CPPUNIT_TEST_EXCEPTION (object_not_registered_throw_runtime_error, std::runtime_error)
- CPPUNIT_TEST_EXCEPTION (store_null_pointer_throw_runtime_error, std::runtime_error)
- CPPUNIT_TEST_SUITE (MyTestCAse)
- CPPUNIT_TEST_SUITE_END ()

Private Attributes

- client * a
- client * b
- client * c

4.6.1 Detailed Description

Definition at line 18 of file MyTestCAse.h.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 MyTestCAse::MyTestCAse ( ) [inline]

Definition at line 61 of file MyTestCAse.h.

```
00061 {};
```

4.6.2.2 MyTestCAse::MyTestCAse (const MyTestCAse & *orig*)

Definition at line 9 of file MyTestCAse.cpp.

```
00009                                                   {
00010 }
```

4.6.2.3 MyTestCAse::~MyTestCAse ( ) [virtual]

Definition at line 12 of file MyTestCAse.cpp.

```
00012                                   {
00013 }
```

4.6.3 Member Function Documentation

4.6.3.1 void MyTestCAse::_collection_bject_ (std::vector< std::shared_ptr< OSTM >> *_customer_vec,* TM & *_tm,* double *_amount,* int *loop*)

Test with a vector collection.

Parameters

_customer_vec	collection of OSTM type objects
_tm	TRansaction Manager
_amount	value used in the transaction

Declare required pointers

Register customers accounts from the collection (vector)

From std::shared_ptr<OSTM> to std::shared_ptr<BANK> to access the virtual methods

Make changes with the objects

From std::shared_ptr<BANK> to std::shared_ptr<OSTM> to store the memory spaces

Store changes

Commit changes

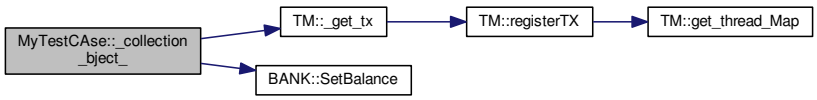
Definition at line 20 of file MyTestCAse.cpp.

References TM::_get_tx(), and BANK::SetBalance().

Referenced by multi_threaded_multiple_objects_test(), and single_threaded_multiple_object_test().

```
00020
            {
00021
     std::shared_ptr<TX> tx = _tm._get_tx();
00023
    std::shared_ptr<OSTM> _TO_OSTM_;
00028    std::shared_ptr<BANK>  _TO_;
00029
00030    bool done = false;
00031    try {
00032        while (!done) {
00033            for (int i = 0; i < loop; ++i) {
00034                for (auto&& obj : _customer_vec) {
00038                    // auto&& obj = _customer_vec.at(i);
00039                    tx->_register(obj);
00043                    _TO_ = std::dynamic_pointer_cast<BANK> (tx->load(obj));
00047                    _TO_->SetBalance(_TO_->GetBalance() + (_amount));
00051                    _TO_OSTM_ = std::dynamic_pointer_cast<OSTM> (_TO_);
00055                    tx->store(_TO_OSTM_);
00056                }
00057            }
00061            done = tx->commit();
00062        }
00063    } catch (std::runtime_error& e) {
00064        std::cout << e.what() << std::endl;
00065    }
00066 }
```

Here is the call graph for this function:



4.6.3.2 void MyTestCAse::_complex_transfer_ (std::shared_ptr< OSTM > *_from_*, std::shared_ptr< OSTM > *_from_two_*, std::vector< std::shared_ptr< OSTM >> *_customer_vec*, TM & *_tm*, double *_amount*)

This function use two single objects and a collection of objects in the transaction. The two single object transfer 1 - 1 (2) unit to every object in the collection.

Parameters

from	pointer used in transaction
from_two	pointer used in transaction
_customer_vec	collection of pointer
_tm	TRansaction Manager
_amount	value used in the transaction

Register the two single account

Declare required pointers

Register customers accounts from the collection (vector)

From std::shared_ptr<OSTM> to std::shared_ptr<BANK> to access the virtual methods

Make changes with the objects

From std::shared_ptr<BANK> to std::shared_ptr<OSTM> to store the memory spaces

Store changes

Commit changes

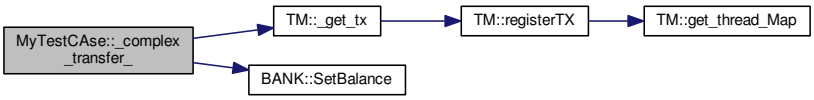
Definition at line 126 of file MyTestCAse.cpp.

References TM::_get_tx(), and BANK::SetBalance().

Referenced by complex_threaded_functionality_hundred_threads(), and complex_threaded_functionality_ten_threads().

```
00126                                                                   {
00127     std::shared_ptr<TX> tx = _tm._get_tx();
00131     tx->_register(_from_);
00132     tx->_register(_from_two_);
00136     std::shared_ptr<OSTM> _FROM_OSTM_ONE_, _FROM_OSTM_TWO_, _TO_OSTM_;
00137     std::shared_ptr<BANK> _FROM_, _FROM_TWO_, _TO_;
00138
00139     bool done = false;
00140     try {
00141         while (!done) {
00142             // for (int i = 0; i < vector_number; ++i) {
00143             for (auto&& obj : _customer_vec) {
00147                 // auto&& obj = _customer_vec.at(i);
00148                 tx->_register(obj);
00152                 _FROM_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_));
00153                 _FROM_TWO_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_two_));
00154                 _TO_ = std::dynamic_pointer_cast<BANK> (tx->load(obj));
00158                 _FROM_->SetBalance(_FROM_->GetBalance() - _amount);
00159                 _FROM_TWO_->SetBalance(_FROM_TWO_->GetBalance() - _amount);
00160                 _TO_->SetBalance(_TO_->GetBalance() + (_amount * 2));
00164                 _FROM_OSTM_ONE_ = std::dynamic_pointer_cast<OSTM> (_FROM_);
00165                 _FROM_OSTM_TWO_ = std::dynamic_pointer_cast<OSTM> (_FROM_TWO_);
00166                 _TO_OSTM_ = std::dynamic_pointer_cast<OSTM> (_TO_);
00170                 tx->store(_FROM_OSTM_ONE_);
00171                 tx->store(_FROM_OSTM_TWO_);
00172                 tx->store(_TO_OSTM_);
00173             }
00177             done = tx->commit();
00178         }
00179     } catch (std::runtime_error& e) {
00180         std::cout << e.what() << std::endl;
00181     }
00182 }
```

Here is the call graph for this function:



4.6.3.3 void MyTestCAse::_nesting_ (std::shared_ptr< OSTM > *_to_*, std::shared_ptr< OSTM > *_from_*, TM & *_tm*, double *_amount*)

Testing nested transaction.

Parameters

to	pointer used in transaction
from	pointer used in transaction
_tm	Transaction Manager
_amount	value used in the transaction

Register the two single account

Declare required pointers

From std::shared_ptr<OSTM> to std::shared_ptr<BANK> to access the virtual methods

Make changes with the objects

From std::shared_ptr<BANK> to std::shared_ptr<OSTM> to store the memory spaces

Store changes

NESTED TRANSACTION

Make changes with the objects

From std::shared_ptr<BANK> to std::shared_ptr<OSTM> to store the memory spaces

Store changes

NESTED TRANSACTION IN THE NESTED TRANSACTION *two_account_transfer* function call

Commit changes

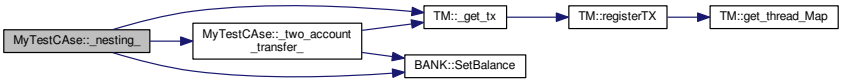
Definition at line 320 of file MyTestCAse.cpp.

References TM::_get_tx(), _two_account_transfer_(), and BANK::SetBalance().

Referenced by nested_hundred_thread_functionality(), and nested_thousand_thread_functionality().

```
{
    std::shared_ptr<TX> tx = _tm._get_tx();
    tx->_register(_to_);
    tx->_register(_from_);
    std::shared_ptr<BANK> _TO_BANK_, _FROM_BANK_;
    std::shared_ptr<OSTM> _TO_OSTM_, _FROM_OSTM_;


    bool done = false;
    try {
        while (!done) {
            _TO_BANK_ = std::dynamic_pointer_cast<BANK> (tx->load(_to_));
            _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_));
            _TO_BANK_->SetBalance(_TO_BANK_->GetBalance() + _amount);
            _FROM_BANK_->SetBalance(_FROM_BANK_->GetBalance() - _amount);
            _TO_OSTM_ = std::dynamic_pointer_cast<OSTM> (_TO_BANK_);
            _FROM_OSTM_ = std::dynamic_pointer_cast<OSTM> (_FROM_BANK_);
            tx->store(_TO_OSTM_);
            tx->store(_FROM_OSTM_);

            std::shared_ptr<TX> txTwo = _tm._get_tx();

            bool nestedDone = false;
            while (!nestedDone) {
                _TO_BANK_ = std::dynamic_pointer_cast<BANK> (txTwo->load(_to_));
                _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (txTwo->load(_from_));
                _TO_BANK_->SetBalance(_TO_BANK_->GetBalance() + _amount);
                _FROM_BANK_->SetBalance(_FROM_BANK_->GetBalance() - _amount);
                _TO_OSTM_ = std::dynamic_pointer_cast<OSTM> (_TO_BANK_);
                _FROM_OSTM_ = std::dynamic_pointer_cast<OSTM> (_FROM_BANK_);
                txTwo->store(_TO_OSTM_);
                txTwo->store(_FROM_OSTM_);
                _two_account_transfer_(_to_, _from_, _tm, _amount);

                nestedDone = txTwo->commit();
            }

            done = tx->commit();
        }
    } catch (std::runtime_error& e) {
        std::cout << e.what() << std::endl;
    }
}
```

Here is the call graph for this function:



4.6.3.4 void MyTestCAse::_one_account_transfer_ (std::shared_ptr< OSTM > *_to_*, TM & *_tm*, double *_amount*)

one object in the trasaction

Parameters

to	pointer used in transaction
_tm	TRansaction Manager
_amount	value used in the transaction

Register the two single account

From std::shared_ptr<OSTM> to std::shared_ptr<BANK> to access the virtual methods

Make changes with the objects

From std::shared_ptr<BANK> to std::shared_ptr<OSTM> to store the memory spaces

Store changes

Commit changes

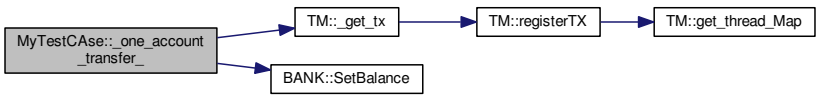
Definition at line 74 of file MyTestCAse.cpp.

References TM::_get_tx(), BANK::SetBalance(), and tm.

Referenced by multi_threaded_single_object_test_with_ten_threads().

```
00074                                                                                                    {
00075      std::shared_ptr<TX> tx = tm._get_tx();
00079     tx->_register(_to_);
00080
00081     std::shared_ptr<OSTM> _TO_OSTM_;
00082     std::shared_ptr<BANK> _TO_;
00083
00084     try {
00085         bool done = false;
00086         while (!done) {
00090             _TO_ = std::dynamic_pointer_cast<BANK> (tx->load(_to_));
00094             _TO_->SetBalance(_TO_->GetBalance() + (_amount ));
00095
00099             _TO_OSTM_ = std::dynamic_pointer_cast<OSTM> (_TO_);
00103             tx->store(_TO_OSTM_);
00104
00108             done = tx->commit();
00109         }
00110     } catch (std::runtime_error& e) {
00111         std::cout << e.what() << std::endl;
00112     }
00113
00114
00115
00116 }
```

Here is the call graph for this function:



4.6.3.5 void MyTestCAse::_six_account_transfer_ (std::shared_ptr< OSTM > *_to_*, std::shared_ptr< OSTM > *_from_one_*, std::shared_ptr< OSTM > *_from_two_*, std::shared_ptr< OSTM > *_from_three_*, std::shared_ptr< OSTM > *_from_four_*, std::shared_ptr< OSTM > *_from_five_*, TM & *_tm*, double *_amount*)

Testing the transactions between six object.

Parameters

to	pointer used in transaction
from_one	pointer used in transaction
from_two	pointer used in transaction
from_three	pointer used in transaction
from_four	pointer used in transaction
from_five	pointer used in transaction
_tm	Transaction Manager
_amount	value used in the transaction

Register the two single account

Required pointers to use in transaction

From std::shared_ptr<OSTM> to std::shared_ptr<BANK> to access the virtual methods

Make changes with the objects

From std::shared_ptr<BANK> to std::shared_ptr<OSTM> to store the memory spaces

Store changes

Commit changes

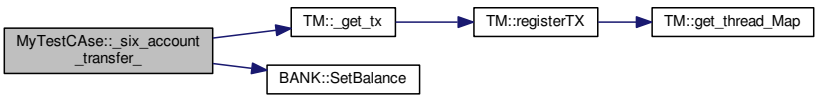
Definition at line 194 of file MyTestCAse.cpp.

References TM::_get_tx(), and BANK::SetBalance().

Referenced by threaded_functionality_hundred_threads_six_account(), and threaded_functionality_thousand_threads_six_account().

```
00194
                                                {
00195     std::shared_ptr<TX> tx = _tm._get_tx();
00199     tx->_register(_to_);
00200     tx->_register(_from_one_);
00201     tx->_register(_from_two_);
00202     tx->_register(_from_three_);
00203     tx->_register(_from_four_);
00204     tx->_register(_from_five_);
00205
00209     std::shared_ptr<OSTM> _TO_OSTM, _FROM_ONE_OSTM, _FROM_TWO_OSTM, _FROM_THREE_OSTM, _FROM_FOUR_OSTM,
      _FROM_FIVE_OSTM;
00210     std::shared_ptr<BANK> _TO_, _FROM_ONE_, _FROM_TWO_, _FROM_THREE_, _FROM_FOUR_, _FROM_FIVE_;
00211     try {
00212         bool done = false;
00213         while (!done) {
00217             _TO_ = std::dynamic_pointer_cast<BANK> (tx->load(_to_));
00218             _FROM_ONE_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_one_));
00219             _FROM_TWO_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_two_));
00220             _FROM_THREE_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_three_));
00221             _FROM_FOUR_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_four_));
00222             _FROM_FIVE_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_five_));
00226             _TO_->SetBalance(_TO_->GetBalance() + (_amount * 5));
00227             _FROM_ONE_->SetBalance(_FROM_ONE_->GetBalance() - _amount);
00228             _FROM_TWO_->SetBalance(_FROM_TWO_->GetBalance() - _amount);
00229             _FROM_THREE_->SetBalance(_FROM_THREE_->GetBalance() - _amount);
00230             _FROM_FOUR_->SetBalance(_FROM_FOUR_->GetBalance() - _amount);
00231             _FROM_FIVE_->SetBalance(_FROM_FIVE_->GetBalance() - _amount);
00235             _TO_OSTM = std::dynamic_pointer_cast<OSTM> (_TO_);
00236             _FROM_ONE_OSTM = std::dynamic_pointer_cast<OSTM> (_FROM_ONE_);
00237             _FROM_TWO_OSTM = std::dynamic_pointer_cast<OSTM> (_FROM_TWO_);
00238             _FROM_THREE_OSTM = std::dynamic_pointer_cast<OSTM> (_FROM_THREE_);
00239             _FROM_FOUR_OSTM = std::dynamic_pointer_cast<OSTM> (_FROM_FOUR_);
00240             _FROM_FIVE_OSTM = std::dynamic_pointer_cast<OSTM> (_FROM_FIVE_);
00244             tx->store(_TO_OSTM);
00245             tx->store(_FROM_ONE_OSTM);
00246             tx->store(_FROM_TWO_OSTM);
00247             tx->store(_FROM_THREE_OSTM);
00248             tx->store(_FROM_FOUR_OSTM);
00249             tx->store(_FROM_FIVE_OSTM);
00253             done = tx->commit();
00254         }
00255     } catch (std::runtime_error& e) {
00256         std::cout << e.what() << std::endl;
00257     }
00258 }
```

Here is the call graph for this function:



4.6.3.6 void MyTestCAse::_two_account_transfer_ (std::shared_ptr< OSTM > *_to_*, std::shared_ptr< OSTM > *_from_*, TM & *tm*, double *_amount*)

Testin transaction between two pointer.

Parameters

to	pointer used in transaction
from	pointer used in transaction
tm	TRansaction Manager
_amount	value used in the transaction

Register the two single account

Declare required pointers

From std::shared_ptr<OSTM> to std::shared_ptr<BANK> to access the virtual methods

Make changes with the objects

From std::shared_ptr<BANK> to std::shared_ptr<OSTM> to store the memory spaces

Store changes

Commit changes

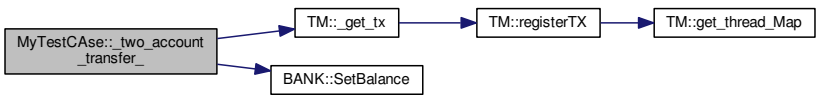
Definition at line 266 of file MyTestCAse.cpp.

References TM::_get_tx(), and BANK::SetBalance().

Referenced by _nesting_(), multi_threaded_multiple_object_exchange_test(), threaded_functionality_hundred_threads(), and threaded_functionality_thousand_threads().

```
00266
                {
00267
00268     std::shared_ptr<TX> tx = tm._get_tx();
00272     tx->_register(_to_);
00273     tx->_register(_from_);
00277     std::shared_ptr<BANK> _TO_BANK_, _FROM_BANK_;
00278     std::shared_ptr<OSTM> _TO_OSTM_, _FROM_OSTM_;
00279
00280     bool done = false;
00281     try {
00282         while (!done) {
00286             _TO_BANK_ = std::dynamic_pointer_cast<BANK> (tx->load(_to_));
00287             _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_));
00291             _TO_BANK_->SetBalance(_TO_BANK_->GetBalance() + _amount);
00292             _FROM_BANK_->SetBalance(_FROM_BANK_->GetBalance() - _amount);
00296             _TO_OSTM_ = std::dynamic_pointer_cast<OSTM> (_TO_BANK_);
00297             _FROM_OSTM_ = std::dynamic_pointer_cast<OSTM> (_FROM_BANK_);
00301             tx->store(_TO_OSTM_);
00302             tx->store(_FROM_OSTM_);
00303
00307             done = tx->commit();
00308         }
00309     } catch (std::runtime_error& e) {
00310         std::cout << e.what() << std::endl;
00311     }
00312 }
```

Here is the call graph for this function:



4.6.3.7 void MyTestCAse::compare_Transaction_Manager_singleton_instance ( )

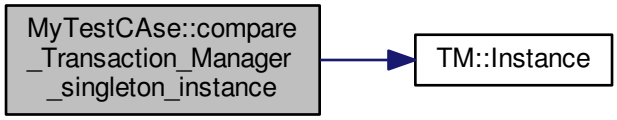
This testing function comparing the Transaction Manager to make sure the application using a Singleton object.

Definition at line 896 of file MyTestCAse.cpp.

References TM::Instance(), and tm.

```
00896                                                                   {
00897     TM& tm_copy = TM::Instance();
00898     CPPUNIT_ASSERT( tm == tm_copy );
00899 }
```

Here is the call graph for this function:



4.6.3.8 void MyTestCAse::complex_threaded_functionality_hundred_threads ( )

\ brief Testing the library consistent behavior LONG RUNNING PROCESS !!!! 1.5 - 2 Minutes !!! This test transfer 1 - 1 unit by 100 threads = 100 ∗ 1 = 100 to every object in the collection (-6000) by single objects After transfer from account has -100 ∗ 600 for both single objects and + 2 unit ∗ 100 to every objects

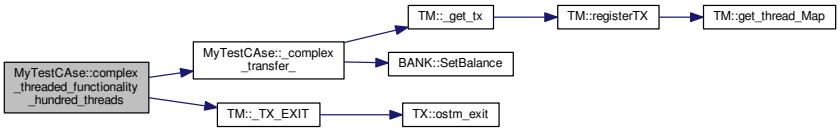
Definition at line 412 of file MyTestCAse.cpp.

References _complex_transfer_(), TM::_TX_EXIT(), aib_ptr, boi_ptr, and tm.

```
00412                                                                   {
00413     tm._TX_EXIT();
00414      std::vector<std::shared_ptr < OSTM>>_customer_vec;
00415      for (int i = 0; i < 600; ++i) {
00416         if (i % 6 == 0) {
00417             std::shared_ptr<OSTM> sharedptr(new AIB(i, 50, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00418             _customer_vec.push_back(std::move(sharedptr));
00419         } else if (i % 5 == 0) {
00420             std::shared_ptr<OSTM> sharedptr(new BOI(i, 50, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00421             _customer_vec.push_back(std::move(sharedptr));
00422         } else if (i % 4 == 0) {
00423             std::shared_ptr<OSTM> sharedptr(new BOA(i, 50, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00424             _customer_vec.push_back(std::move(sharedptr));
00425         } else if (i % 3 == 0) {
00426             std::shared_ptr<OSTM> sharedptr(new SWBPLC(i, 50, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00427             _customer_vec.push_back(std::move(sharedptr));
00428         } else if (i % 2 == 0) {
00429             std::shared_ptr<OSTM> sharedptr(new ULSTER(i, 50, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00430             _customer_vec.push_back(std::move(sharedptr));
00431         } else if (i % 1 == 0) {
00432             std::shared_ptr<OSTM> sharedptr(new UNBL(i, 50, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00433             _customer_vec.push_back(std::move(sharedptr));
```

```
        }
    }
    std::shared_ptr<BANK> _FROM_BANK_;
    std::shared_ptr<BANK> _TO_BANK_;

    std::shared_ptr<OSTM> aib_ptr(new AIB(100, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> boi_ptr(new BOI(200, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));

    int transferAmount = 1;
    int threadArraySize = 100;
    std::thread thArray[threadArraySize];

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i] = std::thread(&MyTestCAse::_complex_transfer_, this, aib_ptr, boi_ptr, std::ref(_customer_vec),  std::ref(tm), transferAmount);
    }

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i].join();
    }

    _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (boi_ptr);
    _TO_BANK_ = std::dynamic_pointer_cast<BANK> (aib_ptr);

    CPPUNIT_ASSERT(_FROM_BANK_->GetBalance() == -59500);
    CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == -59500);

}
```

Here is the call graph for this function:



4.6.3.9 void MyTestCAse::complex_threaded_functionality_ten_threads ( )

Testing the library consistent behavior This test transfer 1 - 1 unit by 10 threads = 10 ∗ 1 = 10 to evey object in the collection (-600) by single objects After transfer from account has -10 ∗ 600 for both single objects and + 2 unit ∗ 10 to every objects.

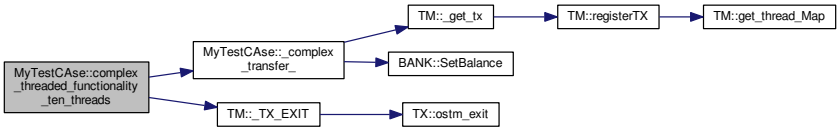
Definition at line 467 of file MyTestCAse.cpp.

References _complex_transfer_(), TM::_TX_EXIT(), aib_ptr, boi_ptr, and tm.

```
                                                    {
    tm._TX_EXIT();
    std::vector<std::shared_ptr < OSTM>>_customer_vec;
     for (int i = 0; i < 600; ++i) {
        if (i % 6 == 0) {
            std::shared_ptr<OSTM> sharedptr(new AIB(i, 50, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
            _customer_vec.push_back(std::move(sharedptr));
        } else if (i % 5 == 0) {
            std::shared_ptr<OSTM> sharedptr(new BOI(i, 50, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
            _customer_vec.push_back(std::move(sharedptr));
        } else if (i % 4 == 0) {
            std::shared_ptr<OSTM> sharedptr(new BOA(i, 50, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
            _customer_vec.push_back(std::move(sharedptr));
        } else if (i % 3 == 0) {
            std::shared_ptr<OSTM> sharedptr(new SWBPLC(i, 50, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
            _customer_vec.push_back(std::move(sharedptr));
```

```
        } else if (i % 2 == 0) {
            std::shared_ptr<OSTM> sharedptr(new ULSTER(i, 50, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
            _customer_vec.push_back(std::move(sharedptr));
        } else if (i % 1 == 0) {
            std::shared_ptr<OSTM> sharedptr(new UNBL(i, 50, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
            _customer_vec.push_back(std::move(sharedptr));
        }
    }
    std::shared_ptr<BANK> _FROM_BANK_;
    std::shared_ptr<BANK> _TO_BANK_;

    std::shared_ptr<OSTM> aib_ptr(new AIB(100, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> boi_ptr(new BOI(200, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));

    int transferAmount = 1;
    int threadArraySize = 10;
    std::thread thArray[threadArraySize];

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i] = std::thread(&MyTestCAse::_complex_transfer_, this, aib_ptr, boi_ptr, std::ref(_customer_vec),  std::ref(tm), transferAmount);
    }

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i].join();
    }

    _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (boi_ptr);
    _TO_BANK_ = std::dynamic_pointer_cast<BANK> (aib_ptr);

    CPPUNIT_ASSERT(_FROM_BANK_->GetBalance() == -5500);
    CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == -5500);

}
```

Here is the call graph for this function:



4.6.3.10 MyTestCAse::CPPUNIT_TEST (threaded_functionality_hundred_threads) [private]

4.6.3.11 MyTestCAse::CPPUNIT_TEST (threaded_functionality_thousand_threads) [private]

4.6.3.12 MyTestCAse::CPPUNIT_TEST (threaded_functionality_hundred_threads_six_account) [private]

4.6.3.13 MyTestCAse::CPPUNIT_TEST (threaded_functionality_thousand_threads_six_account) [private]

4.6.3.14 MyTestCAse::CPPUNIT_TEST (nested_hundred_thread_functionality) [private]

4.6.3.15 MyTestCAse::CPPUNIT_TEST (nested_thousand_thread_functionality) [private]

4.6.3.16 MyTestCAse::CPPUNIT_TEST (complex_threaded_functionality_hundred_threads) [private]

4.6.3.17 MyTestCAse::CPPUNIT_TEST (complex_threaded_functionality_ten_threads) [private]

4.6.3.18 MyTestCAse::CPPUNIT_TEST (two_object_transfer_complete) [private]

4.6.3.19 MyTestCAse::CPPUNIT_TEST (two_object_transfer_state_change) [private]

4.6.3.20 MyTestCAse::CPPUNIT_TEST (nested_transaction_object_test) [private]

4.6.3.21 MyTestCAse::CPPUNIT_TEST (multi_threaded_multiple_object_exchange_test) [private]

4.6.3.22 MyTestCAse::CPPUNIT_TEST (multi_threaded_single_object_test_with_ten_threads) [private]

4.6.3.23 MyTestCAse::CPPUNIT_TEST (single_threaded_multiple_object_test) [private]

4.6.3.24 MyTestCAse::CPPUNIT_TEST (multi_threaded_multiple_objects_test) [private]

4.6.3.25 MyTestCAse::CPPUNIT_TEST (increase_nesting) [private]

4.6.3.26 MyTestCAse::CPPUNIT_TEST (increase_nesting_fail) [private]

4.6.3.27 MyTestCAse::CPPUNIT_TEST (decrease_nesting) [private]

4.6.3.28 MyTestCAse::CPPUNIT_TEST (decrease_nesting_fail) [private]

4.6.3.29 MyTestCAse::CPPUNIT_TEST (two_object_transfer_state_change) [private]

4.6.3.30 MyTestCAse::CPPUNIT_TEST (compare_Transaction_Manager_singleton_instance) [private]

4.6.3.31 MyTestCAse::CPPUNIT_TEST (TM_get_thread_map) [private]

4.6.3.32 MyTestCAse::CPPUNIT_TEST_EXCEPTION (register_null_pointer_throw_runtime_error , std::runtime_error) [private]

4.6.3.33 MyTestCAse::CPPUNIT_TEST_EXCEPTION (object_not_registered_throw_runtime_error , std::runtime_error) [private]

4.6.3.34 MyTestCAse::CPPUNIT_TEST_EXCEPTION (store_null_pointer_throw_runtime_error , std::runtime_error) [private]

4.6.3.35 MyTestCAse::CPPUNIT_TEST_SUITE (MyTestCAse) [private]

4.6.3.36 MyTestCAse::CPPUNIT_TEST_SUITE_END ( ) [private]

4.6.3.37 void MyTestCAse::decrease_nesting ( )

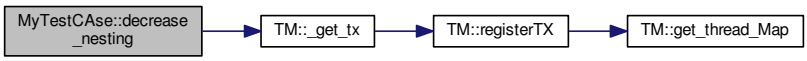
Testing the nesting decrease function.

Definition at line 853 of file MyTestCAse.cpp.

References TM::_get_tx(), and tm.

```
{

    std::shared_ptr<TX> tx = tm._get_tx();
    tx->setTx_nesting_level(10);
    tx->_decrease_tx_nesting();
    tx->_decrease_tx_nesting();
    tx->_decrease_tx_nesting();

    CPPUNIT_ASSERT( tx->getTx_nesting_level() == 7);
    tx->_decrease_tx_nesting();
    CPPUNIT_ASSERT( tx->getTx_nesting_level() == 6);
}
```

Here is the call graph for this function:



4.6.3.38 void MyTestCAse::decrease_nesting_fail ( )

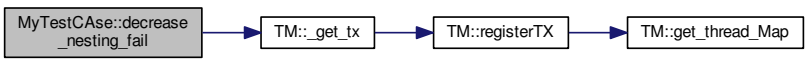
Testing the nesting decrease function to make sure the variable state changing.

Definition at line 881 of file MyTestCAse.cpp.

References TM::_get_tx(), and tm.

```
{

    std::shared_ptr<TX> tx = tm._get_tx();
    tx->setTx_nesting_level(10);
    tx->_decrease_tx_nesting();
    tx->_decrease_tx_nesting();
    tx->_decrease_tx_nesting();

    CPPUNIT_ASSERT( !(tx->getTx_nesting_level() == 10));
    tx->_decrease_tx_nesting();
    CPPUNIT_ASSERT( !(tx->getTx_nesting_level() == 12));
}
```

Here is the call graph for this function:



4.6.3.39 void MyTestCAse::increase_nesting ( )

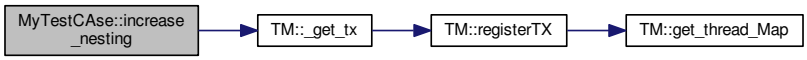
Testing the nesting increase function.

Definition at line 841 of file MyTestCAse.cpp.

References TM::_get_tx(), and tm.

```
00841                                   {
00842     std::shared_ptr<TX> tx = tm._get_tx();
00843     tx->setTx_nesting_level(0);
00844     tx->_increase_tx_nesting();
00845     tx->_increase_tx_nesting();
00846     tx->_increase_tx_nesting();
00847
00848     CPPUNIT_ASSERT( tx->getTx_nesting_level() == 3);
00849 }
```

Here is the call graph for this function:



4.6.3.40 void MyTestCAse::increase_nesting_fail ( )

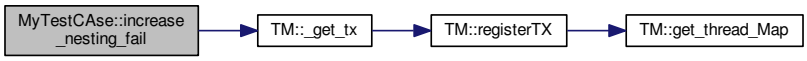
Testing the nesting increase function to make sure the variable state changing.

Definition at line 868 of file MyTestCAse.cpp.

References TM::_get_tx(), and tm.

```
00868                                       {
00869     std::shared_ptr<TX> tx = tm._get_tx();
00870     tx->setTx_nesting_level(0);
00871     tx->_increase_tx_nesting();
00872     tx->_increase_tx_nesting();
00873     tx->_increase_tx_nesting();
00874
00875     CPPUNIT_ASSERT( !(tx->getTx_nesting_level() == 0));
00876
00877 }
```

Here is the call graph for this function:



4.6.3.41 void MyTestCAse::multi_threaded_multiple_object_exchange_test ( )

Design Manual document based tests Implementations.

1. Single thread multiple transactional object exchange test
2. Multi-threaded single Object test with 10 threads.
3. Single-threaded multiple object test.
4. Multi-threaded multiple Objects test.

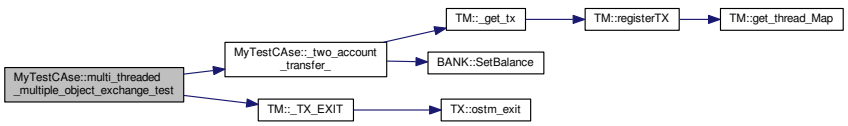
1. Single thread multiple transactional object exchange test

Definition at line 922 of file MyTestCAse.cpp.

References _two_account_transfer_(), TM::_TX_EXIT(), aib_ptr, boi_ptr, and tm.

```
00922                                                                {
00923     //ten threads using same objects
00924     tm._TX_EXIT();
00925     std::shared_ptr<BANK> _FROM_BANK_;
00926     std::shared_ptr<BANK> _TO_BANK_;
00927
00928     std::shared_ptr<OSTM> aib_ptr(new AIB(100, 100, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00929     std::shared_ptr<OSTM> boi_ptr(new BOI(200, 100, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00930
00931     int transferAmount = 5;
00932     int threadArraySize = 10;
00933     std::thread thArray[threadArraySize];
00934
00935     for (int i = 0; i < threadArraySize; ++i) {
00936         thArray[i] = std::thread(&MyTestCAse::_two_account_transfer_,
      this, aib_ptr, boi_ptr, std::ref(tm), transferAmount);
00937     }
00938
00939     for (int i = 0; i < threadArraySize; ++i) {
00940         thArray[i].join();
00941     }
00942
00943     _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (boi_ptr);
00944     _TO_BANK_ = std::dynamic_pointer_cast<BANK> (aib_ptr);
00945
00946     CPPUNIT_ASSERT(_FROM_BANK_->GetBalance() == 50);
00947     CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == 150);
00948
00949 }
```

Here is the call graph for this function:



4.6.3.42 void MyTestCAse::multi_threaded_multiple_objects_test ( )

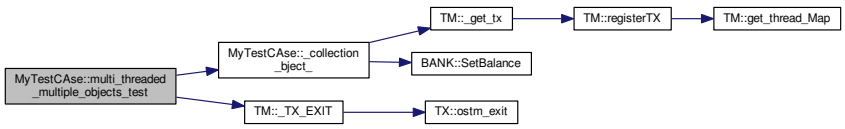
1. Multi-threaded multiple Objects test.

Definition at line 1040 of file MyTestCAse.cpp.

References _collection_bject_(), TM::_TX_EXIT(), and tm.

```
01040                                                   {
01041     //three threads all thread use different objects
01042
01043     tm._TX_EXIT();
01044     std::vector<std::shared_ptr < OSTM>>_customer_vec;
01045      for (int i = 0; i < 10; ++i) {
01046         if (i % 6 == 0) {
01047             std::shared_ptr<OSTM> sharedptr(new AIB(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
01048             _customer_vec.push_back(std::move(sharedptr));
01049         } else if (i % 5 == 0) {
01050             std::shared_ptr<OSTM> sharedptr(new BOI(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
01051             _customer_vec.push_back(std::move(sharedptr));
01052         } else if (i % 4 == 0) {
01053             std::shared_ptr<OSTM> sharedptr(new BOA(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
01054             _customer_vec.push_back(std::move(sharedptr));
01055         } else if (i % 3 == 0) {
01056             std::shared_ptr<OSTM> sharedptr(new SWBPLC(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
01057             _customer_vec.push_back(std::move(sharedptr));
01058         } else if (i % 2 == 0) {
01059             std::shared_ptr<OSTM> sharedptr(new ULSTER(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
01060             _customer_vec.push_back(std::move(sharedptr));
01061         } else if (i % 1 == 0) {
01062             std::shared_ptr<OSTM> sharedptr(new UNBL(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
01063             _customer_vec.push_back(std::move(sharedptr));
01064         }
01065     }
01066     std::shared_ptr<BANK> _one_, _two_, _three_, _four_, _five_, _six_;
01067     int loop = 1;
01068     int transferAmount = 1;
01069     int threadArraySize = 10;
01070     std::thread thArray[threadArraySize];
01071
01072     for (int i = 0; i < threadArraySize; ++i) {
01073         thArray[i] = std::thread(&MyTestCAse::_collection_bject_, this,
      std::ref(_customer_vec),  std::ref(tm), transferAmount , loop);
01074     }
01075
01076     for (int i = 0; i < threadArraySize; ++i) {
01077         thArray[i].join();
01078     }
01079
01080     _one_ = std::dynamic_pointer_cast<BANK> (_customer_vec[0]);
01081     _two_ = std::dynamic_pointer_cast<BANK> (_customer_vec[1]);
01082     _three_ = std::dynamic_pointer_cast<BANK> (_customer_vec[2]);
01083     _four_ = std::dynamic_pointer_cast<BANK> (_customer_vec[3]);
01084     _five_ = std::dynamic_pointer_cast<BANK> (_customer_vec[4]);
01085     _six_ = std::dynamic_pointer_cast<BANK> (_customer_vec[5]);
01086
01087     CPPUNIT_ASSERT(_one_->GetBalance() == 20);
01088     CPPUNIT_ASSERT(_two_->GetBalance() == 20);
01089     CPPUNIT_ASSERT(_three_->GetBalance() == 20);
01090     CPPUNIT_ASSERT(_four_->GetBalance() == 20);
01091     CPPUNIT_ASSERT(_five_->GetBalance() == 20);
01092     CPPUNIT_ASSERT(_six_->GetBalance() == 20);
01093 }
```

Here is the call graph for this function:



4.6.3.43 void MyTestCAse::multi_threaded_single_object_test_with_ten_threads ( )

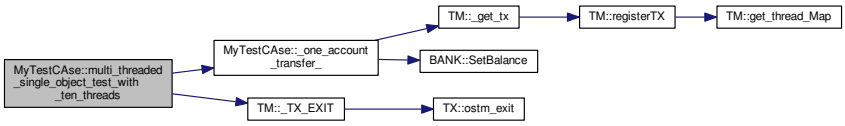
1. Multi-threaded single Object test with 10 threads.

Definition at line 954 of file MyTestCAse.cpp.

References _one_account_transfer_(), TM::_TX_EXIT(), aib_ptr, and tm.

```
00954                                                                   {
00955     //one object ten threads
00956     tm._TX_EXIT();
00957     std::shared_ptr<BANK> _TO_BANK_;
00958
00959     std::shared_ptr<OSTM> aib_ptr(new AIB(100, 100, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00960
00961     int transferAmount = 1;
00962     int threadArraySize = 10;
00963     std::thread thArray[threadArraySize];
00964
00965     for (int i = 0; i < threadArraySize; ++i) {
00966         thArray[i] = std::thread(&MyTestCAse::_one_account_transfer_,
      this, aib_ptr, std::ref(tm), transferAmount);
00967     }
00968
00969     for (int i = 0; i < threadArraySize; ++i) {
00970         thArray[i].join();
00971     }
00972
00973
00974     _TO_BANK_ = std::dynamic_pointer_cast<BANK> (aib_ptr);
00975
00976     CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == 110);
00977
00978 }
```

Here is the call graph for this function:



4.6.3.44 void MyTestCAse::nested_hundred_thread_functionality ( )

Testing the library consistent behavior Nested threaded function : 3 level of nesting, every thread transfer 3 unit from one object to the another object so, at end of the 100 thransaction the from object transfer 100 $*$ 3 (300) to the another object 500 - 300 = 200 AND 500 + 300 = 800.

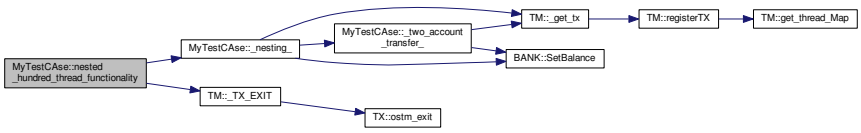
Definition at line 680 of file MyTestCAse.cpp.

References _nesting_(), TM::_TX_EXIT(), aib_ptr, boi_ptr, and tm.

```
00680                                                   {
00681     tm._TX_EXIT();
00682     std::shared_ptr<BANK> _FROM_BANK_;
00683     std::shared_ptr<BANK> _TO_BANK_;
00684
00685     std::shared_ptr<OSTM> aib_ptr(new AIB(100, 500, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00686     std::shared_ptr<OSTM> boi_ptr(new BOI(200, 500, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
```

```
    int transferAmount = 1;
    int threadArraySize = 100;
    std::thread thArray[threadArraySize];

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i] = std::thread(&MyTestCAse::_nesting_, this, aib_ptr, boi_ptr, std::ref(tm), transferAmount);
    }

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i].join();
    }

    _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (boi_ptr);
    _TO_BANK_ = std::dynamic_pointer_cast<BANK> (aib_ptr);

    CPPUNIT_ASSERT(_FROM_BANK_->GetBalance() == 200);
    CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == 800);

}
```

Here is the call graph for this function:



4.6.3.45 void MyTestCAse::nested_thousand_thread_functionality ( )

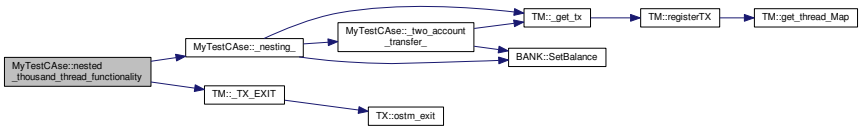
Testing the library consistent behavior Nested threaded function : 3 level of nesting, every thread transfer 3 unit from one object to the another object so, at end of the 100 thransaction the from object transfer 1000 $*$ 3 (3000) to the another object 500 - 3000 = -2500 AND 500 + 3000 = 3500.

Definition at line 715 of file MyTestCAse.cpp.

References _nesting_(), TM::_TX_EXIT(), aib_ptr, boi_ptr, and tm.

```
{
    tm._TX_EXIT();
    std::shared_ptr<BANK> _FROM_BANK_;
    std::shared_ptr<BANK> _TO_BANK_;

    std::shared_ptr<OSTM> aib_ptr(new AIB(100, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> boi_ptr(new BOI(200, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));

    int transferAmount = 1;
    int threadArraySize = 1000;
    std::thread thArray[threadArraySize];

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i] = std::thread(&MyTestCAse::_nesting_, this, aib_ptr, boi_ptr, std::ref(tm), transferAmount);
    }

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i].join();
    }

    _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (boi_ptr);
    _TO_BANK_ = std::dynamic_pointer_cast<BANK> (aib_ptr);

    CPPUNIT_ASSERT(_FROM_BANK_->GetBalance() == -2500);
    CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == 3500);

}
```

Here is the call graph for this function:



4.6.3.46 void MyTestCAse::nested_transaction_object_test ( )

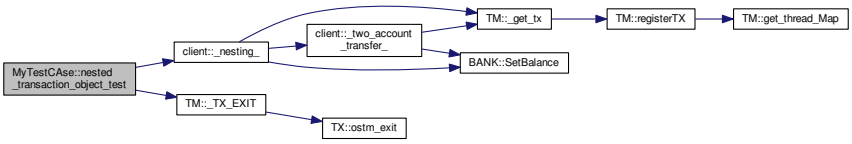
Testing the library consistent behavior This test calls the nested function, where every thread transfer 3 unit in the nested transactions. Because this test in not threaded, the 3 level deep nesting transfer 3∗20 = 60 from one object to the another object. 500 - 60 = 440 AND 500 + 60 = 560.

Definition at line 787 of file MyTestCAse.cpp.

References client::_nesting_(), TM::_TX_EXIT(), a, tm, ulster_ptr, and unbl_ptr.

```
00787                                                {
00788     tm._TX_EXIT();
00789     std::shared_ptr<OSTM> ulster_ptr(new ULSTER(500, 500, "Joe", "Blog", "High street,
       Kilkenny, Co.Kilkenny"));
00790     std::shared_ptr<OSTM> unbl_ptr(new UNBL(600, 500, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00791     std::shared_ptr<BANK> _FROM_BANK_;
00792     std::shared_ptr<BANK> _TO_BANK_;
00793     a->_nesting_(ulster_ptr, unbl_ptr, tm, 20);
00794     _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (unbl_ptr);
00795     _TO_BANK_ = std::dynamic_pointer_cast<BANK> (ulster_ptr);
00796
00797     CPPUNIT_ASSERT(_FROM_BANK_->GetBalance() == 440);
00798     CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == 560);
00799
00800 }
```

Here is the call graph for this function:



4.6.3.47 void MyTestCAse::object_not_registered_throw_runtime_error ( )

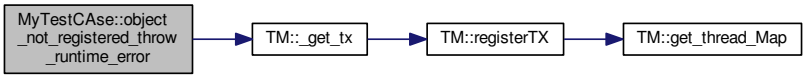
The library function throws runtime error if the client application tries to load a working pointer from the library of a not registered pointer by the client application. Runtime error should be thrown.

Definition at line 818 of file MyTestCAse.cpp.

References TM::_get_tx(), and tm.

```
{

    std::shared_ptr<OSTM> not_registered_object(new BOA(300, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<TX> tx = tm._get_tx();

    tx->load(not_registered_object);
}
```

Here is the call graph for this function:



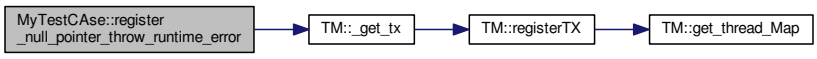
4.6.3.48 void MyTestCAse::register_null_pointer_throw_runtime_error ( )

Definition at line 805 of file MyTestCAse.cpp.

References TM::_get_tx(), and tm.

```
{

    std::shared_ptr<OSTM> null_ptr;
    std::shared_ptr<TX> tx = tm._get_tx();

    tx->_register(null_ptr);
}
```

Here is the call graph for this function:



4.6.3.49 void MyTestCAse::setUp ( ) [inline]

Definition at line 108 of file MyTestCAse.h.

```
{
    a = new client(1);
    b = new client(1);
    c = new client(1);

}
```

4.6.3.50 void MyTestCAse::single_threaded_multiple_object_test ( )

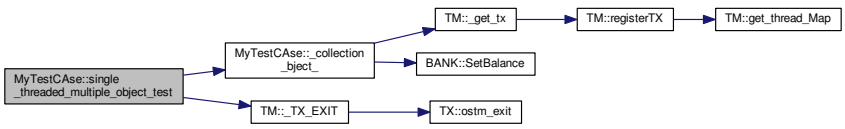
1. Single-threaded multiple object test.

Definition at line 983 of file MyTestCAse.cpp.

References _collection_bject_(), TM::_TX_EXIT(), and tm.

```
00983                                                   {
00984     //one transaction multiple objects six object function
00985     tm._TX_EXIT();
00986     std::vector<std::shared_ptr < OSTM>>_customer_vec;
00987      for (int i = 0; i < 10; ++i) {
00988         if (i % 6 == 0) {
00989             std::shared_ptr<OSTM> sharedptr(new AIB(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00990             _customer_vec.push_back(std::move(sharedptr));
00991         } else if (i % 5 == 0) {
00992             std::shared_ptr<OSTM> sharedptr(new BOI(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00993             _customer_vec.push_back(std::move(sharedptr));
00994         } else if (i % 4 == 0) {
00995             std::shared_ptr<OSTM> sharedptr(new BOA(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00996             _customer_vec.push_back(std::move(sharedptr));
00997         } else if (i % 3 == 0) {
00998             std::shared_ptr<OSTM> sharedptr(new SWBPLC(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00999             _customer_vec.push_back(std::move(sharedptr));
01000         } else if (i % 2 == 0) {
01001             std::shared_ptr<OSTM> sharedptr(new ULSTER(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
01002             _customer_vec.push_back(std::move(sharedptr));
01003         } else if (i % 1 == 0) {
01004             std::shared_ptr<OSTM> sharedptr(new UNBL(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
01005             _customer_vec.push_back(std::move(sharedptr));
01006         }
01007     }
01008     std::shared_ptr<BANK> _one_, _two_, _three_, _four_, _five_, _six_;
01009     int loop = 5;
01010     int transferAmount = 1;
01011     int threadArraySize = 1;
01012     std::thread thArray[threadArraySize];
01013
01014     for (int i = 0; i < threadArraySize; ++i) {
01015         thArray[i] = std::thread(&MyTestCAse::_collection_bject_, this,
      std::ref(_customer_vec),  std::ref(tm), transferAmount , loop);
01016     }
01017
01018     for (int i = 0; i < threadArraySize; ++i) {
01019         thArray[i].join();
01020     }
01021
01022     _one_ = std::dynamic_pointer_cast<BANK> (_customer_vec[0]);
01023     _two_ = std::dynamic_pointer_cast<BANK> (_customer_vec[1]);
01024     _three_ = std::dynamic_pointer_cast<BANK> (_customer_vec[2]);
01025     _four_ = std::dynamic_pointer_cast<BANK> (_customer_vec[3]);
01026     _five_ = std::dynamic_pointer_cast<BANK> (_customer_vec[4]);
01027     _six_ = std::dynamic_pointer_cast<BANK> (_customer_vec[5]);
01028
01029     CPPUNIT_ASSERT(_one_->GetBalance() == 15);
01030     CPPUNIT_ASSERT(_two_->GetBalance() == 15);
01031     CPPUNIT_ASSERT(_three_->GetBalance() == 15);
01032     CPPUNIT_ASSERT(_four_->GetBalance() == 15);
01033     CPPUNIT_ASSERT(_five_->GetBalance() == 15);
01034     CPPUNIT_ASSERT(_six_->GetBalance() == 15);
01035 }
```

Here is the call graph for this function:



4.6.3.51 void MyTestCAse::store_null_pointer_throw_runtime_error ( )

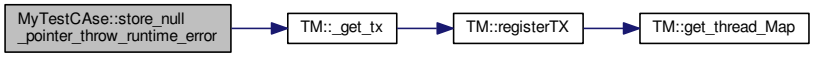
The test function throws runtime error if the client application tries to store the changed working pointer as a null pointer. Runtime error should be thrown.

Definition at line 831 of file MyTestCAse.cpp.

References TM::_get_tx(), and tm.

```
00831                                                   {
00832
00833     std::shared_ptr<OSTM> null_ptr;
00834     std::shared_ptr<TX> tx = tm._get_tx();
00835
00836     tx->store(null_ptr);
00837 }
```

Here is the call graph for this function:



4.6.3.52 void MyTestCAse::tearDown ( ) [inline]

Definition at line 116 of file MyTestCAse.h.

```
00117   {
00118       delete a;
00119       delete b;
00120       delete c;
00121   }
```

4.6.3.53 void MyTestCAse::threaded_functionality_hundred_threads ( )

Testing the library consistent behavior This test transfer 1 unit by 100 threads = 100 ∗1 = 100 After transfer from account has -100, and to account has +100

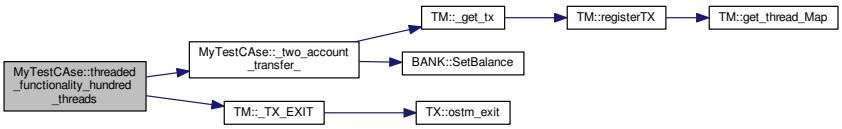
Definition at line 523 of file MyTestCAse.cpp.

References _two_account_transfer_(), TM::_TX_EXIT(), aib_ptr, boi_ptr, and tm.

```
00523                                                   {
00524     tm._TX_EXIT();
00525     std::shared_ptr<BANK> _FROM_BANK_;
00526     std::shared_ptr<BANK> _TO_BANK_;
00527
00528     std::shared_ptr<OSTM> aib_ptr(new AIB(100, 500, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00529     std::shared_ptr<OSTM> boi_ptr(new BOI(200, 500, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00530
00531     int transferAmount = 1;
00532     int threadArraySize = 100;
00533     std::thread thArray[threadArraySize];
00534
00535     for (int i = 0; i < threadArraySize; ++i) {
00536         thArray[i] = std::thread(&MyTestCAse::_two_account_transfer_,
```

```
    this, aib_ptr, boi_ptr, std::ref(tm), transferAmount);
    }

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i].join();
    }

    _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (boi_ptr);
    _TO_BANK_ = std::dynamic_pointer_cast<BANK> (aib_ptr);

    CPPUNIT_ASSERT(_FROM_BANK_->GetBalance() == 400);
    CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == 600);

}
```

Here is the call graph for this function:



4.6.3.54 void MyTestCAse::threaded_functionality_hundred_threads_six_account ( )

Testing the library consistent behavior This test transfer 1 unit by 100 threads from five account to one account After transfer from account has - 100∗1 from the accounts, and to account has +100∗5.

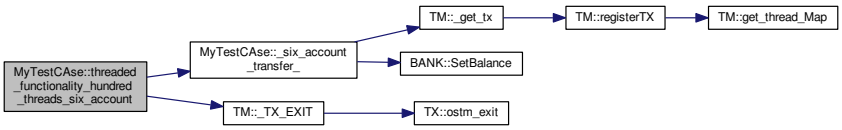
Definition at line 589 of file MyTestCAse.cpp.

References _six_account_transfer_(), TM::_TX_EXIT(), aib_ptr, boa_ptr, boi_ptr, swplc_ptr, tm, ulster_ptr, and unbl_ptr.

```
                                                                    {
    tm._TX_EXIT();
    std::shared_ptr<BANK> _FROM_BANK_ONE,_FROM_BANK_TWO,_FROM_BANK_THREE, _FROM_BANK_FOUR, _FROM_BANK_FIVE;
    std::shared_ptr<BANK> _TO_BANK_;

    std::shared_ptr<OSTM> aib_ptr(new AIB(100, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> boi_ptr(new BOI(200, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> boa_ptr(new BOA(100, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> ulster_ptr(new ULSTER(200, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> unbl_ptr(new UNBL(100, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> swplc_ptr(new SWBPLC(200, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));

    int transferAmount = 1;
    int threadArraySize = 100;
    std::thread thArray[threadArraySize];

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i] = std::thread(&MyTestCAse::_six_account_transfer_, this, aib_ptr, boi_ptr, boa_ptr, ulster_ptr, unbl_ptr, swplc_ptr, std::ref(tm), transferAmount);
    }

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i].join();
    }

    _FROM_BANK_ONE = std::dynamic_pointer_cast<BANK> (boi_ptr);
    _FROM_BANK_TWO = std::dynamic_pointer_cast<BANK> (boa_ptr);
    _FROM_BANK_THREE = std::dynamic_pointer_cast<BANK> (ulster_ptr);
    _FROM_BANK_FOUR = std::dynamic_pointer_cast<BANK> (unbl_ptr);
```

```
    _FROM_BANK_FIVE = std::dynamic_pointer_cast<BANK> (swplc_ptr);
    _TO_BANK_ = std::dynamic_pointer_cast<BANK> (aib_ptr);

    CPPUNIT_ASSERT(_FROM_BANK_ONE->GetBalance() == 400);
    CPPUNIT_ASSERT(_FROM_BANK_TWO->GetBalance() == 400);
    CPPUNIT_ASSERT(_FROM_BANK_THREE->GetBalance() == 400);
    CPPUNIT_ASSERT(_FROM_BANK_FOUR->GetBalance() == 400);
    CPPUNIT_ASSERT(_FROM_BANK_FIVE->GetBalance() == 400);
    CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == 1000);

}
```

Here is the call graph for this function:



4.6.3.55 void MyTestCAse::threaded_functionality_thousand_threads ( )

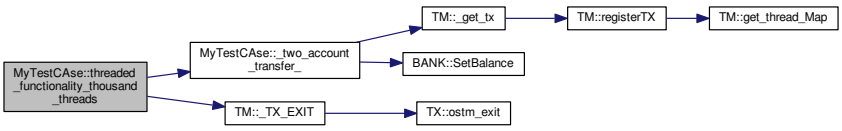
Testing the library consistent behavior This test transfer 1 unit by 1000 threads = 1000 ∗1 = 100 After transfer from account has -1000, and to account has +1000.

Definition at line 556 of file MyTestCAse.cpp.

References _two_account_transfer_(), TM::_TX_EXIT(), aib_ptr, boi_ptr, and tm.

```
556                                                     {
557     tm._TX_EXIT();
558     std::shared_ptr<BANK> _FROM_BANK_;
559     std::shared_ptr<BANK> _TO_BANK_;
560
561     std::shared_ptr<OSTM> aib_ptr(new AIB(100, 500, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
562     std::shared_ptr<OSTM> boi_ptr(new BOI(200, 500, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
563
564     int transferAmount = 1;
565     int threadArraySize = 1000;
566     std::thread thArray[threadArraySize];
567
568     for (int i = 0; i < threadArraySize; ++i) {
569         thArray[i] = std::thread(&MyTestCAse::_two_account_transfer_,
      this, aib_ptr, boi_ptr, std::ref(tm), transferAmount);
570     }
571
572     for (int i = 0; i < threadArraySize; ++i) {
573         thArray[i].join();
574     }
575
576     _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (boi_ptr);
577     _TO_BANK_ = std::dynamic_pointer_cast<BANK> (aib_ptr);
578
579     CPPUNIT_ASSERT(_FROM_BANK_->GetBalance() == -500);
580     CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == 1500);
581
582 }
```

Here is the call graph for this function:



4.6.3.56 void MyTestCAse::threaded_functionality_thousand_threads_six_account ( )

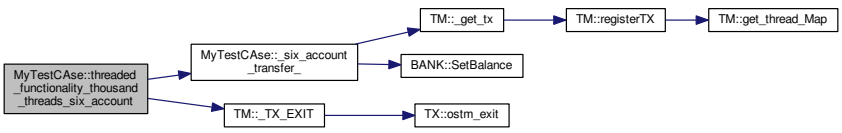
Testing the library consistent behavior This test transfer 1 unit by 1000 threads from five account to one account After transfer from account has - 1000∗1 from the accounts, and to account has +1000∗5.

Definition at line 634 of file MyTestCAse.cpp.

References _six_account_transfer_(), TM::_TX_EXIT(), aib_ptr, boa_ptr, boi_ptr, swplc_ptr, tm, ulster_ptr, and unbl_ptr.

```
00634                                                                   {
00635     tm._TX_EXIT();
00636     std::shared_ptr<BANK> _FROM_BANK_ONE,_FROM_BANK_TWO,_FROM_BANK_THREE, _FROM_BANK_FOUR, _FROM_BANK_FIVE;
00637     std::shared_ptr<BANK> _TO_BANK_;
00638
00639     std::shared_ptr<OSTM> aib_ptr(new AIB(100, 500, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00640     std::shared_ptr<OSTM> boi_ptr(new BOI(200, 500, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00641     std::shared_ptr<OSTM> boa_ptr(new BOA(100, 500, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00642     std::shared_ptr<OSTM> ulster_ptr(new ULSTER(200, 500, "Joe", "Blog", "High street,
       Kilkenny, Co.Kilkenny"));
00643     std::shared_ptr<OSTM> unbl_ptr(new UNBL(100, 500, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00644     std::shared_ptr<OSTM> swplc_ptr(new SWBPLC(200, 500, "Joe", "Blog", "High street,
       Kilkenny, Co.Kilkenny"));
00645
00646     int transferAmount = 1;
00647     int threadArraySize = 1000;
00648     std::thread thArray[threadArraySize];
00649
00650     for (int i = 0; i < threadArraySize; ++i) {
00651         thArray[i] = std::thread(&MyTestCAse::_six_account_transfer_,
      this, aib_ptr, boi_ptr, boa_ptr, ulster_ptr, unbl_ptr,
      swplc_ptr, std::ref(tm), transferAmount);
00652     }
00653
00654     for (int i = 0; i < threadArraySize; ++i) {
00655         thArray[i].join();
00656     }
00657
00658     _FROM_BANK_ONE = std::dynamic_pointer_cast<BANK> (boi_ptr);
00659     _FROM_BANK_TWO = std::dynamic_pointer_cast<BANK> (boa_ptr);
00660     _FROM_BANK_THREE = std::dynamic_pointer_cast<BANK> (ulster_ptr);
00661     _FROM_BANK_FOUR = std::dynamic_pointer_cast<BANK> (unbl_ptr);
00662     _FROM_BANK_FIVE = std::dynamic_pointer_cast<BANK> (swplc_ptr);
00663     _TO_BANK_ = std::dynamic_pointer_cast<BANK> (aib_ptr);
00664
00665     CPPUNIT_ASSERT(_FROM_BANK_ONE->GetBalance() == -500);
00666     CPPUNIT_ASSERT(_FROM_BANK_TWO->GetBalance() == -500);
00667     CPPUNIT_ASSERT(_FROM_BANK_THREE->GetBalance() == -500);
00668     CPPUNIT_ASSERT(_FROM_BANK_FOUR->GetBalance() == -500);
00669     CPPUNIT_ASSERT(_FROM_BANK_FIVE->GetBalance() == -500);
00670     CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == 5500);
00671
00672 }
```

Here is the call graph for this function:



4.6.3.57 void MyTestCAse::TM_get_thread_map ( )

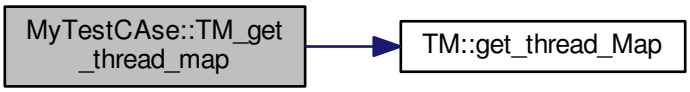
This function testing the returned map from the Transaction Manager class.

Definition at line 903 of file MyTestCAse.cpp.

References TM::get_thread_Map(), and tm.

```
00903                                   {
00904
00905     std::map< std::thread::id, int > localMap;
00906     std::map< std::thread::id, int > tmMap = tm.get_thread_Map();
00907
00908     CPPUNIT_ASSERT(localMap == tmMap);
00909 }
```

Here is the call graph for this function:



4.6.3.58 void MyTestCAse::two_object_transfer_complete ( )

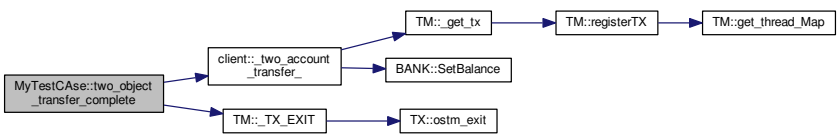
Testing the library consistent behavior Transfer between two objects : the from object transfer 20 to the another object 500 - 20 = 480 AND 500 + 20 = 520.

Definition at line 748 of file MyTestCAse.cpp.

References client::_two_account_transfer_(), TM::_TX_EXIT(), a, aib_ptr, boi_ptr, and tm.

```
00748                                           {
00749     tm._TX_EXIT();
00750     std::shared_ptr<OSTM> aib_ptr(new AIB(100, 500, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00751     std::shared_ptr<OSTM> boi_ptr(new BOI(200, 500, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00752     std::shared_ptr<BANK> _FROM_BANK_;
00753     std::shared_ptr<BANK> _TO_BANK_;
00754     a->_two_account_transfer_(aib_ptr, boi_ptr,
      tm, 20);
00755     _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (boi_ptr);
00756     _TO_BANK_ = std::dynamic_pointer_cast<BANK> (aib_ptr);
00757
00758     CPPUNIT_ASSERT(_FROM_BANK_->GetBalance() == 480);
00759     CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == 520);
00760
00761 }
```

Here is the call graph for this function:



4.6.3.59 void MyTestCAse::two_object_transfer_state_change ( )

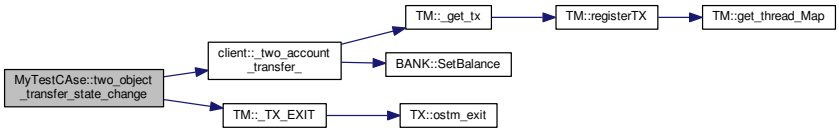
This function proves the objects states must change from the base values.

Definition at line 765 of file MyTestCAse.cpp.

References client::_two_account_transfer_(), TM::_TX_EXIT(), a, boa_ptr, swplc_ptr, and tm.

```
00765                                                   {
00766     tm._TX_EXIT();
00767     std::shared_ptr<OSTM> boa_ptr(new BOA(300, 500, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00768     std::shared_ptr<OSTM> swplc_ptr(new SWBPLC(400, 500, "Joe", "Blog", "High street,
       Kilkenny, Co.Kilkenny"));
00769     std::shared_ptr<BANK> _FROM_BANK_;
00770     std::shared_ptr<BANK> _TO_BANK_;
00771     a->_two_account_transfer_(swplc_ptr, boa_ptr,
      tm, 20);
00772     _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (boa_ptr);
00773     _TO_BANK_ = std::dynamic_pointer_cast<BANK> (swplc_ptr);
00774
00775
00776     CPPUNIT_ASSERT(!(_FROM_BANK_->GetBalance() == 500));
00777     CPPUNIT_ASSERT(!(_TO_BANK_->GetBalance() == 500));
00778
00779 }
```

Here is the call graph for this function:



4.6.4 Member Data Documentation

4.6.4.1 client* MyTestCAse::a [private]

Definition at line 126 of file MyTestCAse.h.

Referenced by nested_transaction_object_test(), two_object_transfer_complete(), and two_object_transfer_state↩_change().

4.6.4.2 std::shared_ptr<OSTM> MyTestCAse::aib_ptr

Definition at line 66 of file MyTestCAse.h.

Referenced by complex_threaded_functionality_hundred_threads(), complex_threaded_functionality_ten↩_threads(), multi_threaded_multiple_object_exchange_test(), multi_threaded_single_object_test_with_ten↩_threads(), nested_hundred_thread_functionality(), nested_thousand_thread_functionality(), threaded_↩functionality_hundred_threads(), threaded_functionality_hundred_threads_six_account(), threaded_functionality↩_thousand_threads(), threaded_functionality_thousand_threads_six_account(), and two_object_transfer_↩complete().

4.6.4.3 client * MyTestCAse::b [private]

Definition at line 126 of file MyTestCAse.h.

4.6.4.4 std::shared_ptr<OSTM> MyTestCAse::boa_ptr

Definition at line 68 of file MyTestCAse.h.

Referenced by threaded_functionality_hundred_threads_six_account(), threaded_functionality_thousand_threads_six_account(), and two_object_transfer_state_change().

4.6.4.5 std::shared_ptr<OSTM> MyTestCAse::boi_ptr

Definition at line 67 of file MyTestCAse.h.

Referenced by complex_threaded_functionality_hundred_threads(), complex_threaded_functionality_ten_threads(), multi_threaded_multiple_object_exchange_test(), nested_hundred_thread_functionality(), nested_thousand_thread_functionality(), threaded_functionality_hundred_threads(), threaded_functionality_hundred_threads_six_account(), threaded_functionality_thousand_threads(), threaded_functionality_thousand_threads_six_account(), and two_object_transfer_complete().

4.6.4.6 client * MyTestCAse::c [private]

Definition at line 126 of file MyTestCAse.h.

4.6.4.7 std::shared_ptr<OSTM> MyTestCAse::swplc_ptr

Definition at line 69 of file MyTestCAse.h.

Referenced by threaded_functionality_hundred_threads_six_account(), threaded_functionality_thousand_threads_six_account(), and two_object_transfer_state_change().

4.6.4.8 TM& MyTestCAse::tm = TM::Instance()

Definition at line 65 of file MyTestCAse.h.

Referenced by _one_account_transfer_(), compare_Transaction_Manager_singleton_instance(), complex_threaded_functionality_hundred_threads(), complex_threaded_functionality_ten_threads(), decrease_nesting(), decrease_nesting_fail(), increase_nesting(), increase_nesting_fail(), multi_threaded_multiple_object_exchange_test(), multi_threaded_multiple_objects_test(), multi_threaded_single_object_test_with_ten_threads(), nested_hundred_thread_functionality(), nested_thousand_thread_functionality(), nested_transaction_object_test(), object_not_registered_throw_runtime_error(), register_null_pointer_throw_runtime_error(), single_threaded_multiple_object_test(), store_null_pointer_throw_runtime_error(), threaded_functionality_hundred_threads(), threaded_functionality_hundred_threads_six_account(), threaded_functionality_thousand_threads(), threaded_functionality_thousand_threads_six_account(), TM_get_thread_map(), two_object_transfer_complete(), and two_object_transfer_state_change().

4.6.4.9 std::shared_ptr<OSTM> MyTestCAse::ulster_ptr

Definition at line 70 of file MyTestCAse.h.

Referenced by nested_transaction_object_test(), threaded_functionality_hundred_threads_six_account(), and threaded_functionality_thousand_threads_six_account().

4.6.4.10 std::shared_ptr<OSTM> MyTestCAse::unbl_ptr

Definition at line 71 of file MyTestCAse.h.

Referenced by nested_transaction_object_test(), threaded_functionality_hundred_threads_six_account(), and threaded_functionality_thousand_threads_six_account().

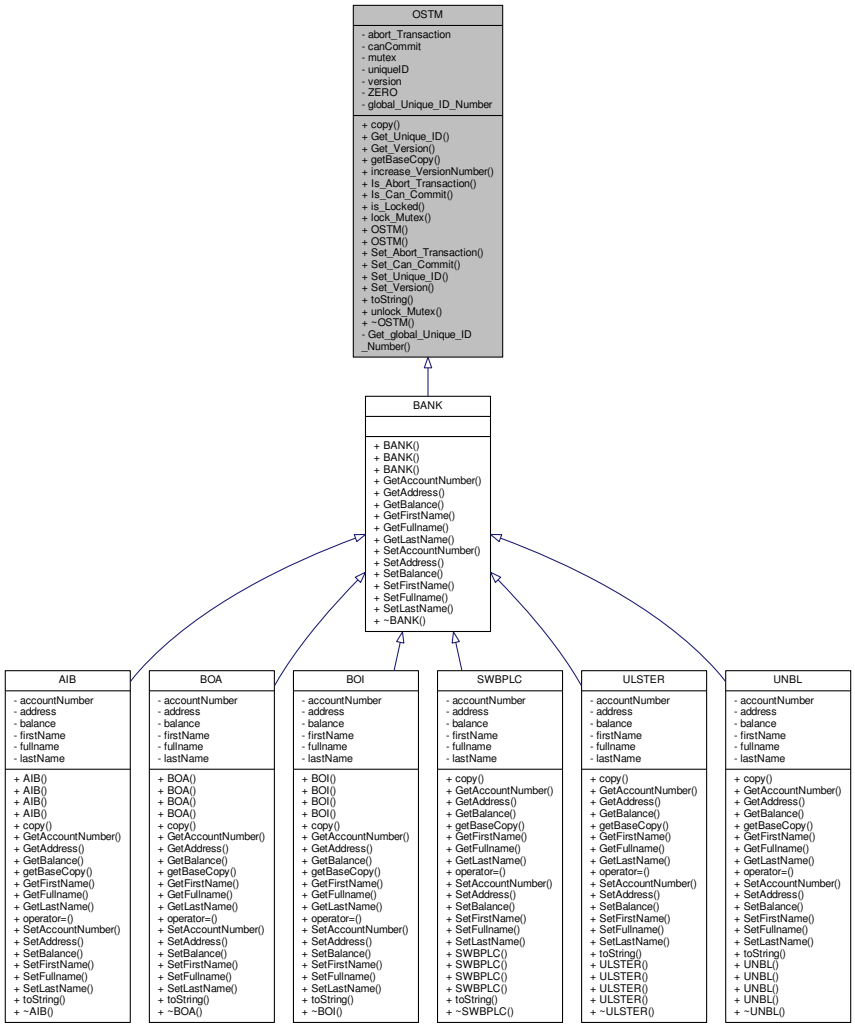
The documentation for this class was generated from the following files:

- MyTestCAse.h
- MyTestCAse.cpp

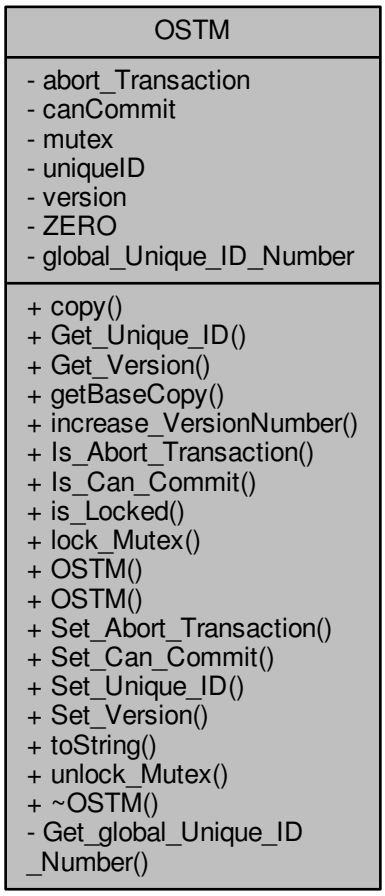
4.7 OSTM Class Reference

```
#include <OSTM.h>
```

Inheritance diagram for OSTM:



Collaboration diagram for OSTM:



Public Member Functions

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

  OSTM required virtual method for deep copy.
- int Get_Unique_ID () const

  getter for unique id
- int Get_Version () const

  getter for version number
- 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.
- void increase_VersionNumber ()

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

  NOT USED YET.
- bool Is_Can_Commit () const

  NOT USED YET.

- bool is_Locked ()

  object unique lock, try locks mutex return boolean value depends on the lock state
- void lock_Mutex ()

  object unique lock, locks mutex
- OSTM ()

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

  OSTM Custom Constructor.
- void Set_Abort_Transaction (bool abortTransaction)

  NOT USED YET.
- void Set_Can_Commit (bool canCommit)

  NOT USED YET.
- void Set_Unique_ID (int uniqueID)

  setter for unique id
- void Set_Version (int version)

  setter for version number
- virtual void toString ()

  OSTM required virtual method for display object.
- void unlock_Mutex ()

  object unique lock, unlocks mutex
- virtual ~OSTM ()

  De-constructor.

Private Member Functions

- int Get_global_Unique_ID_Number ()

Private Attributes

- bool abort_Transaction
- bool canCommit
- std::mutex mutex

  Object built in lock.
- int uniqueID
- int version
- const int ZERO = 0

  Meaningful display for value 0.

Static Private Attributes

- static int global_Unique_ID_Number = 0

  Unique object number increase at object creation.

4.7.1 Detailed Description

Definition at line 17 of file OSTM.h.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 OSTM::OSTM ( )

OSTM Constructor.

Default constructor.

Parameters

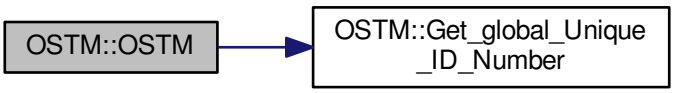
version	indicates the version number of the inherited child pointer
uniqueID	is a unique identifier assigned to every object registered in OSTM library
canCommit	NOT USED YET
abort_Transaction	NOT USED YET

Definition at line 20 of file OSTM.cpp.

References abort_Transaction, canCommit, Get_global_Unique_ID_Number(), uniqueID, version, and ZERO.

```
00021 {
00022     this->version = ZERO;
00023     this->uniqueID = Get_global_Unique_ID_Number(); //
      ++global_Unique_ID_Number;
00024     this->canCommit = true;
00025     this->abort_Transaction = false;
00026 }
```

Here is the call graph for this function:



4.7.2.2 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	NOT USED YET
abort_Transaction	NOT USED YET

Definition at line 36 of file OSTM.cpp.

References abort_Transaction, canCommit, uniqueID, and version.

```
00037 {
00038    // std::cout << "OSTM COPY CONSTRUCTOR" << global_Unique_ID_Number << std::endl;
00039     this->uniqueID = _unique_id_;
00040     this->version = _version_number_;
00041     this->canCommit = true;
00042     this->abort_Transaction = false;
00043 }
```

4.7.2.3 OSTM::~OSTM ( ) [virtual]

De-constructor.

De-constructor

Definition at line 48 of file OSTM.cpp.

```
00048             {
00049     //std::cout << "[OSTM DELETE]" << std::endl;
00050 }
```

4.7.3 Member Function Documentation

4.7.3.1 virtual void OSTM::copy (std::shared_ptr< OSTM > *from*, std::shared_ptr< OSTM > *to*) [inline], [virtual]

OSTM required virtual method for deep copy.

Reimplemented in BOI, AIB, BOA, SWBPLC, ULSTER, and UNBL.

Definition at line 34 of file OSTM.h.

```
00034 {};
```

4.7.3.2 int OSTM::Get_global_Unique_ID_Number ( ) [private]

Returning global_Unique_ID_Number to the constructor

If global_Unique_ID_Number equals to 10000000 then reset back to ZERO, to make sure the value of global_↩ Unique_ID_Number never exceed the MAX_INT value

Definition at line 56 of file OSTM.cpp.

References global_Unique_ID_Number.

Referenced by OSTM().

```
00056                                   {
00057     if(global_Unique_ID_Number > 10000000)
00058         global_Unique_ID_Number = 0;
00059     return ++global_Unique_ID_Number;
00060 }
```

4.7.3.3 int OSTM::Get_Unique_ID ( ) const

getter for unique id

Parameters

uniqueID	int

Definition at line 73 of file OSTM.cpp.

References uniqueID.

Referenced by toString(), UNBL::toString(), ULSTER::toString(), SWBPLC::toString(), BOA::toString(), BOI::to↩String(), and AIB::toString().

```
00074 {
00075     return uniqueID;
00076 }
```

4.7.3.4 int OSTM::Get_Version ( ) const

getter for version number

Parameters

version	int

Definition at line 89 of file OSTM.cpp.

References version.

Referenced by toString(), UNBL::toString(), ULSTER::toString(), SWBPLC::toString(), BOA::toString(), BOI::to↩String(), and AIB::toString().

```
00090 {
00091     return version;
00092 }
```

4.7.3.5 virtual std::shared_ptr<OSTM> OSTM::getBaseCopy (std::shared_ptr< OSTM > *object*) [inline], [virtual]

OSTM required virtual method for returning a pointer that is copy of the original pointer.

Reimplemented in AIB, BOA, BOI, SWBPLC, ULSTER, and UNBL.

Definition at line 38 of file OSTM.h.

```
00038 {};//std::cout << "[OSTM GETBASECOPY]" << std::endl;};
```

4.7.3.6 void OSTM::increase_VersionNumber ( )

commit time increase version number to child object

Parameters

version	int

Definition at line 97 of file OSTM.cpp.

References version.

Referenced by toString().

```
00098 {
00099     this->version += 1;
00100 }
```

4.7.3.7 bool OSTM::Is_Abort_Transaction ( ) const

NOT USED YET.

Parameters

abort_Transaction	boolean

Definition at line 126 of file OSTM.cpp.

References abort_Transaction.

Referenced by toString().

```
00126                                    {
00127     return abort_Transaction;
00128 }
```

4.7.3.8 bool OSTM::Is_Can_Commit ( ) const

NOT USED YET.

Parameters

canCommit	boolean

Definition at line 112 of file OSTM.cpp.

References canCommit.

Referenced by toString().

```
00112                                {
00113     return canCommit;
00114 }
```

4.7.3.9 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.

References mutex.

Referenced by toString().

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

4.7.3.10 void OSTM::lock_Mutex ( )

object unique lock, locks mutex

Parameters

mutex	std::mutex

Definition at line 133 of file OSTM.cpp.

References mutex.

Referenced by toString().

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

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

NOT USED YET.

Parameters

abort_Transaction	boolean

Definition at line 119 of file OSTM.cpp.

References abort_Transaction.

Referenced by toString().

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

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

NOT USED YET.

Parameters

canCommit	boolean

Definition at line 105 of file OSTM.cpp.

References canCommit.

Referenced by toString().

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

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

setter for unique id

Parameters

uniqueID	int

Definition at line 66 of file OSTM.cpp.

References uniqueID.

Referenced by UNBL::copy(), ULSTER::copy(), SWBPLC::copy(), BOA::copy(), AIB::copy(), BOI::copy(), and to↩String().

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

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

setter for version number

Parameters

version	int

Definition at line 81 of file OSTM.cpp.

References version.

Referenced by toString().

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

4.7.3.15 virtual void OSTM::toString ( ) [inline], [virtual]

OSTM required virtual method for display object.

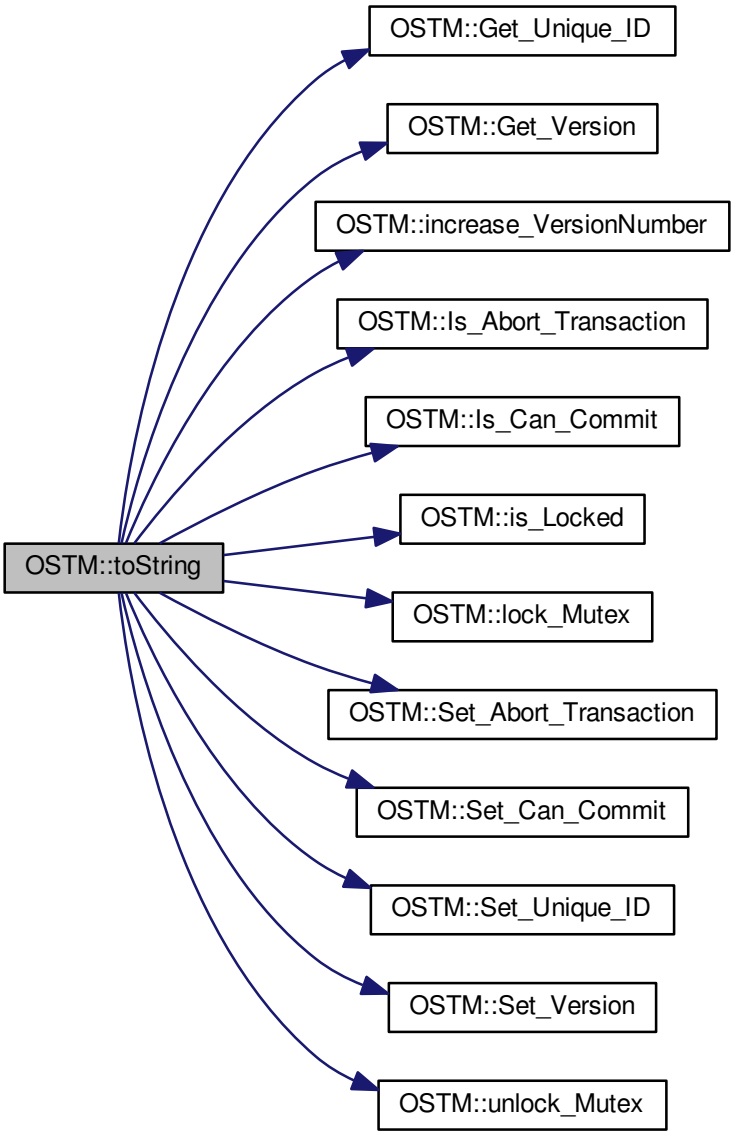
Reimplemented in AIB, BOA, BOI, SWBPLC, ULSTER, and UNBL.

Definition at line 42 of file OSTM.h.

References canCommit, Get_Unique_ID(), Get_Version(), increase_VersionNumber(), Is_Abort_Transaction(), Is_Can_Commit(), is_Locked(), lock_Mutex(), Set_Abort_Transaction(), Set_Can_Commit(), Set_Unique_ID(), Set_Version(), uniqueID, unlock_Mutex(), and version.

```
00042 {};
```

Here is the call graph for this function:



4.7.3.16 void OSTM::unlock_Mutex ( )

object unique lock, unlocks mutex

Parameters

mutex	std::mutex

Definition at line 140 of file OSTM.cpp.

References mutex.

Referenced by toString().

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

4.7.4 Member Data Documentation

4.7.4.1 bool OSTM::abort_Transaction [private]

Definition at line 108 of file OSTM.h.

Referenced by Is_Abort_Transaction(), OSTM(), and Set_Abort_Transaction().

4.7.4.2 bool OSTM::canCommit [private]

Definition at line 104 of file OSTM.h.

Referenced by Is_Can_Commit(), OSTM(), Set_Can_Commit(), and toString().

4.7.4.3 int OSTM::global_Unique_ID_Number = 0 [static], [private]

Unique object number increase at object creation.

Definition at line 112 of file OSTM.h.

Referenced by Get_global_Unique_ID_Number().

4.7.4.4 std::mutex OSTM::mutex [private]

Object built in lock.

Definition at line 120 of file OSTM.h.

Referenced by is_Locked(), lock_Mutex(), and unlock_Mutex().

4.7.4.5 int OSTM::uniqueID [private]

Definition at line 100 of file OSTM.h.

Referenced by Get_Unique_ID(), OSTM(), Set_Unique_ID(), and toString().

4.7.4.6 int OSTM::version [private]

Definition at line 96 of file OSTM.h.

Referenced by Get_Version(), increase_VersionNumber(), OSTM(), Set_Version(), and toString().

4.7.4.7 const int OSTM::ZERO = 0 [private]

Meaningful display for value 0.

Definition at line 116 of file OSTM.h.

Referenced by OSTM().

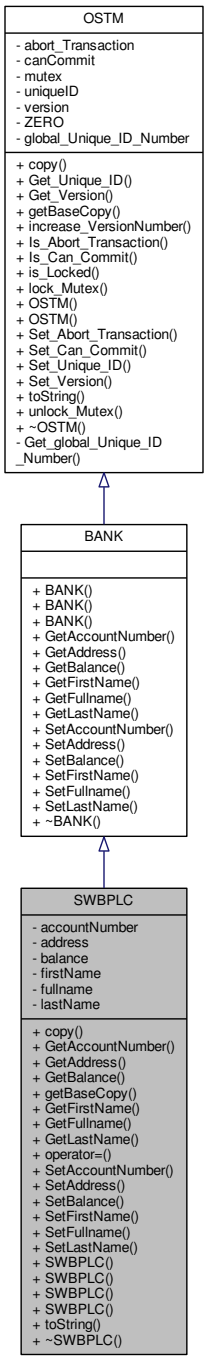
The documentation for this class was generated from the following files:

- OSTM.h
- OSTM.cpp

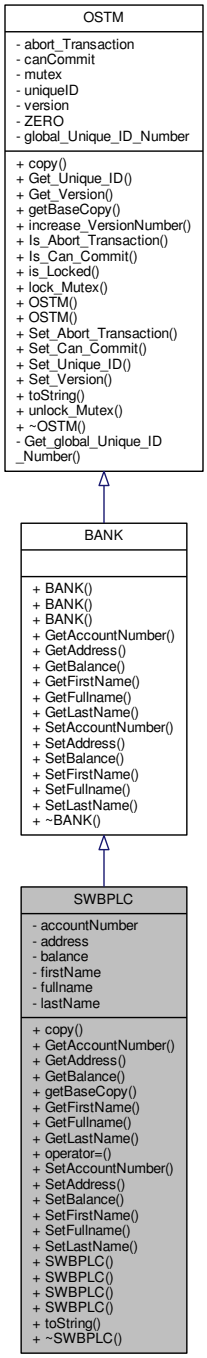
4.8 SWBPLC Class Reference

```
#include <SWBPLC.h>
```

Inheritance diagram for SWBPLC:



Collaboration diagram for SWBPLC:



Public Member Functions

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

  OSTM required virtual method for deep copy.
- virtual int GetAccountNumber () const
- virtual std::string GetAddress () const
- virtual double GetBalance () const

- 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 std::string GetFirstName () const
- virtual std::string GetFullname () const
- virtual std::string GetLastName () const
- SWBPLC operator= (const SWBPLC &orig)
- virtual void SetAccountNumber (int accountNumber)
- virtual void SetAddress (std::string address)
- virtual void SetBalance (double balance)
- virtual void SetFirstName (std::string firstName)
- virtual void SetFullname (std::string fullname)
- virtual void SetLastName (std::string lastName)
- SWBPLC ()
- SWBPLC (int accountNumber, double balance, std::string firstName, std::string lastName, std::string address)
- SWBPLC (std::shared_ptr< BANK > obj, int _version, int _unique_id)
- SWBPLC (const SWBPLC &orig)
- virtual void toString ()

  OSTM required virtual method for display object.
- virtual ~SWBPLC ()

Private Attributes

- int accountNumber
- std::string address
- double balance
- std::string firstName
- std::string fullname
- std::string lastName

4.8.1 Detailed Description

Definition at line 19 of file SWBPLC.h.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 SWBPLC::SWBPLC ( ) [inline]

Definition at line 24 of file SWBPLC.h.

References accountNumber, address, balance, firstName, fullname, and lastName.

Referenced by getBaseCopy(), and SWBPLC().

```
00024             : BANK() {
00025         this->accountNumber = 0;
00026         this->balance = 50;
00027         this->firstName = "Joe";
00028         this->lastName = "Blog";
00029         this->address = "High street, Carlow";
00030         this->fullname = firstName + " " + lastName;
00031     };
```

4.8.2.2 SWBPLC::SWBPLC (int *accountNumber,* double *balance,* std::string *firstName,* std::string *lastName,* std::string *address*) [inline]

Definition at line 35 of file SWBPLC.h.

References accountNumber, address, balance, firstName, fullname, and lastName.

```
00035                                                                                                              :
       BANK() {
00036          this->accountNumber = accountNumber;
00037          this->balance = balance;
00038          this->firstName = firstName;
00039          this->lastName = lastName;
00040          this->address = address;
00041          this->fullname = firstName + " " + lastName;
00042     };
```

4.8.2.3 SWBPLC::SWBPLC (std::shared_ptr< BANK > *obj,* int *_version,* int *_unique_id*) [inline]

Definition at line 46 of file SWBPLC.h.

References accountNumber, address, balance, firstName, fullname, lastName, and SWBPLC().

```
00046                                                                      : BANK(_version, _unique_id) {
00047
00048          this->accountNumber = obj->GetAccountNumber();
00049          this->balance = obj->GetBalance();
00050          this->firstName = obj->GetFirstName();
00051          this->lastName = obj->GetLastName();
00052          this->address = obj->GetAddress();
00053          this->fullname = obj->GetFirstName() + " " + obj->GetLastName();
00054
00055     };
```

Here is the call graph for this function:

4.8.2.4 SWBPLC::SWBPLC (const SWBPLC & *orig*)

Definition at line 12 of file SWBPLC.cpp.

```
00012                            {
00013 }
```

4.8.2.5 SWBPLC::∼SWBPLC ( ) [virtual]

Definition at line 15 of file SWBPLC.cpp.

Referenced by operator=().

```
00015                 {
00016 }
```

4.8.3 Member Function Documentation

4.8.3.1 void SWBPLC::copy (std::shared_ptr< OSTM > *from*, std::shared_ptr< OSTM > *to*) [virtual]

OSTM required virtual method for deep copy.

Reimplemented from OSTM.

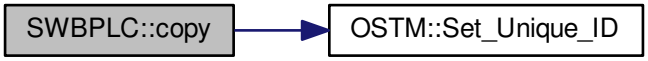
Definition at line 34 of file SWBPLC.cpp.

References OSTM::Set_Unique_ID().

Referenced by operator=().

```
00034                                                                   {
00035
00036     std::shared_ptr<SWBPLC> objTO = std::dynamic_pointer_cast<SWBPLC>(to);
00037     std::shared_ptr<SWBPLC> objFROM = std::dynamic_pointer_cast<SWBPLC>(from);
00038     objTO->Set_Unique_ID(objFROM->Get_Unique_ID());
00039     objTO->Set_Version(objFROM->Get_Version());
00040     objTO->SetAccountNumber(objFROM->GetAccountNumber());
00041     objTO->SetBalance(objFROM->GetBalance());
00042
00043
00044 }
```

Here is the call graph for this function:



4.8.3.2 int SWBPLC::GetAccountNumber ( ) const [virtual]

Reimplemented from BANK.

Definition at line 77 of file SWBPLC.cpp.

References accountNumber.

Referenced by operator=(), and toString().

```
00077                                   {
00078     return accountNumber;
00079 }
```

4.8.3.3 std::string SWBPLC::GetAddress ( ) const [virtual]

Reimplemented from BANK.

Definition at line 61 of file SWBPLC.cpp.

References address.

Referenced by operator=().

```
00061                                   {
00062     return address;
00063 }
```

4.8.3.4 double SWBPLC::GetBalance ( ) const [virtual]

Reimplemented from BANK.

Definition at line 69 of file SWBPLC.cpp.

References balance.

Referenced by operator=(), and toString().

```
00069                                 {
00070     return balance;
00071 }
```

4.8.3.5 std::shared_ptr< OSTM > SWBPLC::getBaseCopy (std::shared_ptr< OSTM > *object*) [virtual]

OSTM required virtual method for returning a pointer that is copy of the original pointer.

Reimplemented from OSTM.

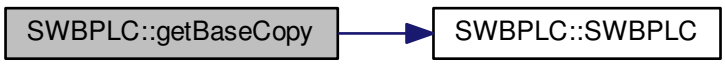
Definition at line 22 of file SWBPLC.cpp.

References SWBPLC().

Referenced by operator=().

```
00023 {
00024     std::shared_ptr<BANK> objTO = std::dynamic_pointer_cast<BANK>(object);
00025     std::shared_ptr<BANK> obj(new SWBPLC(objTO,object->Get_Version(),object->Get_Unique_ID()));
00026     std::shared_ptr<OSTM> ostm_obj = std::dynamic_pointer_cast<OSTM>(obj);

00027     return ostm_obj;
00028 }
```

Here is the call graph for this function:



4.8.3.6 std::string SWBPLC::GetFirstName ( ) const [virtual]

Reimplemented from BANK.

Definition at line 93 of file SWBPLC.cpp.

References firstName.

Referenced by operator=(), and toString().

```
00093                                   {
00094     return firstName;
00095 }
```

4.8.3.7 std::string SWBPLC::GetFullname ( ) const [virtual]

Reimplemented from BANK.

Definition at line 101 of file SWBPLC.cpp.

References fullname.

Referenced by operator=().

```
00101                                   {
00102     return fullname;
00103 }
```

4.8.3.8 std::string SWBPLC::GetLastName ( ) const [virtual]

Reimplemented from BANK.

Definition at line 85 of file SWBPLC.cpp.

References lastName.

Referenced by operator=(), and toString().

```
00085                                   {
00086     return lastName;
00087 }
```

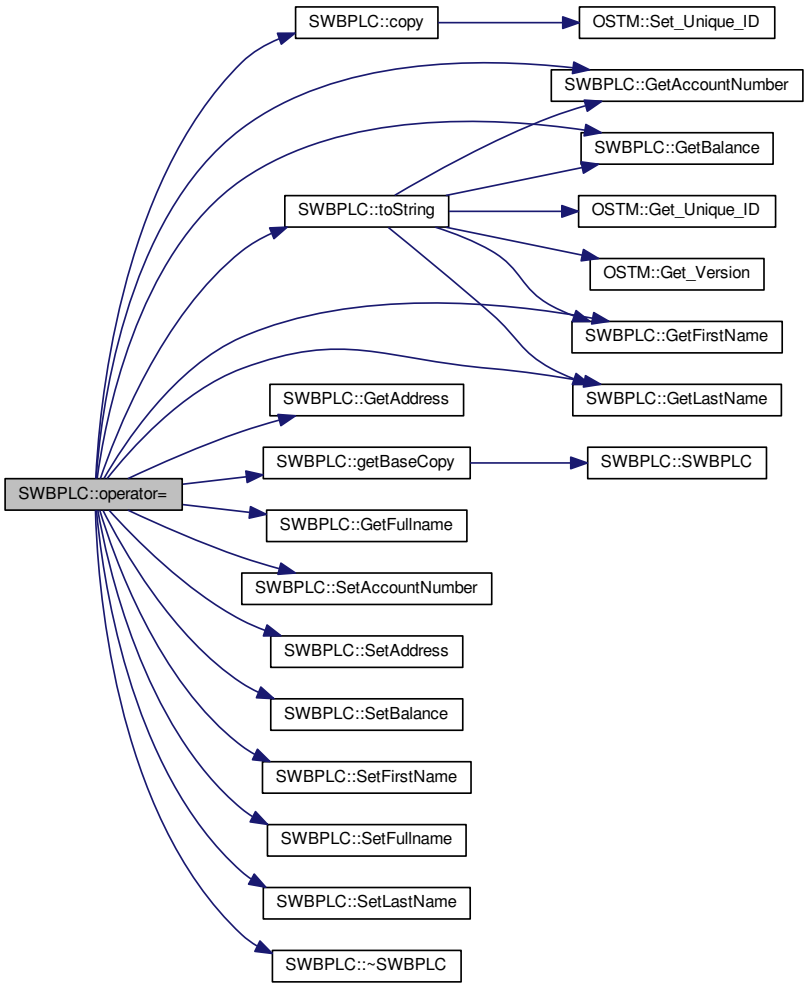
4.8.3.9 SWBPLC SWBPLC::operator= (const SWBPLC & *orig*) [inline]

Definition at line 63 of file SWBPLC.h.

References accountNumber, address, balance, copy(), firstName, fullname, GetAccountNumber(), GetAddress(), GetBalance(), getBaseCopy(), GetFirstName(), GetFullname(), GetLastName(), lastName, SetAccountNumber(), SetAddress(), SetBalance(), SetFirstName(), SetFullname(), SetLastName(), toString(), and ~SWBPLC().

```
00063 {};
```

Here is the call graph for this function:



4.8.3.10 void SWBPLC::SetAccountNumber (int *accountNumber*) [virtual]

Reimplemented from BANK.

Definition at line 73 of file SWBPLC.cpp.

References accountNumber.

Referenced by operator=().

```
00073                                           {
00074     this->accountNumber = accountNumber;
00075 }
```

4.8.3.11 void SWBPLC::SetAddress (std::string *address*) [virtual]

Reimplemented from BANK.

Definition at line 57 of file SWBPLC.cpp.

References address.

Referenced by operator=().

```
00057                                       {
00058     this->address = address;
00059 }
```

4.8.3.12 void SWBPLC::SetBalance (double *balance*) [virtual]

Reimplemented from BANK.

Definition at line 65 of file SWBPLC.cpp.

References balance.

Referenced by operator=().

```
00065                                     {
00066     this->balance = balance;
00067 }
```

4.8.3.13 void SWBPLC::SetFirstName (std::string *firstName*) [virtual]

Reimplemented from BANK.

Definition at line 89 of file SWBPLC.cpp.

References firstName.

Referenced by operator=().

```
00089                                           {
00090     this->firstName = firstName;
00091 }
```

4.8.3.14 void SWBPLC::SetFullname (std::string *fullname*) [virtual]

Reimplemented from BANK.

Definition at line 97 of file SWBPLC.cpp.

References fullname.

Referenced by operator=().

```
00097                                         {
00098     this->fullname = fullname;
00099 }
```

4.8.3.15 void SWBPLC::SetLastName (std::string *lastName*) [virtual]

Reimplemented from BANK.

Definition at line 81 of file SWBPLC.cpp.

References lastName.

Referenced by operator=().

```
00081                                                {
00082     this->lastName = lastName;
00083 }
```

4.8.3.16 void SWBPLC::toString ( ) [virtual]

OSTM required virtual method for display object.

Reimplemented from OSTM.

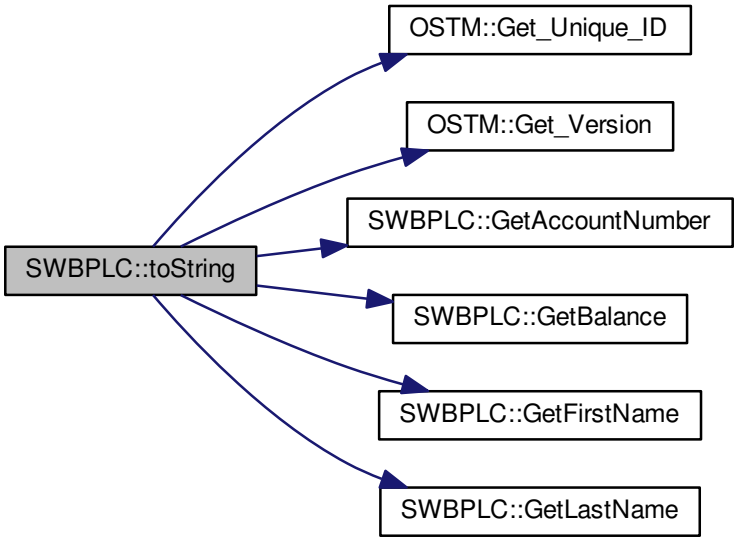
Definition at line 52 of file SWBPLC.cpp.

References OSTM::Get_Unique_ID(), OSTM::Get_Version(), GetAccountNumber(), GetBalance(), GetFirstName(), and GetLastName().

Referenced by operator=().

```
00053 {
00054      std::cout << "\nSWBPLC BANK" << "\nUnique ID : " << this->Get_Unique_ID() << "\nInt
       account : " << this->GetAccountNumber() << "\nDouble value : " << this->
      GetBalance() << "\nFirst name: " << this->GetFirstName() << "\nLast name : " <<
      this->GetLastName()  << "\nVersion number : " << this->Get_Version() << std::endl;
00055 }
```

Here is the call graph for this function:



4.8.4 Member Data Documentation

4.8.4.1 int SWBPLC::accountNumber [private]

Definition at line 96 of file SWBPLC.h.

Referenced by GetAccountNumber(), operator=(), SetAccountNumber(), and SWBPLC().

4.8.4.2 std::string SWBPLC::address [private]

Definition at line 98 of file SWBPLC.h.

Referenced by GetAddress(), operator=(), SetAddress(), and SWBPLC().

4.8.4.3 double SWBPLC::balance [private]

Definition at line 97 of file SWBPLC.h.

Referenced by GetBalance(), operator=(), SetBalance(), and SWBPLC().

4.8.4.4 std::string SWBPLC::firstName [private]

Definition at line 94 of file SWBPLC.h.

Referenced by GetFirstName(), operator=(), SetFirstName(), and SWBPLC().

4.8.4.5 std::string SWBPLC::fullname [private]

Definition at line 93 of file SWBPLC.h.

Referenced by GetFullname(), operator=(), SetFullname(), and SWBPLC().

4.8.4.6 std::string SWBPLC::lastName [private]

Definition at line 95 of file SWBPLC.h.

Referenced by GetLastName(), operator=(), SetLastName(), and SWBPLC().

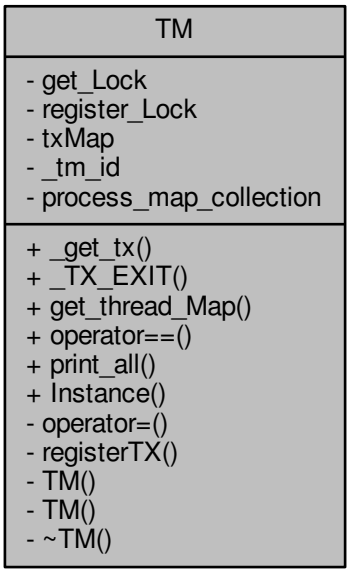
The documentation for this class was generated from the following files:

- SWBPLC.h
- SWBPLC.cpp

4.9 TM Class Reference

```
#include <TM.h>
```

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
- std::map< std::thread::id, int > get_thread_Map ()

  get_thread_Map std::map, returning a map to store all unique ID from all objects from all transactions within the main process
- bool operator== (const TM &rhs) const
- 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.

Private Member Functions

- TM & operator= (const TM &)=delete

  TM copy operator, prevent from copying the Transaction Manager.
- void registerTX ()

  get_thread_Map returning and map to insert to the process_map_collection as an inner value
- TM ()=default

  TM constructor, prevent from multiple instantiation.
- TM (const TM &)

  TM copy constructor, prevent from copying the Transaction Manager.
- ~TM ()=default

  TM de-constructor, prevent from deletion.

Private Attributes

- std::mutex get_Lock
- std::mutex register_Lock
- std::map< std::thread::id, std::shared_ptr< TX > > txMap

Static Private Attributes

- static pid_t _tm_id
- static std::map< pid_t, std::map< std::thread::id, int > > process_map_collection

  STATIC GLOBAL MAP Collection to store all process associated keys to find when deleting transactions.

4.9.1 Detailed Description

Definition at line 21 of file TM.h.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 TM::TM ( ) [private],[default]

TM constructor, prevent from multiple instantiation.

4.9.2.2 TM::~TM ( ) [private],[default]

TM de-constructor, prevent from deletion.

4.9.2.3 TM::TM (const TM &) [private]

TM copy constructor, prevent from copying the Transaction Manager.

4.9.3 Member Function Documentation

4.9.3.1 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

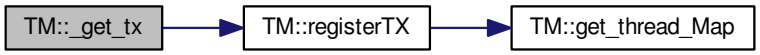
Definition at line 79 of file TM.cpp.

References get_Lock, registerTX(), and txMap.

Referenced by MyTestCAse::_collection_bject_(), MyTestCAse::_complex_transfer_(), client::_complex_transfer_(), client::_nesting_(), MyTestCAse::_nesting_(), MyTestCAse::_one_account_transfer_(), MyTestCAse::_six_account_transfer_(), client::_six_account_transfer_(), client::_two_account_transfer_(), MyTestCAse::_two_account_transfer_(), MyTestCAse::decrease_nesting(), MyTestCAse::decrease_nesting_fail(), MyTestCAse::increase_nesting(), MyTestCAse::increase_nesting_fail(), MyTestCAse::object_not_registered_throw_runtime_error(), MyTestCAse::register_null_pointer_throw_runtime_error(), and MyTestCAse::store_null_pointer_throw_runtime_error().

```
{
    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;

}
```

Here is the call graph for this function:



4.9.3.2 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 102 of file TM.cpp.

References TX::ostm_exit(), process_map_collection, and txMap.

Referenced by MyTestCAse::complex_threaded_functionality_hundred_threads(), MyTestCAse::complex_threaded_functionality_ten_threads(), MyTestCAse::multi_threaded_multiple_object_exchange_test(), MyTestCAse::multi_threaded_multiple_objects_test(), MyTestCAse::multi_threaded_single_object_test_with_ten_threads(), MyTestCAse::nested_hundred_thread_functionality(), MyTestCAse::nested_thousand_thread_functionality(), MyTestCAse::nested_transaction_object_test(), MyTestCAse::single_threaded_multiple_object_test(), MyTestCAse::threaded_functionality_hundred_threads(), MyTestCAse::threaded_functionality_hundred_threads_six_account(), MyTestCAse::threaded_functionality_thousand_threads(), MyTestCAse::threaded_functionality_thousand_threads_six_account(), MyTestCAse::two_object_transfer_complete(), and MyTestCAse::two_object_transfer_state_change().

```
00102                 {
00103     TX tx(std::this_thread::get_id());
00104     int ppid = getpid();
00105     std::map<int, std::map< std::thread::id, int >>::iterator process_map_collection_Iterator =
      TM::process_map_collection.find(ppid);
00106     if (process_map_collection_Iterator != TM::process_map_collection.end()) {
00107
00108         for (auto current = process_map_collection_Iterator->second.begin(); current !=
      process_map_collection_Iterator->second.end(); ++current) {
00109             /*
00110              * Delete all transaction associated with the actual main process

00111              */
00112             txMap.erase(current->first);
00113         }
00114         TM::process_map_collection.erase(ppid);
00115
00116     }
00117     tx.ostm_exit();
00118 }
```

Here is the call graph for this function:

4.9.3.3 std::map< std::thread::id, int > TM::get_thread_Map ( )

get_thread_Map std::map, returning a map to store all unique ID from all objects from all transactions within the main process

Parameters

thread_Map	std::map< int, int >,

Definition at line 134 of file TM.cpp.

Referenced by operator==(), registerTX(), and MyTestCAse::TM_get_thread_map().

```
00134                                         {
00135     std::map< std::thread::id, int > thread_Map;
00136     return thread_Map;
00137 }
```

4.9.3.4 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 28 of file TM.cpp.

References _tm_id.

Referenced by MyTestCAse::compare_Transaction_Manager_singleton_instance().

```
00028                    {
00029     static TM _instance;
00030     _instance._tm_id = getpid();
00031
00032     return _instance;
00033 }
```

4.9.3.5 TM& TM::operator= (const TM &) [private],[delete]

TM copy operator, prevent from copying the Transaction Manager.

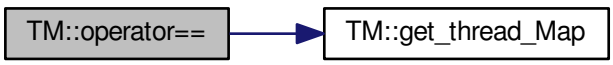
4.9.3.6 bool TM::operator== (const TM & *rhs*) const [inline]

Definition at line 91 of file TM.h.

References get_thread_Map().

```
00091                                        {
00092         return &rhs == this;
00093     }
```

Here is the call graph for this function:



4.9.3.7 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 122 of file TM.cpp.

References get_Lock, and txMap.

```
00122                   {
00123     get_Lock.lock();
00124     for (auto current = txMap.begin(); current != txMap.end(); ++current) {
00125         std::cout << "KEY : " << current->first << std::endl;
00126     }
00127     get_Lock.unlock();
00128 }
```

4.9.3.8 void TM::registerTX ( ) [private]

get_thread_Map returning and map to insert to the process_map_collection as an inner value

registerTX void, register a new TX Transaction object into ythe txMap/Transaction Map to manage all the transactions within the shared library

registerTX void, register transaction into txMap

Parameters

txMap	std::map, collection to store all transaction created by the Transaction Manager
register_Lock	std::mutex, used by the lock_guard to protect shared map from race conditions
guard	std::lock_guard, locks the register_Lock mutex, unlock automatically when goes out of the scope

Definition at line 43 of file TM.cpp.

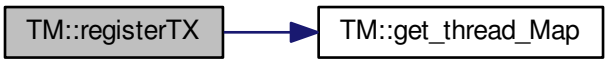
References get_thread_Map(), process_map_collection, register_Lock, and txMap.

Referenced by _get_tx().

```
00044 {
00045     std::lock_guard<std::mutex> guard(register_Lock);
00046     int ppid = getpid();
00047     std::map<int, std::map< std::thread::id, int >>::iterator process_map_collection_Iterator =
      TM::process_map_collection.find(ppid);
00048     if (process_map_collection_Iterator == TM::process_map_collection.end()) {
00049         /*
00050          * Register main process/application to the global map

00051          */
00052         std::map< std::thread::id, int >map = get_thread_Map();
00053         TM::process_map_collection.insert({ppid, map});
00054
00055     }
00056     std::map<std::thread::id, std::shared_ptr < TX>>::iterator it = txMap.find(
      std::this_thread::get_id());
00057     if (it == txMap.end()) {
00058         std::shared_ptr<TX> _transaction_object(new TX(std::this_thread::get_id()));
00059         txMap.insert({std::this_thread::get_id(), _transaction_object});
00060         /*
00061          * Get the map if registered first time

00062          */
00063         process_map_collection_Iterator = TM::process_map_collection.find(ppid);
```

```
        /*
         * Insert to the GLOBAL MAP as a helper to clean up at end of main process

         */
        process_map_collection_Iterator->second.insert({std::this_thread::get_id(), 1});

    }

}
```

Here is the call graph for this function:



4.9.4 Member Data Documentation

4.9.4.1 int TM::_tm_id [static], [private]

Parameters

_tm↩ _id	pid_t, process id determine the actual process between process in the shared OSTM library

Definition at line 67 of file TM.h.

Referenced by Instance().

4.9.4.2 std::mutex TM::get_Lock [private]

Parameters

register_Lock	std::mutex, used in the _get_tx function

Definition at line 63 of file TM.h.

Referenced by _get_tx(), and print_all().

4.9.4.3 std::map< int, std::map< std::thread::id, int > > TM::process_map_collection [static], [private]

STATIC GLOBAL MAP Collection to store all process associated keys to find when deleting transactions.

Parameters

process_map_collection	std::map
static	Global std::map process_map_collection store all transactional objects/pointers

Definition at line 47 of file TM.h.

Referenced by _TX_EXIT(), and registerTX().

4.9.4.4 std::mutex TM::register_Lock [private]

Parameters

register_Lock	std::mutex, used in the registerTX function

Definition at line 59 of file TM.h.

Referenced by registerTX().

4.9.4.5 std::map<std::thread::id, std::shared_ptr<TX> > TM::txMap [private]

Parameters

txMap	std::map, store all transactional objects created with Transaction Manager

Definition at line 42 of file TM.h.

Referenced by _get_tx(), _TX_EXIT(), print_all(), and registerTX().

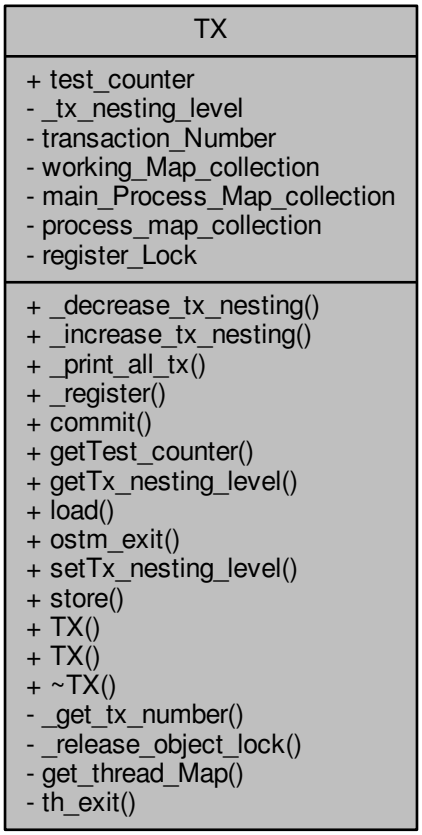
The documentation for this class was generated from the following files:

- TM.h
- TM.cpp

4.10 TX Class Reference

```
#include <TX.h>
```

Collaboration diagram for TX:



Public Member Functions

- void _decrease_tx_nesting ()

  Remove TX nesting level by one.
- void _increase_tx_nesting ()

  Add TX nesting level by one.
- void _print_all_tx ()
- void _register (std::shared_ptr< OSTM > object)

  Register OSTM pointer into STM library.
- bool commit ()

  Commit transactional changes.
- int getTest_counter ()

  getTest_counter TESTING ONLY!!! returning the value of the test_counter stored, number of rollbacks
- int getTx_nesting_level () const
- 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 ostm_exit ()

Delete all map entries associated with the main process.

- void setTx_nesting_level (int _tx_nesting_level)
- void store (std::shared_ptr< OSTM > object)

  Store transactional changes.
- TX (std::thread::id id)

  Constructor.
- TX (const TX &orig)

  Default copy constructor.
- ~TX ()

  De-constructor.

Static Public Attributes

- static int test_counter = 0

Private Member Functions

- const std::thread::id _get_tx_number () const

  _get_tx_number returning the transaction uniqe identifier
- void _release_object_lock ()

  _release_object_lock void, is get called from commit function, with the purpose to release the locks on all the objects participating in the transaction
- std::map< int, int > get_thread_Map ()

  get_thread_Map returning and map to insert to the process_map_collection as an inner value
- void th_exit ()

  Clean up all associated values by the thread delete from working_Map_collection, it is an automated function.

Private Attributes

- int _tx_nesting_level
- std::thread::id transaction_Number

  Returning the transaction number.
- std::map< int, std::shared_ptr< OSTM > > working_Map_collection

  MAP Collection to store OSTM∗ parent based pointers to make invisible changes during isolated transaction.

Static Private Attributes

- static std::map< int, std::shared_ptr< OSTM > > main_Process_Map_collection

  STATIC GLOBAL MAP Collection to store OSTM∗ parent based pointers to control/lock and compare objects version number within transactions.
- static std::map< pid_t, std::map< int, int > > process_map_collection

  STATIC GLOBAL MAP Collection to store all process associated keys to find when deleting transactions.
- static std::mutex register_Lock

Friends

- class TM

4.10.1 Detailed Description

Definition at line 24 of file TX.h.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 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.

References _tx_nesting_level, and transaction_Number.

```
00031                       {
00032     this->transaction_Number = id;
00033     this->_tx_nesting_level = 0;
00034 }
```

4.10.2.2 TX::~TX ( )

De-constructor.

Definition at line 38 of file TX.cpp.

```
00038         {
00039
00040 }
```

4.10.2.3 TX::TX (const TX & *orig*)

Default copy constructor.

Definition at line 44 of file TX.cpp.

```
00044                       {
00045
00046 }
```

4.10.3 Member Function Documentation

4.10.3.1 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.

References _tx_nesting_level.

Referenced by commit().

```
00316                                {
00317    // std::cout << "[this->_tx_nesting_level] = " << this->_tx_nesting_level << std::endl;
00318     this->_tx_nesting_level -= 1;
00319 ;
00320 }
```

4.10.3.2 const std::thread::id TX::_get_tx_number ( ) const [private]

_get_tx_number returning the transaction uniqe identifier

_get_tx_number std::thread::id, returning the thread id that has assigned the given transaction

Parameters

transaction_Number	int

Definition at line 331 of file TX.cpp.

References transaction_Number.

```
00331                                          {
00332     return transaction_Number;
00333 }
```

4.10.3.3 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.

References _tx_nesting_level.

```
00307                                {
00308
00309     this->_tx_nesting_level += 1;
00310     // std::cout << "[this->_tx_nesting_level] = " << this->_tx_nesting_level << std::endl;
00311 }
```

4.10.3.4 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.

References process_map_collection, and working_Map_collection.

```
00346                        {
00347
00348     std::cout << "[PRINTALLTHREAD]" << std::endl;
00349     std::map< int, std::shared_ptr<OSTM> >::iterator it;
00350     /*
00351      * All registered thread id in the TX global

00352      */
00353     int ppid = getpid();
00354     std::map<int, std::map< int, int >>::iterator process_map_collection_Iterator =
      TX::process_map_collection.find(ppid);
00355     if (process_map_collection_Iterator != TX::process_map_collection.end()) {
00356
00357         for (auto current = process_map_collection_Iterator->second.begin(); current !=
      process_map_collection_Iterator->second.end(); ++current) {
00358             it = working_Map_collection.find(current->first);
00359             if(it != working_Map_collection.end()){
00360                 std::cout << "[Unique number ] : " <<it->second->Get_Unique_ID() << std::endl;
00361             }
00362
00363
00364         }
00365
00366     }
00367 }
```

4.10.3.5 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

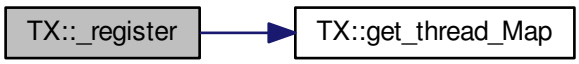
Definition at line 104 of file TX.cpp.

References get_thread_Map(), main_Process_Map_collection, process_map_collection, register_Lock, and working_Map_collection.

```
00104                                          {
00105     /*
00106      * MUST USE SHARED LOCK TO PROTECT SHARED GLOBAL MAP/COLLECTION

00107      */
00108     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]") );
    }

    int ppid = getpid();
    std::map<int, 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()) {

        working_Map_collection.insert({object->Get_Unique_ID(), object->getBaseCopy(object)});
    }

}
```

Here is the call graph for this function:



4.10.3.6 void TX::_release_object_lock ( ) [private]

_release_object_lock void, is get called from commit function, with the purpose to release the locks on all the objects participating in the transaction

Release the locks in objects with transaction associated collection

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 release the lock on object

Definition at line 286 of file TX.cpp.

References main_Process_Map_collection, and working_Map_collection.

Referenced by commit().

```
00286                                {
00287
00288     std::map< int, std::shared_ptr<OSTM> >::iterator working_Map_collection_Object_Shared_Pointer_Iterator;
00289     std::map<int, std::shared_ptr<OSTM>>::iterator main_Process_Map_collection_Iterator;
00290     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++) {
00291
00292             main_Process_Map_collection_Iterator =
      TX::main_Process_Map_collection.find((
      working_Map_collection_Object_Shared_Pointer_Iterator->second)->Get_Unique_ID());
00293             if (main_Process_Map_collection_Iterator !=
      TX::main_Process_Map_collection.end()) {
00294                 /*
00295                  * Release object lock

00296                  */
00297                 (main_Process_Map_collection_Iterator)->second->unlock_Mutex();
00298
00299             }
00300         }
00301 }
```

4.10.3.7 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.

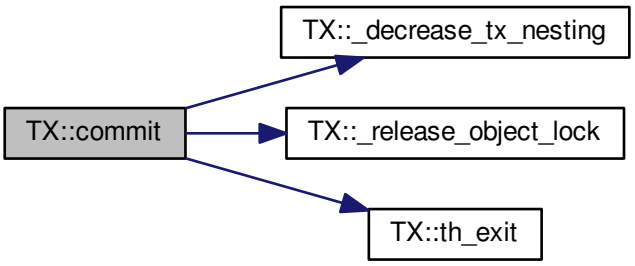
References _decrease_tx_nesting(), _release_object_lock(), _tx_nesting_level, main_Process_Map_collection, test_counter, th_exit(), and working_Map_collection.

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

00208      */
```

```
    if (this->_tx_nesting_level > 0) {
        _decrease_tx_nesting();
        return true;
    }

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

    std::map<int, std::shared_ptr<OSTM>>::iterator main_Process_Map_collection_Iterator;
    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());
            /*
             * Throws runtime error if object can not find

             */
            if(main_Process_Map_collection_Iterator == TX::main_Process_Map_collection.end())
            {
                throw std::runtime_error(std::string("[RUNTIME ERROR : CAN'T FIND OBJECT COMMIT FUNCTION]"));
            }

        /*
         * Busy wait WHILE object locked by other thread

         */
        while(!(main_Process_Map_collection_Iterator->second)->is_Locked());

        if (main_Process_Map_collection_Iterator->second->Get_Version() > working_Map_collection_Object_Shared_Pointer_Iterator->second->Get_Version()) {

            working_Map_collection_Object_Shared_Pointer_Iterator->second->Set_Can_Commit(false);
            can_Commit = false;
            break;
        } else {

            working_Map_collection_Object_Shared_Pointer_Iterator->second->Set_Can_Commit(true);
        }
    }
    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:



4.10.3.8 std::map< int, int > TX::get_thread_Map ( ) [private]

get_thread_Map returning and map to insert to the process_map_collection as an inner value

get_thread_Map std::map, returning a map to store all unique ID from all objects from all transactions within the main process

Parameters

thread_Map	std::map< int, int >,

Definition at line 338 of file TX.cpp.

Referenced by _register().

```
{
    std::map< int, int > thread_Map;
    return thread_Map;
}
```

4.10.3.9 int TX::getTest_counter ( )

getTest_counter TESTING ONLY!!! returning the value of the test_counter stored, number of rollbacks

Definition at line 324 of file TX.cpp.

References test_counter.

```
{
    return TX::test_counter;
}
```

4.10.3.10 int TX::getTx_nesting_level ( ) const

Definition at line 374 of file TX.cpp.

References _tx_nesting_level.

```
00374                                   {
00375     return _tx_nesting_level;
00376 }
```

4.10.3.11 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.

References working_Map_collection.

```
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 }
```

4.10.3.12 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.

References main_Process_Map_collection, and process_map_collection.

Referenced by TM::_TX_EXIT().

```
00072                    {
00073     std::map<int, std::shared_ptr<OSTM>>::iterator main_Process_Map_collection_Iterator;
00074
00075     int ppid = getpid();
00076     std::map<int, 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 }
```

4.10.3.13 void TX::setTx_nesting_level (int *_tx_nesting_level*)

Definition at line 370 of file TX.cpp.

References _tx_nesting_level.

```
00370                                                {
00371     this->_tx_nesting_level = _tx_nesting_level;
00372 }
```

4.10.3.14 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.

References working_Map_collection.

```
{
    /*
     * 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; }
}
```

4.10.3.15 void TX::th_exit ( ) [private]

Clean up all associated values by the thread delete from working_Map_collection, it is an automated function.

th_exit void, delete all std::shared_ptr<OSTM> elements from working_Map_collection, that store pointers to working objects

Parameters

working_Map_collection	std::map, store std::shared_ptr<OSTM> transaction pointers

Definition at line 52 of file TX.cpp.

References _tx_nesting_level, and working_Map_collection.

Referenced by commit().

```
{

    if (this->_tx_nesting_level > 0) {
        /*
         * Active nested transactions running in background, do not delete anything yet
         */
    } else {
        /*
         * Remove all elements map entries from transaction and clear the map
         */
        working_Map_collection.clear();
    }
}
```

4.10.4 Friends And Related Function Documentation

4.10.4.1 friend class TM [friend]

Only TM Transaction Manager can create instance of TX Transaction

Definition at line 70 of file TX.h.

4.10.5 Member Data Documentation

4.10.5.1 int TX::_tx_nesting_level [private]

Parameters

_tx_nesting_level	int

Definition at line 104 of file TX.h.

Referenced by _decrease_tx_nesting(), _increase_tx_nesting(), commit(), getTx_nesting_level(), setTx_nesting_level(), th_exit(), and TX().

4.10.5.2 std::map< int, std::shared_ptr< OSTM > > TX::main_Process_Map_collection [static], [private]

STATIC GLOBAL MAP Collection to store OSTM∗ parent based pointers to control/lock and compare objects version number within transactions.

Parameters

main_Process_Map_collection	std::map
static	Global std::map main_Process_Map_collection store all transactional objects/pointers

Definition at line 110 of file TX.h.

Referenced by _register(), _release_object_lock(), commit(), and ostm_exit().

4.10.5.3 std::map< int, std::map< int, int > > TX::process_map_collection [static], [private]

STATIC GLOBAL MAP Collection to store all process associated keys to find when deleting transactions.

Parameters

process_map_collection	std::map
static	Global std::map process_map_collection store all transactional objects/pointers

Definition at line 115 of file TX.h.

Referenced by _print_all_tx(), _register(), and ostm_exit().

4.10.5.4 std::mutex TX::register_Lock [static], [private]

Parameters

register_Lock	std::mutex to control shared access on MAIN MAP
static	shared std:mutex register_Lock to protect writes into shared global collection

Definition at line 123 of file TX.h.

Referenced by _register().

4.10.5.5 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().

4.10.5.6 std::thread::id TX::transaction_Number [private]

Returning the transaction number.

Parameters

transaction_Number	std::thread::id NOT USED YET

Definition at line 100 of file TX.h.

Referenced by _get_tx_number(), and TX().

4.10.5.7 std::map< int, std::shared_ptr<OSTM> > TX::working_Map_collection [private]

MAP Collection to store OSTM∗ parent based pointers to make invisible changes during isolated transaction.

Parameters

working_Map_collection	std::map

Definition at line 94 of file TX.h.

Referenced by _print_all_tx(), _register(), _release_object_lock(), commit(), load(), store(), and th_exit().

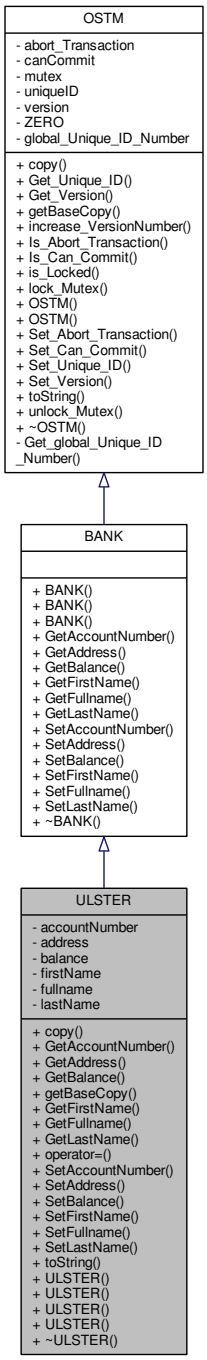
The documentation for this class was generated from the following files:

- TX.h
- TX.cpp

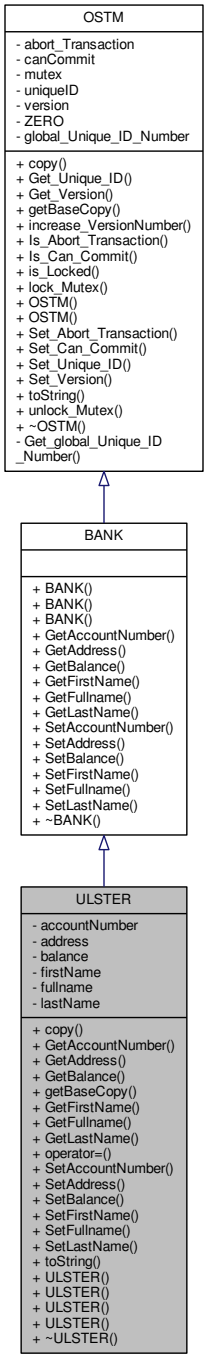
4.11 ULSTER Class Reference

```
#include <ULSTER.h>
```

Inheritance diagram for ULSTER:



Collaboration diagram for ULSTER:



Public Member Functions

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

  OSTM required virtual method for deep copy.
- virtual int GetAccountNumber () const
- virtual std::string GetAddress () const
- virtual double GetBalance () const

- 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 std::string GetFirstName () const
- virtual std::string GetFullname () const
- virtual std::string GetLastName () const
- ULSTER operator= (const ULSTER &orig)
- virtual void SetAccountNumber (int accountNumber)
- virtual void SetAddress (std::string address)
- virtual void SetBalance (double balance)
- virtual void SetFirstName (std::string firstName)
- virtual void SetFullname (std::string fullname)
- virtual void SetLastName (std::string lastName)
- virtual void toString ()

  OSTM required virtual method for display object.
- ULSTER ()
- ULSTER (int accountNumber, double balance, std::string firstName, std::string lastName, std::string address)
- ULSTER (std::shared_ptr< BANK > obj, int _version, int _unique_id)
- ULSTER (const ULSTER &orig)
- virtual ~ULSTER ()

Private Attributes

- int accountNumber
- std::string address
- double balance
- std::string firstName
- std::string fullname
- std::string lastName

4.11.1 Detailed Description

Definition at line 19 of file ULSTER.h.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 ULSTER::ULSTER ( ) [inline]

Definition at line 24 of file ULSTER.h.

References accountNumber, address, balance, firstName, fullname, and lastName.

Referenced by getBaseCopy(), and ULSTER().

```
00024             : BANK() {
00025         this->accountNumber = 0;
00026         this->balance = 50;
00027         this->firstName = "Joe";
00028         this->lastName = "Blog";
00029         this->address = "High street, Carlow";
00030         this->fullname = firstName + " " + lastName;
00031     };
```

4.11.2.2 ULSTER::ULSTER (int *accountNumber,* double *balance,* std::string *firstName,* std::string *lastName,* std::string *address*) [inline]

Definition at line 35 of file ULSTER.h.

References accountNumber, address, balance, firstName, fullname, and lastName.

```
00035                                                                                           :
       BANK() {
00036         this->accountNumber = accountNumber;
00037         this->balance = balance;
00038         this->firstName = firstName;
00039         this->lastName = lastName;
00040         this->address = address;
00041         this->fullname = firstName + " " + lastName;
00042     };
```

4.11.2.3 ULSTER::ULSTER (std::shared_ptr< BANK > *obj,* int *_version,* int *_unique_id*) [inline]

Definition at line 46 of file ULSTER.h.

References accountNumber, address, balance, firstName, fullname, lastName, and ULSTER().

```
00046                                                             : BANK(_version, _unique_id) {
00047 
00048         this->accountNumber = obj->GetAccountNumber();
00049         this->balance = obj->GetBalance();
00050         this->firstName = obj->GetFirstName();
00051         this->lastName = obj->GetLastName();
00052         this->address = obj->GetAddress();
00053         this->fullname = obj->GetFirstName() + " " + obj->GetLastName();
00054     };
```

Here is the call graph for this function:

4.11.2.4 ULSTER::ULSTER (const ULSTER & *orig*)

Definition at line 13 of file ULSTER.cpp.

```
00013                                    {
00014 }
```

4.11.2.5 ULSTER::∼ULSTER ( ) [virtual]

Definition at line 16 of file ULSTER.cpp.

Referenced by operator=().

```
00016                    {
00017 }
```

4.11.3 Member Function Documentation

4.11.3.1 void ULSTER::copy (std::shared_ptr< OSTM > *from*, std::shared_ptr< OSTM > *to*) [virtual]

OSTM required virtual method for deep copy.

Reimplemented from OSTM.

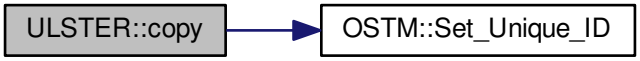
Definition at line 35 of file ULSTER.cpp.

References OSTM::Set_Unique_ID().

Referenced by operator=().

```
00035                                                                   {
00036
00037     std::shared_ptr<ULSTER> objTO = std::dynamic_pointer_cast<ULSTER>(to);
00038     std::shared_ptr<ULSTER> objFROM = std::dynamic_pointer_cast<ULSTER>(from);
00039     objTO->Set_Unique_ID(objFROM->Get_Unique_ID());
00040     objTO->Set_Version(objFROM->Get_Version());
00041     objTO->SetAccountNumber(objFROM->GetAccountNumber());
00042     objTO->SetBalance(objFROM->GetBalance());
00043
00044
00045 }
```

Here is the call graph for this function:



4.11.3.2 int ULSTER::GetAccountNumber ( ) const [virtual]

Reimplemented from BANK.

Definition at line 78 of file ULSTER.cpp.

References accountNumber.

Referenced by operator=(), and toString().

```
00078                                   {
00079     return accountNumber;
00080 }
```

4.11.3.3 std::string ULSTER::GetAddress ( ) const [virtual]

Reimplemented from BANK.

Definition at line 62 of file ULSTER.cpp.

References address.

Referenced by operator=().

```
00062                                   {
00063     return address;
00064 }
```

4.11.3.4 double ULSTER::GetBalance ( ) const [virtual]

Reimplemented from BANK.

Definition at line 70 of file ULSTER.cpp.

References balance.

Referenced by operator=(), and toString().

```
00070                                   {
00071     return balance;
00072 }
```

4.11.3.5 std::shared_ptr< OSTM > ULSTER::getBaseCopy (std::shared_ptr< OSTM > *object*) [virtual]

OSTM required virtual method for returning a pointer that is copy of the original pointer.

Reimplemented from OSTM.

Definition at line 23 of file ULSTER.cpp.

References ULSTER().

Referenced by operator=().

```
00024 {
00025     std::shared_ptr<BANK> objTO = std::dynamic_pointer_cast<BANK>(object);
00026     std::shared_ptr<BANK> obj(new ULSTER(objTO,object->Get_Version(),object->Get_Unique_ID()));
00027     std::shared_ptr<OSTM> ostm_obj = std::dynamic_pointer_cast<OSTM>(obj);

00028     return ostm_obj;
00029 }
```

Here is the call graph for this function:

4.11.3.6 std::string ULSTER::GetFirstName ( ) const [virtual]

Reimplemented from BANK.

Definition at line 94 of file ULSTER.cpp.

References firstName.

Referenced by operator=(), and toString().

```
00094                                   {
00095     return firstName;
00096 }
```

4.11.3.7 std::string ULSTER::GetFullname ( ) const [virtual]

Reimplemented from BANK.

Definition at line 102 of file ULSTER.cpp.

References fullname.

Referenced by operator=().

```
00102                                   {
00103     return fullname;
00104 }
```

4.11.3.8 std::string ULSTER::GetLastName ( ) const [virtual]

Reimplemented from BANK.

Definition at line 86 of file ULSTER.cpp.

References lastName.

Referenced by operator=(), and toString().

```
00086                                   {
00087     return lastName;
00088 }
```

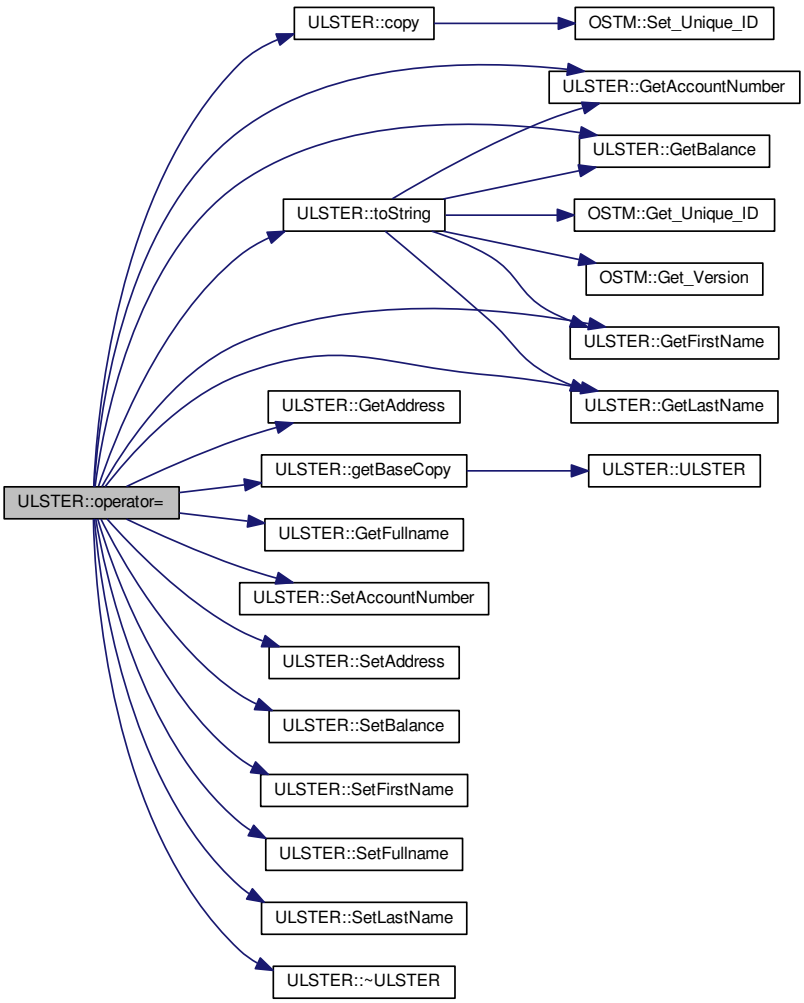
4.11.3.9 ULSTER ULSTER::operator= (const ULSTER & *orig*) [inline]

Definition at line 62 of file ULSTER.h.

References accountNumber, address, balance, copy(), firstName, fullname, GetAccountNumber(), GetAddress(), GetBalance(), getBaseCopy(), GetFirstName(), GetFullname(), GetLastName(), lastName, SetAccountNumber(), SetAddress(), SetBalance(), SetFirstName(), SetFullname(), SetLastName(), toString(), and ~ULSTER().

```
00062 {};
```

Here is the call graph for this function:



4.11.3.10 void ULSTER::SetAccountNumber (int *accountNumber*) [virtual]

Reimplemented from BANK.

Definition at line 74 of file ULSTER.cpp.

References accountNumber.

Referenced by operator=().

```
00074                                          {
00075     this->accountNumber = accountNumber;
00076 }
```

4.11.3.11 void ULSTER::SetAddress (std::string *address*) [virtual]

Reimplemented from BANK.

Definition at line 58 of file ULSTER.cpp.

References address.

Referenced by operator=().

```
00058                                       {
00059     this->address = address;
00060 }
```

4.11.3.12 void ULSTER::SetBalance (double *balance*) [virtual]

Reimplemented from BANK.

Definition at line 66 of file ULSTER.cpp.

References balance.

Referenced by operator=().

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

4.11.3.13 void ULSTER::SetFirstName (std::string *firstName*) [virtual]

Reimplemented from BANK.

Definition at line 90 of file ULSTER.cpp.

References firstName.

Referenced by operator=().

```
00090                                           {
00091     this->firstName = firstName;
00092 }
```

4.11.3.14 void ULSTER::SetFullname (std::string *fullname*) [virtual]

Reimplemented from BANK.

Definition at line 98 of file ULSTER.cpp.

References fullname.

Referenced by operator=().

```
00098                                         {
00099     this->fullname = fullname;
00100 }
```

4.11.3.15 void ULSTER::SetLastName (std::string *lastName*) [virtual]

Reimplemented from BANK.

Definition at line 82 of file ULSTER.cpp.

References lastName.

Referenced by operator=().

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

4.11.3.16 void ULSTER::toString ( ) [virtual]

OSTM required virtual method for display object.

Reimplemented from OSTM.

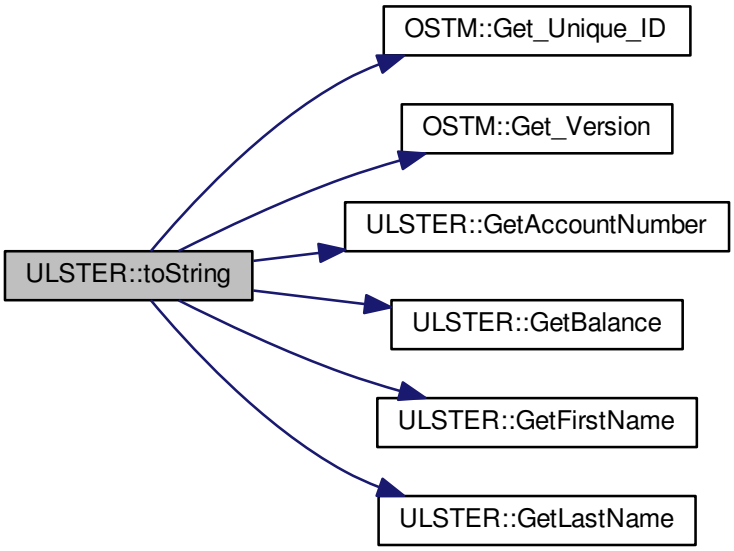
Definition at line 53 of file ULSTER.cpp.

References OSTM::Get_Unique_ID(), OSTM::Get_Version(), GetAccountNumber(), GetBalance(), GetFirstName(), and GetLastName().

Referenced by operator=().

```
00054 {
00055    std::cout << "\nULSTER BANK" << "\nUnique ID : " << this->Get_Unique_ID() << "\nInt account
       : " << this->GetAccountNumber() << "\nDouble value : " << this->
      GetBalance() << "\nFirst name: " << this->GetFirstName() << "\nLast name : " <<
      this->GetLastName()  << "\nVersion number : " << this->Get_Version() << std::endl;
00056 }
```

Here is the call graph for this function:



4.11.4 Member Data Documentation

4.11.4.1 int ULSTER::accountNumber [private]

Definition at line 95 of file ULSTER.h.

Referenced by GetAccountNumber(), operator=(), SetAccountNumber(), and ULSTER().

4.11.4.2 std::string ULSTER::address [private]

Definition at line 97 of file ULSTER.h.

Referenced by GetAddress(), operator=(), SetAddress(), and ULSTER().

4.11.4.3 double ULSTER::balance [private]

Definition at line 96 of file ULSTER.h.

Referenced by GetBalance(), operator=(), SetBalance(), and ULSTER().

4.11.4.4 std::string ULSTER::firstName [private]

Definition at line 93 of file ULSTER.h.

Referenced by GetFirstName(), operator=(), SetFirstName(), and ULSTER().

4.11.4.5 std::string ULSTER::fullname [private]

Definition at line 92 of file ULSTER.h.

Referenced by GetFullname(), operator=(), SetFullname(), and ULSTER().

4.11.4.6 std::string ULSTER::lastName [private]

Definition at line 94 of file ULSTER.h.

Referenced by GetLastName(), operator=(), SetLastName(), and ULSTER().

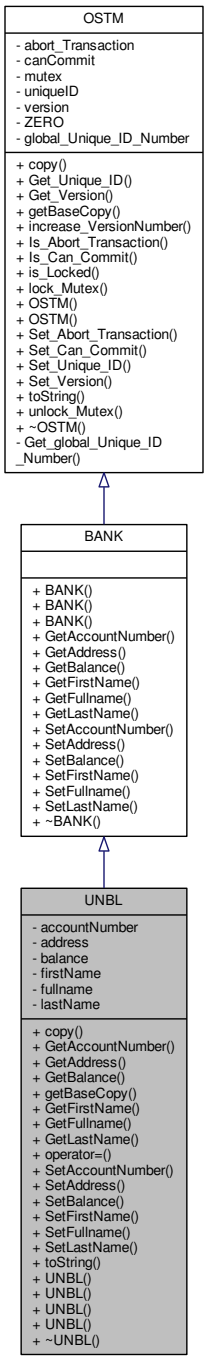
The documentation for this class was generated from the following files:

- ULSTER.h
- ULSTER.cpp

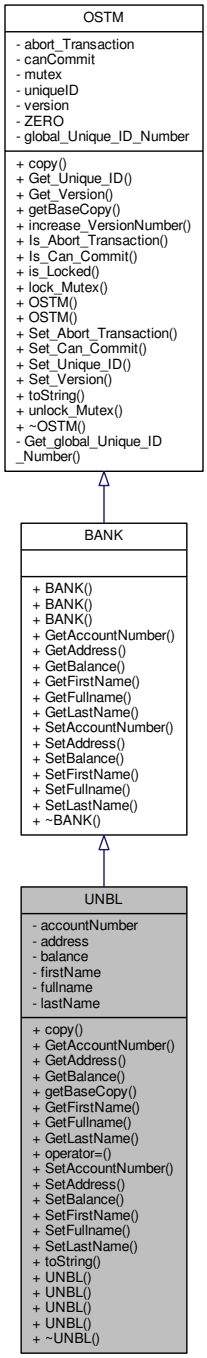
4.12 UNBL Class Reference

```
#include <UNBL.h>
```

Inheritance diagram for UNBL:



Collaboration diagram for UNBL:



Public Member Functions

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

  OSTM required virtual method for deep copy.
- virtual int GetAccountNumber () const
- virtual std::string GetAddress () const
- virtual double GetBalance () const

- 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 std::string GetFirstName () const
- virtual std::string GetFullname () const
- virtual std::string GetLastName () const
- UNBL operator= (const UNBL &orig)
- virtual void SetAccountNumber (int accountNumber)
- virtual void SetAddress (std::string address)
- virtual void SetBalance (double balance)
- virtual void SetFirstName (std::string firstName)
- virtual void SetFullname (std::string fullname)
- virtual void SetLastName (std::string lastName)
- virtual void toString ()

  OSTM required virtual method for display object.
- UNBL ()
- UNBL (int accountNumber, double balance, std::string firstName, std::string lastName, std::string address)
- UNBL (std::shared_ptr< BANK > obj, int _version, int _unique_id)
- UNBL (const UNBL &orig)
- virtual ~UNBL ()

Private Attributes

- int accountNumber
- std::string address
- double balance
- std::string firstName
- std::string fullname
- std::string lastName

4.12.1 Detailed Description

Definition at line 17 of file UNBL.h.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 UNBL::UNBL ( ) [inline]

Definition at line 22 of file UNBL.h.

References accountNumber, address, balance, firstName, fullname, and lastName.

Referenced by getBaseCopy(), and UNBL().

```
00022            : BANK() {
00023        this->accountNumber = 0;
00024        this->balance = 50;
00025        this->firstName = "Joe";
00026        this->lastName = "Blog";
00027        this->address = "High street, Carlow";
00028        this->fullname = firstName + " " + lastName;
00029    };
```

4.12.2.2 UNBL::UNBL (int *accountNumber,* double *balance,* std::string *firstName,* std::string *lastName,* std::string *address*) [inline]

Definition at line 33 of file UNBL.h.

References accountNumber, address, balance, firstName, fullname, and lastName.

```
00033                                                                                                         :
      BANK() {
00034         this->accountNumber = accountNumber;
00035         this->balance = balance;
00036         this->firstName = firstName;
00037         this->lastName = lastName;
00038         this->address = address;
00039         this->fullname = firstName + " " + lastName;
00040     };
```

4.12.2.3 UNBL::UNBL (std::shared_ptr< BANK > *obj,* int *_version,* int *_unique_id*) [inline]

Definition at line 44 of file UNBL.h.

References accountNumber, address, balance, firstName, fullname, lastName, and UNBL().

```
00044                                                                  : BANK(_version, _unique_id) {
00045 
00046         this->accountNumber = obj->GetAccountNumber();
00047         this->balance = obj->GetBalance();
00048         this->firstName = obj->GetFirstName();
00049         this->lastName = obj->GetLastName();
00050         this->address = obj->GetAddress();
00051         this->fullname = obj->GetFirstName() + " " + obj->GetLastName();
00052     };
```

Here is the call graph for this function:

4.12.2.4 UNBL::UNBL (const **UNBL** & *orig*)

Definition at line 11 of file UNBL.cpp.

```
00011                             {
00012 }
```

4.12.2.5 UNBL::∼UNBL ( ) [virtual]

Definition at line 14 of file UNBL.cpp.

Referenced by operator=().

```
00014               {
00015 }
```

4.12.3 Member Function Documentation

4.12.3.1 void UNBL::copy (std::shared_ptr< OSTM > *from*, std::shared_ptr< OSTM > *to*) [virtual]

OSTM required virtual method for deep copy.

Reimplemented from OSTM.

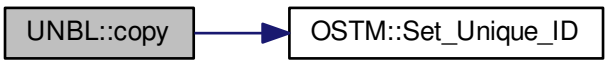
Definition at line 33 of file UNBL.cpp.

References OSTM::Set_Unique_ID().

Referenced by operator=().

```
00033                                                                   {
00034
00035     std::shared_ptr<UNBL> objTO = std::dynamic_pointer_cast<UNBL>(to);
00036     std::shared_ptr<UNBL> objFROM = std::dynamic_pointer_cast<UNBL>(from);
00037     objTO->Set_Unique_ID(objFROM->Get_Unique_ID());
00038     objTO->Set_Version(objFROM->Get_Version());
00039     objTO->SetAccountNumber(objFROM->GetAccountNumber());
00040     objTO->SetBalance(objFROM->GetBalance());
00041
00042 }
```

Here is the call graph for this function:



4.12.3.2 int UNBL::GetAccountNumber ( ) const [virtual]

Reimplemented from BANK.

Definition at line 75 of file UNBL.cpp.

References accountNumber.

Referenced by operator=(), and toString().

```
00075                                   {
00076     return accountNumber;
00077 }
```

4.12.3.3 std::string UNBL::GetAddress ( ) const [virtual]

Reimplemented from BANK.

Definition at line 59 of file UNBL.cpp.

References address.

Referenced by operator=().

```
00059                                   {
00060     return address;
00061 }
```

4.12.3.4 double UNBL::GetBalance ( ) const [virtual]

Reimplemented from BANK.

Definition at line 67 of file UNBL.cpp.

References balance.

Referenced by operator=(), and toString().

```
00067                                 {
00068     return balance;
00069 }
```

4.12.3.5 std::shared_ptr< OSTM > UNBL::getBaseCopy (std::shared_ptr< OSTM > *object*) [virtual]

OSTM required virtual method for returning a pointer that is copy of the original pointer.

Reimplemented from OSTM.

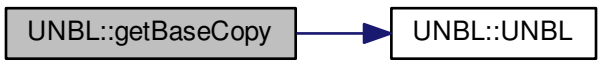
Definition at line 21 of file UNBL.cpp.

References UNBL().

Referenced by operator=().

```
00022 {
00023     std::shared_ptr<BANK> objTO = std::dynamic_pointer_cast<BANK>(object);
00024     std::shared_ptr<BANK> obj(new UNBL(objTO,object->Get_Version(),object->Get_Unique_ID()));
00025     std::shared_ptr<OSTM> ostm_obj = std::dynamic_pointer_cast<OSTM>(obj);

00026     return ostm_obj;
00027 }
```

Here is the call graph for this function:



4.12.3.6 std::string UNBL::GetFirstName ( ) const [virtual]

Reimplemented from BANK.

Definition at line 91 of file UNBL.cpp.

References firstName.

Referenced by operator=(), and toString().

```
00091                                   {
00092     return firstName;
00093 }
```

4.12.3.7 std::string UNBL::GetFullname ( ) const [virtual]

Reimplemented from BANK.

Definition at line 99 of file UNBL.cpp.

References fullname.

Referenced by operator=().

```
00099                                   {
00100     return fullname;
00101 }
```

4.12.3.8 std::string UNBL::GetLastName ( ) const [virtual]

Reimplemented from BANK.

Definition at line 83 of file UNBL.cpp.

References lastName.

Referenced by operator=(), and toString().

```
00083                                   {
00084     return lastName;
00085 }
```

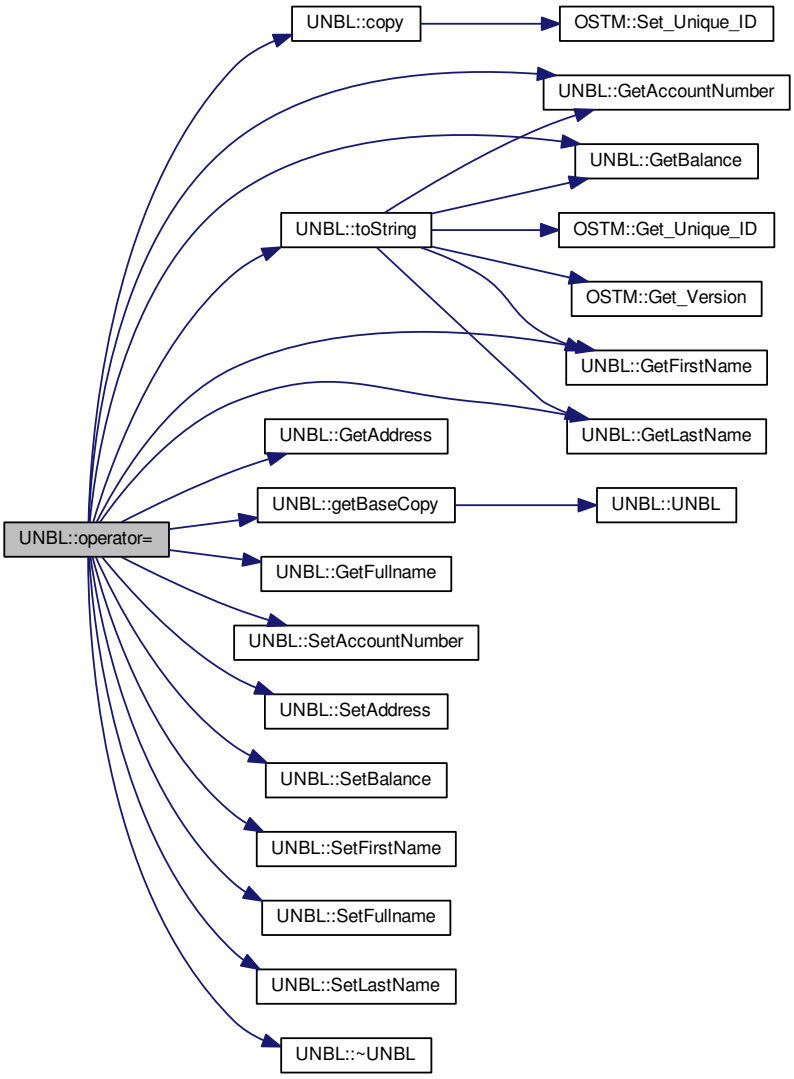
4.12.3.9 UNBL UNBL::operator= (const UNBL & *orig*) [inline]

Definition at line 60 of file UNBL.h.

References accountNumber, address, balance, copy(), firstName, fullname, GetAccountNumber(), GetAddress(), GetBalance(), getBaseCopy(), GetFirstName(), GetFullname(), GetLastName(), lastName, SetAccountNumber(), SetAddress(), SetBalance(), SetFirstName(), SetFullname(), SetLastName(), toString(), and ~UNBL().

```
00060 {};
```

Here is the call graph for this function:



4.12.3.10 void UNBL::SetAccountNumber (int *accountNumber*) [virtual]

Reimplemented from BANK.

Definition at line 71 of file UNBL.cpp.

References accountNumber.

Referenced by operator=().

```
00071                                           {
00072     this->accountNumber = accountNumber;
00073 }
```

4.12.3.11 void UNBL::SetAddress (std::string *address*) [virtual]

Reimplemented from BANK.

Definition at line 55 of file UNBL.cpp.

References address.

Referenced by operator=().

```
00055                                       {
00056     this->address = address;
00057 }
```

4.12.3.12 void UNBL::SetBalance (double *balance*) [virtual]

Reimplemented from BANK.

Definition at line 63 of file UNBL.cpp.

References balance.

Referenced by operator=().

```
00063                                     {
00064     this->balance = balance;
00065 }
```

4.12.3.13 void UNBL::SetFirstName (std::string *firstName*) [virtual]

Reimplemented from BANK.

Definition at line 87 of file UNBL.cpp.

References firstName.

Referenced by operator=().

```
00087                                           {
00088     this->firstName = firstName;
00089 }
```

4.12.3.14 void UNBL::SetFullname (std::string *fullname*) [virtual]

Reimplemented from BANK.

Definition at line 95 of file UNBL.cpp.

References fullname.

Referenced by operator=().

```
00095                                         {
00096     this->fullname = fullname;
00097 }
```

4.12.3.15 void UNBL::SetLastName (std::string *lastName*) [virtual]

Reimplemented from BANK.

Definition at line 79 of file UNBL.cpp.

References lastName.

Referenced by operator=().

```
00079                                         {
00080     this->lastName = lastName;
00081 }
```

4.12.3.16 void UNBL::toString ( ) [virtual]

OSTM required virtual method for display object.

Reimplemented from OSTM.

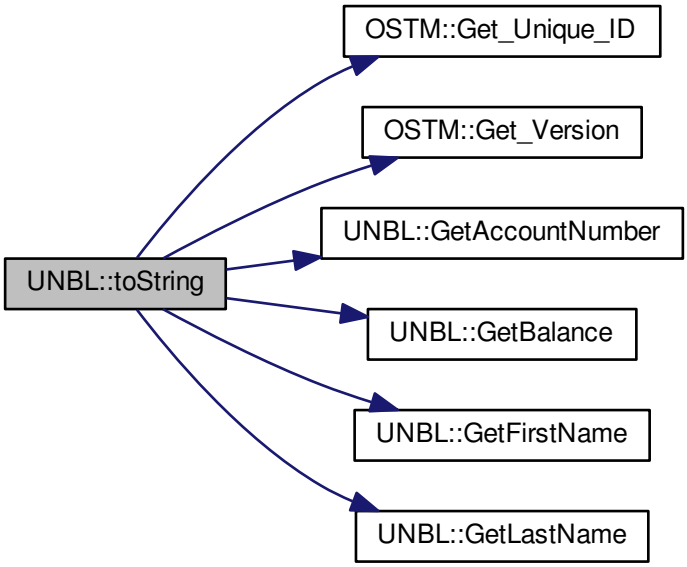
Definition at line 50 of file UNBL.cpp.

References OSTM::Get_Unique_ID(), OSTM::Get_Version(), GetAccountNumber(), GetBalance(), GetFirstName(), and GetLastName().

Referenced by operator=().

```
00051 {
00052    std::cout << "\nUNBL BANK" << "\nUnique ID : " << this->Get_Unique_ID() << "\nInt account :
       " << this->GetAccountNumber() << "\nDouble value : " << this->
      GetBalance() << "\nFirst name: " << this->GetFirstName() << "\nLast name : " <<
      this->GetLastName()  << "\nVersion number : " << this->Get_Version() << std::endl;
00053 }
```

Here is the call graph for this function:



4.12.4 Member Data Documentation

4.12.4.1 int UNBL::accountNumber [private]

Definition at line 93 of file UNBL.h.

Referenced by GetAccountNumber(), operator=(), SetAccountNumber(), and UNBL().

4.12.4.2 std::string UNBL::address [private]

Definition at line 95 of file UNBL.h.

Referenced by GetAddress(), operator=(), SetAddress(), and UNBL().

4.12.4.3 double UNBL::balance [private]

Definition at line 94 of file UNBL.h.

Referenced by GetBalance(), operator=(), SetBalance(), and UNBL().

4.12.4.4 std::string UNBL::firstName [private]

Definition at line 91 of file UNBL.h.

Referenced by GetFirstName(), operator=(), SetFirstName(), and UNBL().

4.12.4.5 std::string UNBL::fullname [private]

Definition at line 90 of file UNBL.h.

Referenced by GetFullname(), operator=(), SetFullname(), and UNBL().

4.12.4.6 std::string UNBL::lastName [private]

Definition at line 92 of file UNBL.h.

Referenced by GetLastName(), operator=(), SetLastName(), and UNBL().

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

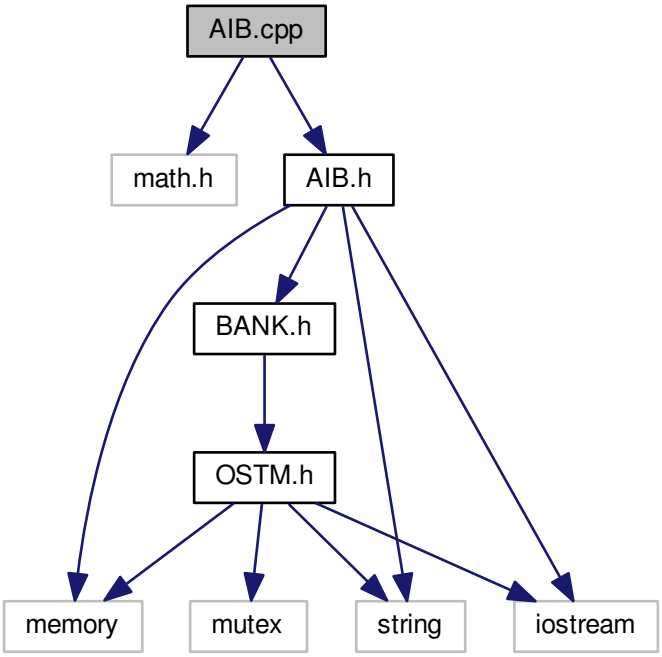
- UNBL.h
- UNBL.cpp

5 File Documentation

5.1 AIB.cpp File Reference

```
#include <math.h>
#include "AIB.h"
```

Include dependency graph for AIB.cpp:



5.2 AIB.cpp

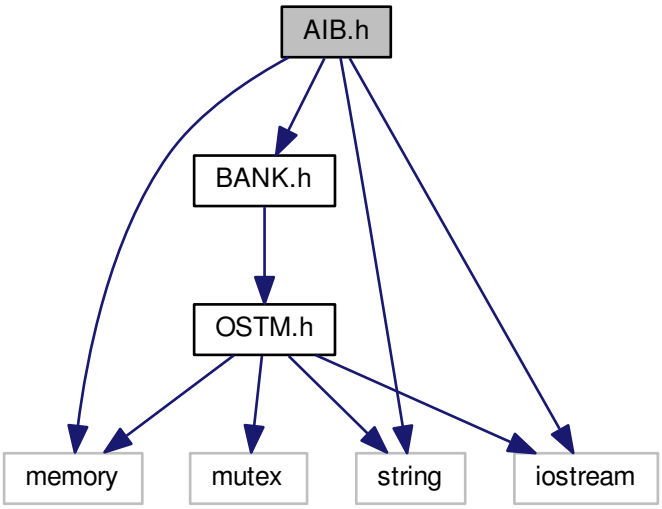
```
/*
 * File:   AIB.cpp
 * Author: Zoltan Fuzesi
 * IT Carlow : C00197361
 *
 * Created on January 17, 2018, 8:02 PM
 */

#include <math.h>

#include "AIB.h"


AIB::AIB(const AIB& orig) {
}

AIB::~AIB() {
}
/*
 * \brief getBaseCopy function, make deep copy of the object/pointer and Return a new std::shared_ptr<BANK> type object
 * \param objTO is a BANK type pointer for casting
 * \param obj is a std::shared_ptr<BANK> return type
 */
std::shared_ptr<OSTM> AIB::getBaseCopy(std::shared_ptr<OSTM> object)
{

    std::shared_ptr<BANK> objTO = std::dynamic_pointer_cast<BANK>(object);
    std::shared_ptr<BANK> obj(new AIB(objTO, object->Get_Version(),object->Get_Unique_ID()));
    std::shared_ptr<OSTM> ostm_obj = std::dynamic_pointer_cast<OSTM>(obj);
    return ostm_obj;
}
/*
 * \brief copy function, make deep copy of the object/pointer
 * \param objTO is a std::shared_ptr<BANK> type object casted back from std::shared_ptr<OSTM>
 * \param objFROM is a std::shared_ptr<BANK> type object casted back from std::shared_ptr<OSTM>
 */
void AIB::copy(std::shared_ptr<OSTM> to, std::shared_ptr<OSTM> from){

    std::shared_ptr<AIB> objTO = std::dynamic_pointer_cast<AIB>(to);
    std::shared_ptr<AIB> objFROM = std::dynamic_pointer_cast<AIB>(from);
    objTO->Set_Unique_ID(objFROM->Get_Unique_ID());
    objTO->Set_Version(objFROM->Get_Version());
    objTO->SetAccountNumber(objFROM->GetAccountNumber());
    objTO->SetBalance(objFROM->GetBalance());
}

/*
 *  \brief toString function, displays the object values in formatted way
 */
void AIB::toString()
{
    std::cout << "\nAIB BANK" << "\nUnique ID : " << this->Get_Unique_ID() << "\nInt account : " << this->GetAccountNumber() << "\nDouble value : " << this->GetBalance() << "\nFirst name: " << this->GetFirstName() << "\nLast name : " << this->GetLastName()  << "\nVersion number : " << this->Get_Version() << std::endl;
}

void AIB::SetAddress(std::string address) {
    this->address = address;
}

std::string AIB::GetAddress() const {
    return address;
}

void AIB::SetBalance(double balance) {
    this->balance = balance;
}

double AIB::GetBalance() const {
    return balance;
}

void AIB::SetAccountNumber(int accountNumber) {
    this->accountNumber = accountNumber;
}

int AIB::GetAccountNumber() const {
    return accountNumber;
}

void AIB::SetLastName(std::string lastName) {
    this->lastName = lastName;
```

```
}

std::string AIB::GetLastName() const {
    return lastName;
}

void AIB::SetFirstName(std::string firstName) {
    this->firstName = firstName;
}

std::string AIB::GetFirstName() const {
    return firstName;
}

void AIB::SetFullname(std::string fullname) {
    this->fullname = fullname;
}

std::string AIB::GetFullname() const {
    return fullname;
}
```

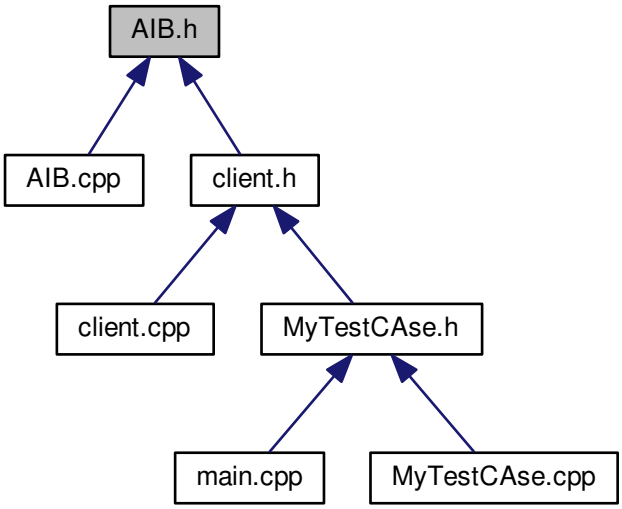
5.3 AIB.h File Reference

```
#include "BANK.h"
#include <string>
#include <memory>
#include <iostream>
```

Include dependency graph for AIB.h:



This graph shows which files directly or indirectly include this file:



Classes

- class AIB

5.4 AIB.h

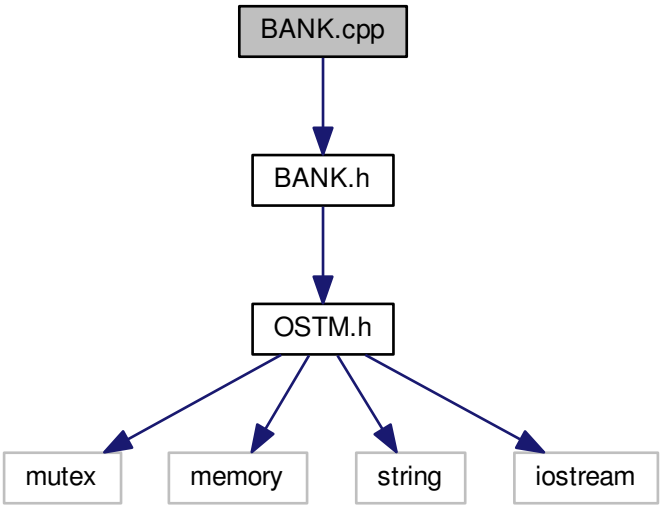
```
00001 /*
00002  * File:   AIB.h
00003  * Author: Zoltan Fuzesi
00004  * IT Carlow : C00197361
00005  *
00006  * Created on January 17, 2018, 8:02 PM
00007  */
00008
00009 #ifndef AIB_H
00010 #define AIB_H
00011 #include "BANK.h"
00012 #include <string>
00013 #include <memory>
00014 #include <iostream>
00015 /*
00016  * Inherit from BANK
00017  */
00018 class AIB : public BANK {
00019 public:
00020     /*
00021      * Constructor
00022      */
00023     AIB(): BANK()
00024     {
00025         this->accountNumber = 0;
00026         this->balance = 50;
00027         this->firstName = "Joe";
00028         this->lastName = "Blog";
00029         this->address = "High street, Carlow";
00030         this->fullname = firstName + " " + lastName;
00031
00032     };
00033     /*
00034      * Custom constructor
```

```
     */
    AIB(int accountNumber, double balance, std::string firstName, std::string lastName, std::string address): BANK()
    {
        this->accountNumber = accountNumber;
        this->balance = balance;
        this->firstName = firstName;
        this->lastName = lastName;
        this->address = address;
        this->fullname = firstName + " " + lastName;
    };
    /*
     * Custom constructor, used by the library for deep copying
     */
    AIB(std::shared_ptr<BANK> obj, int _version, int _unique_id): BANK(_version, _unique_id)
    {

        this->accountNumber = obj->GetAccountNumber();
        this->balance = obj->GetBalance();
        this->firstName = obj->GetFirstName();
        this->lastName = obj->GetLastName();
        this->address = obj->GetAddress();
        this->fullname = obj->GetFirstName() + " " + obj->GetLastName();

    };
    /*
     * Copy constructor
     */
    AIB(const AIB& orig);
    /*
     * Operator
     */
    AIB operator=(const AIB& orig){};
    /*
     * de-constructor
     */
    virtual ~AIB();

    /*
     * Implement OSTM virtual methods
     */
    virtual void copy(std::shared_ptr<OSTM> to, std::shared_ptr<OSTM> from);
    virtual std::shared_ptr<OSTM> getBaseCopy(std::shared_ptr<OSTM> object);
    virtual void toString();

    /*
     * Implement BANK virtual methods
     */
    virtual void SetAddress(std::string address);
    virtual std::string GetAddress() const;
    virtual void SetBalance(double balance);
    virtual double GetBalance() const;
    virtual void SetAccountNumber(int accountNumber);
    virtual int GetAccountNumber() const;
    virtual void SetLastName(std::string lastName);
    virtual std::string GetLastName() const;
    virtual void SetFirstName(std::string firstName);
    virtual std::string GetFirstName() const;
    virtual void SetFullname(std::string fullname);
    virtual std::string GetFullname() const;

private:
    std::string fullname;
    std::string firstName;
    std::string lastName;
    int accountNumber;
    double balance;
    std::string address;


};

#endif /* AIB_H */
```

5.5 BANK.cpp File Reference

```
#include "BANK.h"
```

Include dependency graph for BANK.cpp:



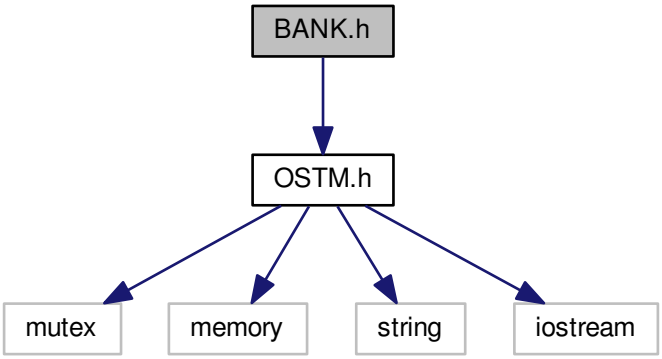
5.6 BANK.cpp

```
00001 /*
00002  * File:   BANK.cpp
00003  * Author: Zoltan Fuzesi
00004  * IT Carlow : C00197361
00005  *
00006  * Created on January 17, 2018, 8:02 PM
00007  */
00008
00009 #include "BANK.h"
00010
00011 BANK::BANK(const BANK& orig) {
00012 }
00013
00014 BANK::~BANK() {
00015 }
00016
```

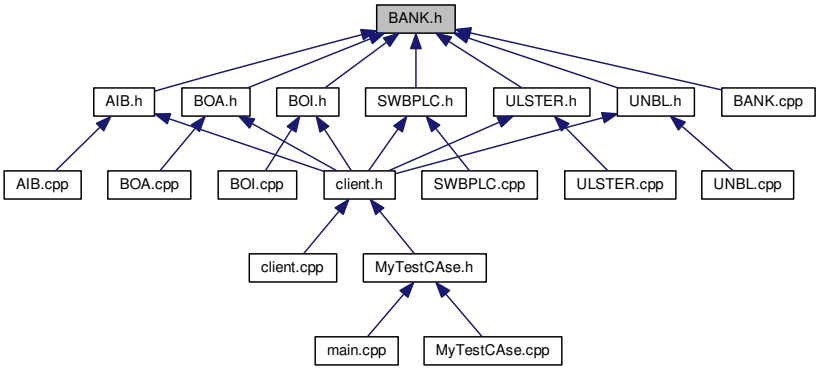
5.7 BANK.h File Reference

```
#include "OSTM.h"
```

Include dependency graph for BANK.h:



This graph shows which files directly or indirectly include this file:



Classes

- class BANK

5.8 BANK.h

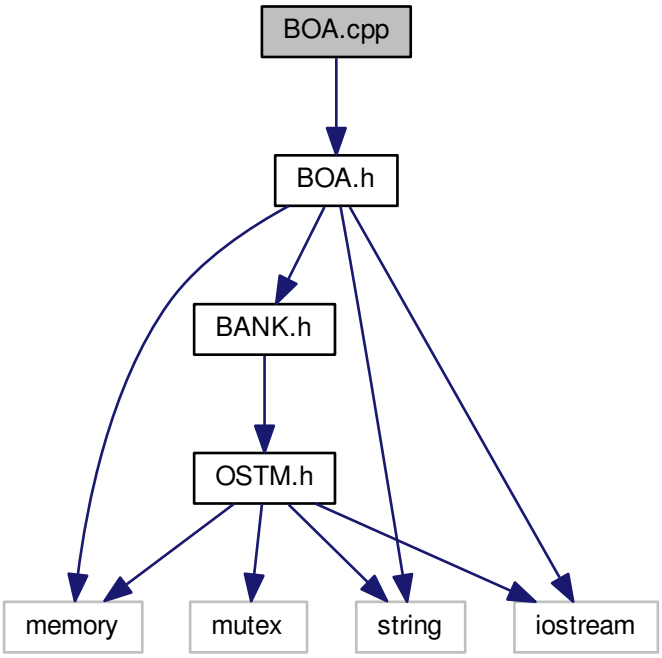
```
/*
 * File:   BANK.h
 * Author: Zoltan Fuzesi
 * IT Carlow : C00197361
 *
 * Created on January 17, 2018, 8:02 PM
 */

#ifndef BANK_H
#define BANK_H
#include "OSTM.h"
```

```
/*
 * BANK inherit from the OSTM library. It is declares the common functions in the child classes as a virtual  function.
 *
 */
class BANK : public OSTM {


public:
    /*
     * Constructor
     */
    BANK(): OSTM(){

    };
    /*
     * Custom Constructor
     */
    BANK(int _version, int _unique_id) : OSTM(_version, _unique_id){

    };
    /*
     * Copy constructor
     */
    BANK(const BANK& orig);
    /*
     * de-constructor
     */
    virtual ~BANK();

    /*
     * Bank specific virtual functions
     */
    virtual void SetAddress(std::string address){};
    virtual std::string GetAddress() const{};
    virtual void SetBalance(double balance){};
    virtual double GetBalance() const{};
    virtual void SetAccountNumber(int accountNumber){};
    virtual int GetAccountNumber() const{};
    virtual void SetLastName(std::string lastName){};
    virtual std::string GetLastName() const{};
    virtual void SetFirstName(std::string firstName){};
    virtual std::string GetFirstName() const{};
    virtual void SetFullname(std::string fullname){};
    virtual std::string GetFullname() const{};

private:

};

#endif /* BANK_H */

```

5.9 BOA.cpp File Reference

```
#include "BOA.h"
```

Include dependency graph for BOA.cpp:



5.10 BOA.cpp

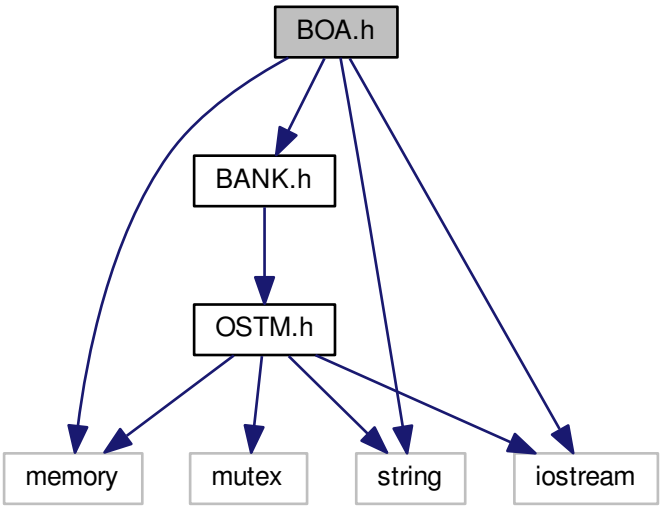
```
00001 /*
00002  * File:   BOA.cpp
00003  * Author: Zoltan Fuzesi
00004  * IT Carlow : C00197361
00005  *
00006  * Created on January 17, 2018, 8:02 PM
00007  */
00008
00009 #include "BOA.h"
00010
00011
00012 BOA::BOA(const BOA& orig) {
00013 }
00014
00015 BOA::~BOA() {
00016 }
00017 /*
00018  * \brief getBaseCopy function, make deep copy of the object/pointer and Return a new std::shared_ptr<BANK>
       type object
00019  * \param objTO is a BANK type pointer for casting
00020  * \param obj is a std::shared_ptr<BANK> return type
00021  */
00022 std::shared_ptr<OSTM> BOA::getBaseCopy(std::shared_ptr<OSTM> object)
00023 {
00024         std::shared_ptr<BANK> objTO = std::dynamic_pointer_cast<BANK>(object);
00025     std::shared_ptr<BANK> obj(new BOA(objTO,object->Get_Version(),object->Get_Unique_ID()));
00026         std::shared_ptr<OSTM> ostm_obj = std::dynamic_pointer_cast<OSTM>(obj);
00027     return ostm_obj;
00028 }
00029 /*
00030  * \brief copy function, make deep copy of the object/pointer
00031  * \param objTO is a std::shared_ptr<BANK> type object casted back from std::shared_ptr<OSTM>
00032  * \param objFROM is a std::shared_ptr<BANK> type object casted back from std::shared_ptr<OSTM>
00033  */
00034 void BOA::copy(std::shared_ptr<OSTM> to, std::shared_ptr<OSTM> from){
```

```
00035
00036     std::shared_ptr<BOA> objTO = std::dynamic_pointer_cast<BOA>(to);
00037     std::shared_ptr<BOA> objFROM = std::dynamic_pointer_cast<BOA>(from);
00038     objTO->Set_Unique_ID(objFROM->Get_Unique_ID());
00039     objTO->Set_Version(objFROM->Get_Version());
00040     objTO->SetAccountNumber(objFROM->GetAccountNumber());
00041     objTO->SetBalance(objFROM->GetBalance());
00042
00043 }
00044
00045 /*
00046  *  \brief toString function, displays the object values in formatted way
00047  */
00048 void BOA::toString()
00049 {
00050    // std::cout << "\nUnique ID : " << this->GetUniqueID() << "\nInt value : " << this->GetV_int() <<
       "\nDouble value : " << this->GetV_double() << "\nFloat value : " << this->GetV_float() << "\nString value : " <<
       this->GetV_string()  << "\nVersion number : " << this->GetVersion() << "\nLoad Counter : "<<
       this->GetLoadCounter() << "\nWrite Counter : "<< this->GetWriteCounter() << std::endl;
00051      std::cout << "\nBOA BANK" << "\nUnique ID : " << this->Get_Unique_ID() << "\nInt account
       : " << this->GetAccountNumber() << "\nDouble value : " << this->
      GetBalance() << "\nFirst name: " << this->GetFirstName() << "\nLast name : " <<
      this->GetLastName()  << "\nVersion number : " << this->Get_Version() << std::endl;
00052 }
00053
00054 void BOA::SetAddress(std::string address) {
00055     this->address = address;
00056 }
00057
00058 std::string BOA::GetAddress() const {
00059     return address;
00060 }
00061
00062 void BOA::SetBalance(double balance) {
00063     this->balance = balance;
00064 }
00065
00066 double BOA::GetBalance() const {
00067     return balance;
00068 }
00069
00070 void BOA::SetAccountNumber(int accountNumber) {
00071     this->accountNumber = accountNumber;
00072 }
00073
00074 int BOA::GetAccountNumber() const {
00075     return accountNumber;
00076 }
00077
00078 void BOA::SetLastName(std::string lastName) {
00079     this->lastName = lastName;
00080 }
00081
00082 std::string BOA::GetLastName() const {
00083     return lastName;
00084 }
00085
00086 void BOA::SetFirstName(std::string firstName) {
00087     this->firstName = firstName;
00088 }
00089
00090 std::string BOA::GetFirstName() const {
00091     return firstName;
00092 }
00093
00094 void BOA::SetFullname(std::string fullname) {
00095     this->fullname = fullname;
00096 }
00097
00098 std::string BOA::GetFullname() const {
00099     return fullname;
00100 }
00101
```

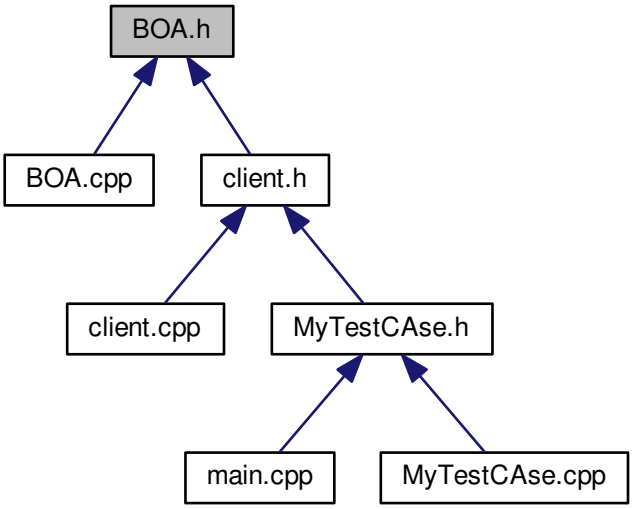
5.11 BOA.h File Reference

```
#include "BANK.h"
#include <string>
#include <memory>
#include <iostream>
```

Include dependency graph for BOA.h:



This graph shows which files directly or indirectly include this file:



Classes

- class BOA

5.12 BOA.h

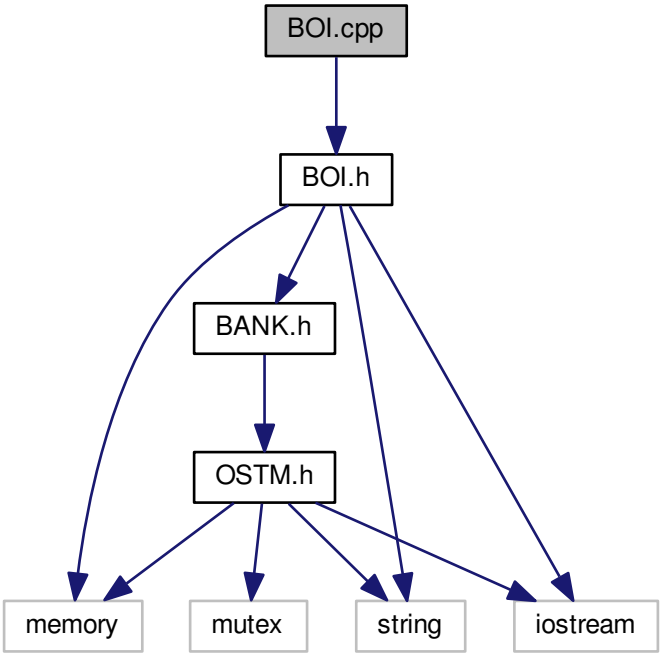
```
/*
 * File:   BOA.h
 * Author: Zoltan Fuzesi
 * IT Carlow : C00197361
 *
 * Created on January 17, 2018, 8:02 PM
 */

#ifndef BOA_H
#define BOA_H
#include "BANK.h"
#include <string>
#include <memory>
#include <iostream>
/*
 * Inherit from BANK
 */
class BOA : public BANK {
public:

    /*
     * Constructor
     */
    BOA() : BANK() {
        this->accountNumber = 0;
        this->balance = 50;
        this->firstName = "Joe";
        this->lastName = "Blog";
        this->address = "High street, Carlow";
        this->fullname = firstName + " " + lastName;
    };
    /*
     * Custom constructor
     */
    BOA(int accountNumber, double balance, std::string firstName, std::string lastName, std::string address) : BANK() {
        this->accountNumber = accountNumber;
        this->balance = balance;
        this->firstName = firstName;
        this->lastName = lastName;
        this->address = address;
        this->fullname = firstName + " " + lastName;
    };
    /*
     * Custom constructor, used by the library for deep copying
     */
    BOA(std::shared_ptr<BANK> obj, int _version, int _unique_id) : BANK(_version, _unique_id) {

        this->accountNumber = obj->GetAccountNumber();
        this->balance = obj->GetBalance();
        this->firstName = obj->GetFirstName();
        this->lastName = obj->GetLastName();
        this->address = obj->GetAddress();
        this->fullname = obj->GetFirstName() + " " + obj->GetLastName();
    };


    /*
     * Copy constructor
     */
    BOA(const BOA& orig);
    /*
     * Operator
     */
    BOA operator=(const BOA& orig) {
    };
    /*
     * de-constructor
     */
    virtual ~BOA();

    /*
     * Implement OSTM virtual methods
     */

    virtual void copy(std::shared_ptr<OSTM> to, std::shared_ptr<OSTM> from);
    virtual std::shared_ptr<OSTM> getBaseCopy(std::shared_ptr<OSTM> object);
    virtual void toString();

    /*
     * Implement BANK virtual methods
     */
    virtual void SetAddress(std::string address);
```

```
    virtual std::string GetAddress() const;
    virtual void SetBalance(double balance);
    virtual double GetBalance() const;
    virtual void SetAccountNumber(int accountNumber);
    virtual int GetAccountNumber() const;
    virtual void SetLastName(std::string lastName);
    virtual std::string GetLastName() const;
    virtual void SetFirstName(std::string firstName);
    virtual std::string GetFirstName() const;
    virtual void SetFullname(std::string fullname);
    virtual std::string GetFullname() const;
private:
    std::string fullname;
    std::string firstName;
    std::string lastName;
    int accountNumber;
    double balance;
    std::string address;

};

#endif /* BOA_H */

```

5.13 BOI.cpp File Reference

`#include "BOI.h"`

Include dependency graph for BOI.cpp:



5.14 BOI.cpp

```

/*
 * File:   BOI.cpp
```

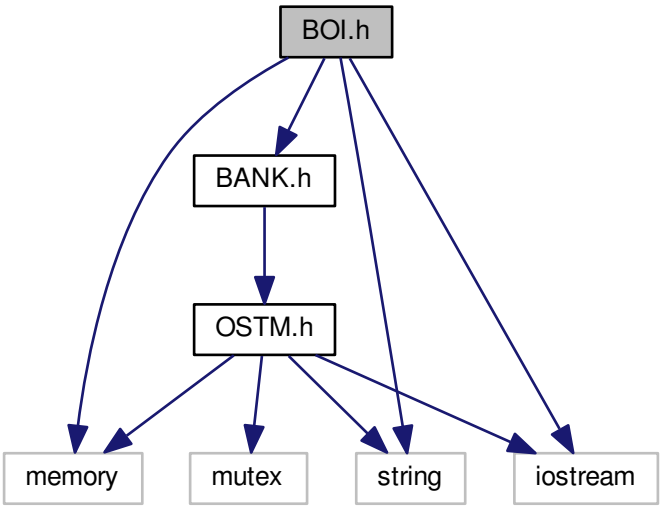
```
 * Author: Zoltan Fuzesi
 * IT Carlow : C00197361
 *
 * Created on January 17, 2018, 8:02 PM
 */

#include "BOI.h"

BOI::~BOI() {
}

BOI::BOI(const BOI& orig) {
}
/*
 * \brief getBaseCopy function, make deep copy of the object/pointer and Return a new BANK* type object
 * \param objTO is a BANK type pointer for casting
 * \param obj is a BANK* return type
 */
std::shared_ptr<OSTM> BOI::getBaseCopy(std::shared_ptr<OSTM> object)
{

    std::shared_ptr<BOI> objTO = std::dynamic_pointer_cast<BOI>(object);
    std::shared_ptr<BOI> obj(new BOI(objTO,object->Get_Version(),object->Get_Unique_ID()));
    std::shared_ptr<OSTM> ostm_obj = std::dynamic_pointer_cast<OSTM>(obj);
    return ostm_obj;
}
/*
 * \brief copy function, make deep copy of the object/pointer
 * \param objTO is a BANK* type object casted back from std::shared_ptr<OSTM>
 * \param objFROM is a BANK* type object casted back from std::shared_ptr<OSTM>
 */
void BOI::copy(std::shared_ptr<OSTM> to, std::shared_ptr<OSTM> from){

    std::shared_ptr<BOI> objTO = std::dynamic_pointer_cast<BOI>(to);
    std::shared_ptr<BOI> objFROM = std::dynamic_pointer_cast<BOI>(from);
    objTO->Set_Unique_ID(objFROM->Get_Unique_ID());
    objTO->Set_Version(objFROM->Get_Version());
    objTO->SetAccountNumber(objFROM->GetAccountNumber());
    objTO->SetBalance(objFROM->GetBalance());
}


/*
 *  \brief toString function, displays the object values in formatted way
 */
void BOI::toString()
{
   std::cout << "\nBOI BANK" << "\nUnique ID : " << this->Get_Unique_ID() << "\nInt account : " << this->GetAccountNumber() << "\nDouble value : " << this->GetBalance() << "\nFirst name: " << this->GetFirstName() << "\nLast name : " << this->GetLastName()  << "\nVersion number : " << this->Get_Version() << std::endl;
}
void BOI::SetAddress(std::string address) {
    this->address = address;
}

std::string BOI::GetAddress() const {
    return address;
}

void BOI::SetBalance(double balance) {
    this->balance = balance;
}

double BOI::GetBalance() const {
    return balance;
}

void BOI::SetAccountNumber(int accountNumber) {
    this->accountNumber = accountNumber;
}

int BOI::GetAccountNumber() const {
    return accountNumber;
}

void BOI::SetLastName(std::string lastName) {
    this->lastName = lastName;
}

std::string BOI::GetLastName() const {
    return lastName;
}

void BOI::SetFirstName(std::string firstName) {
    this->firstName = firstName;
}
```

```
std::string BOI::GetFirstName() const {
    return firstName;
}

void BOI::SetFullname(std::string fullname) {
    this->fullname = fullname;
}

std::string BOI::GetFullname() const {
    return fullname;
}
```

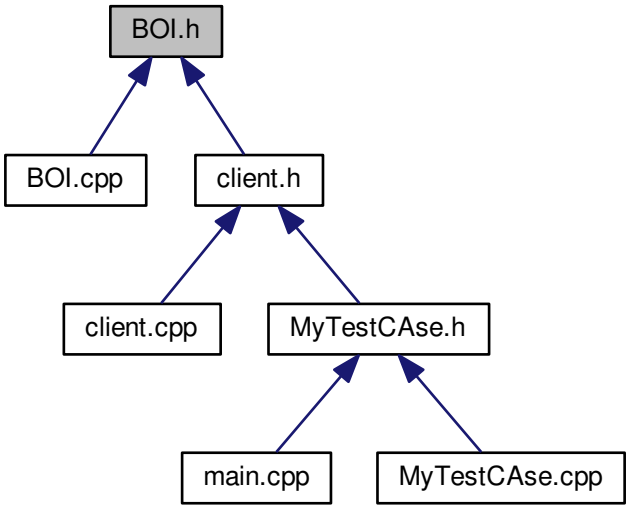
5.15 BOI.h File Reference

```
#include "BANK.h"
#include <string>
#include <memory>
#include <iostream>
```

Include dependency graph for BOI.h:



This graph shows which files directly or indirectly include this file:



Classes

- class BOI

5.16 BOI.h

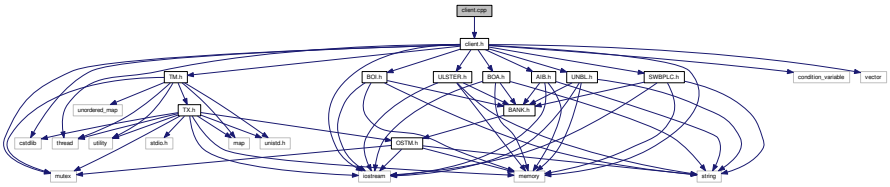
```
00001
00002 /*
00003  * File:   BOI.h
00004  * Author: Zoltan Fuzesi
00005  * IT Carlow : C00197361
00006  *
00007  * Created on January 17, 2018, 8:02 PM
00008  */
00009
00010 #ifndef BOI_H
00011 #define BOI_H
00012 #include "BANK.h"
00013 #include <string>
00014 #include <memory>
00015 #include <iostream>
00016 /*
00017  * Inherit from BANK
00018  */
00019 class BOI: public BANK {
00020 public:
00021     /*
00022      * Constructor
00023      */
00024     BOI(): BANK()
00025     {
00026         this->accountNumber = 0;
00027         this->balance = 50;
00028         this->firstName = "Joe";
00029         this->lastName = "Blog";
00030         this->address = "High street, Carlow";
00031         this->fullname = firstName + " " + lastName;
00032
00033     }
00034     /*
```

```
     * Custom constructor
     */
    BOI(int accountNumber, double balance, std::string firstName, std::string lastName, std::string address): BANK()
    {
        this->accountNumber = accountNumber;
        this->balance = balance;
        this->firstName = firstName;
        this->lastName = lastName;
        this->address = address;
        this->fullname = firstName + " " + lastName;
    };
    /*
     * Custom constructor, used by the library for deep copying
     */
    BOI(std::shared_ptr<BOI> obj, int _version, int _unique_id): BANK(_version, _unique_id)
    {
        this->accountNumber = obj->GetAccountNumber();
        this->balance = obj->GetBalance();
        this->firstName = obj->GetFirstName();
        this->lastName = obj->GetLastName();
        this->address = obj->GetAddress();
        this->fullname = obj->GetFirstName() + " " + obj->GetLastName();
    };
    /*
     * Copy constructor
     */
    BOI(const BOI& orig);
    /*
     * Operator
     */
    BOI operator=(const BOI& orig){};
    /*
     * de-constructor
     */
    virtual ~BOI();

    /*
     * Implement OSTM virtual methods
     */

    virtual std::shared_ptr<OSTM> getBaseCopy(std::shared_ptr<OSTM> object);
    virtual void copy(std::shared_ptr<OSTM> to, std::shared_ptr<OSTM> from);
    virtual void toString();

    /*
     * Implement BANK virtual methods
     */
    virtual void SetAddress(std::string address);
    virtual std::string GetAddress() const;
    virtual void SetBalance(double balance);
    virtual double GetBalance() const;
    virtual void SetAccountNumber(int accountNumber);
    virtual int GetAccountNumber() const;
    virtual void SetLastName(std::string lastName);
    virtual std::string GetLastName() const;
    virtual void SetFirstName(std::string firstName);
    virtual std::string GetFirstName() const;
    virtual void SetFullname(std::string fullname);
    virtual std::string GetFullname() const;

private:
    std::string fullname;
    std::string firstName;
    std::string lastName;
    int accountNumber;
    double balance;
    std::string address;

};

#endif /* BOI_H */
```

5.17 client.cpp File Reference

```
#include "client.h"
```

Include dependency graph for client.cpp:



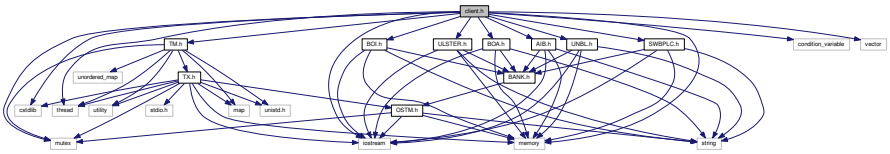
5.18 client.cpp

```
00001
00002
00003 #include "client.h"
```

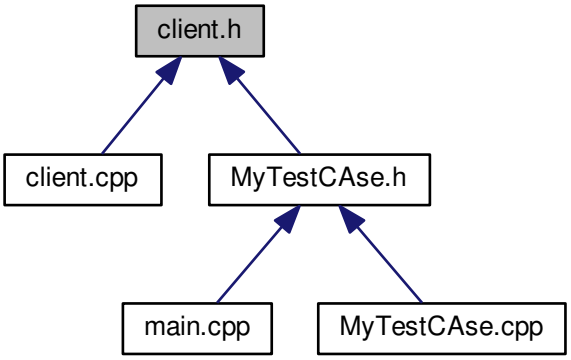
5.19 client.h File Reference

```
#include <cstdlib>
#include <iostream>
#include <thread>
#include "TM.h"
#include "AIB.h"
#include "BOI.h"
#include "BOA.h"
#include "SWBPLC.h"
#include "ULSTER.h"
#include "UNBL.h"
#include <mutex>
#include <memory>
#include <condition_variable>
#include <vector>
```

Include dependency graph for client.h:



This graph shows which files directly or indirectly include this file:



Classes

- class client

Macros

- #define CLIENT_H

5.19.1 Macro Definition Documentation

5.19.1.1 #define CLIENT_H

Definition at line 32 of file client.h.

5.20 client.h

```
00001 /*
00002  * To change this license header, choose License Headers in Project Properties.
00003  * To change this template file, choose Tools | Templates
00004  * and open the template in the editor.
00005  */
00006
00007 /*
00008  * File:   main.cpp
00009  * Author: zoltan
00010  *
00011  * Created on November 27, 2017, 9:26 PM
00012  */
00013
00014 #include <cstdlib>
00015 #include <iostream>
00016 #include <thread>
00017
00018 #include "TM.h"
00019 #include "AIB.h"    //Bank Account
00020 #include "BOI.h"    //Bank Account
00021 #include "BOA.h"    //Bank Account
```

```
#include "SWBPLC.h" //Bank Account
#include "ULSTER.h" //Bank Account
#include "UNBL.h"   //Bank Account
#include <mutex>
#include <memory>
#include <condition_variable>
#include <vector>


#ifndef CLIENT_H
#define CLIENT_H

class client {
private:

public:
    int value = 0;
    client(int value){ this->value = value; };

    void _two_account_transfer_(std::shared_ptr<OSTM> _to_, std::shared_ptr<OSTM> _from_, TM& tm, double _amount) {

    std::shared_ptr<TX> tx = tm._get_tx();
    /*
     * Register the two single account
     */
    tx->_register(_to_);
    tx->_register(_from_);
    /*
     * Declare required pointers
     */
    std::shared_ptr<BANK> _TO_BANK_, _FROM_BANK_;
    std::shared_ptr<OSTM> _TO_OSTM_, _FROM_OSTM_;

    bool done = false;
    try {
        while (!done) {
            /*
             * From std::shared_ptr<OSTM> to std::shared_ptr<BANK> to access the virtual methods
             */
            _TO_BANK_ = std::dynamic_pointer_cast<BANK> (tx->load(_to_));
            _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_));
            /*
             * Make changes with the objects
             */
            _TO_BANK_->SetBalance(_TO_BANK_->GetBalance() + _amount);
            _FROM_BANK_->SetBalance(_FROM_BANK_->GetBalance() - _amount);
            /*
             * From std::shared_ptr<BANK> to std::shared_ptr<OSTM> to store the memory spaces
             */
            _TO_OSTM_ = std::dynamic_pointer_cast<OSTM> (_TO_BANK_);
            _FROM_OSTM_ = std::dynamic_pointer_cast<OSTM> (_FROM_BANK_);
            /*
             * Store changes
             */
            tx->store(_TO_OSTM_);
            tx->store(_FROM_OSTM_);

            /*
             * Commit changes
             */
            done = tx->commit();
        }
    } catch (std::runtime_error& e) {
        std::cout << e.what() << std::endl;
    }
}

void _nesting_(std::shared_ptr<OSTM> _to_, std::shared_ptr<OSTM> _from_, TM& tm, double _amount) {
    std::shared_ptr<TX> tx = tm._get_tx();
    /*
     * Register the two single account
     */
    tx->_register(_to_);
    tx->_register(_from_);
    /*
     * Declare required pointers
     */
    std::shared_ptr<BANK> _TO_BANK_, _FROM_BANK_;
    std::shared_ptr<OSTM> _TO_OSTM_, _FROM_OSTM_;


    bool done = false;
    try {
        while (!done) {
            /*
```

```
             * From std::shared_ptr<OSTM> to std::shared_ptr<BANK> to access the virtual methods
             */
            _TO_BANK_ = std::dynamic_pointer_cast<BANK> (tx->load(_to_));
            _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_));
            /*
             * Make changes with the objects
             */
            _TO_BANK_->SetBalance(_TO_BANK_->GetBalance() + _amount);
            _FROM_BANK_->SetBalance(_FROM_BANK_->GetBalance() - _amount);
            /*
             * From std::shared_ptr<BANK> to std::shared_ptr<OSTM> to store the memory spaces
             */
            _TO_OSTM_ = std::dynamic_pointer_cast<OSTM> (_TO_BANK_);
            _FROM_OSTM_ = std::dynamic_pointer_cast<OSTM> (_FROM_BANK_);
            /*
             * Store changes
             */
            tx->store(_TO_OSTM_);
            tx->store(_FROM_OSTM_);

            /*
             * NESTED TRANSACTION
             */
            std::shared_ptr<TX> txTwo = tm._get_tx();

            bool nestedDone = false;
            while (!nestedDone) {
                _TO_BANK_ = std::dynamic_pointer_cast<BANK> (txTwo->load(_to_));
                _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (txTwo->load(_from_));
                /*
                 * Make changes with the objects
                 */
                _TO_BANK_->SetBalance(_TO_BANK_->GetBalance() + _amount);
                _FROM_BANK_->SetBalance(_FROM_BANK_->GetBalance() - _amount);
                /*
                 * From std::shared_ptr<BANK> to std::shared_ptr<OSTM> to store the memory spaces
                 */
                _TO_OSTM_ = std::dynamic_pointer_cast<OSTM> (_TO_BANK_);
                _FROM_OSTM_ = std::dynamic_pointer_cast<OSTM> (_FROM_BANK_);
                /*
                 * Store changes
                 */
                txTwo->store(_TO_OSTM_);
                txTwo->store(_FROM_OSTM_);
                /*
                 * NESTED TRANSACTION IN THE NESTED TRANSACTION
                 * _two_account_transfer_ function call
                 */
                _two_account_transfer_(_to_, _from_, tm, _amount);

                nestedDone = txTwo->commit();
            }

            /*
             * Commit changes
             */
            done = tx->commit();
        }
    } catch (std::runtime_error& e) {
        std::cout << e.what() << std::endl;
    }
}

void _six_account_transfer_(std::shared_ptr<OSTM> _to_, std::shared_ptr<OSTM> _from_one_, std::shared_ptr<OSTM> _from_two_, std::shared_ptr<OSTM> _from_three_, std::shared_ptr<OSTM> _from_four_, std::shared_ptr<OSTM> _from_five_, TM& _tm, double _amount) {
    std::shared_ptr<TX> tx = _tm._get_tx();
    /*
     * Register the two single account
     */
    tx->_register(_to_);
    tx->_register(_from_one_);
    tx->_register(_from_two_);
    tx->_register(_from_three_);
    tx->_register(_from_four_);
    tx->_register(_from_five_);

    /*
     * Required pointers to use in transaction
     */
    std::shared_ptr<OSTM> _TO_OSTM, _FROM_ONE_OSTM, _FROM_TWO_OSTM, _FROM_THREE_OSTM, _FROM_FOUR_OSTM, _FROM_FIVE_OSTM;
    std::shared_ptr<BANK> _TO_, _FROM_ONE_, _FROM_TWO_, _FROM_THREE_, _FROM_FOUR_, _FROM_FIVE_;
    try {
        bool done = false;
        while (!done) {
            /*
```

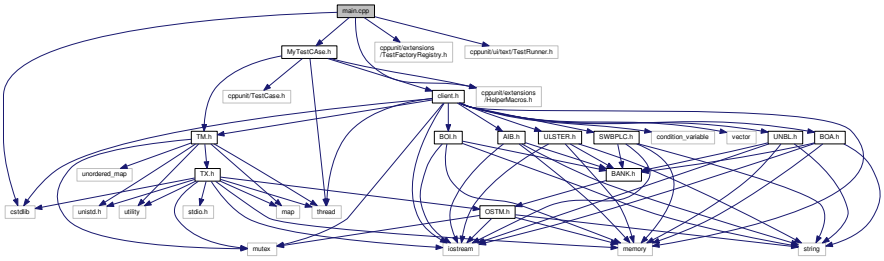
```
             * From std::shared_ptr<OSTM> to std::shared_ptr<BANK> to access the virtual methods
             */
            _TO_ = std::dynamic_pointer_cast<BANK> (tx->load(_to_));
            _FROM_ONE_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_one_));
            _FROM_TWO_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_two_));
            _FROM_THREE_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_three_));
            _FROM_FOUR_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_four_));
            _FROM_FIVE_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_five_));
            /*
             * Make changes with the objects
             */
            _TO_->SetBalance(_TO_->GetBalance() + (_amount * 5));
            _FROM_ONE_->SetBalance(_FROM_ONE_->GetBalance() - _amount);
            _FROM_TWO_->SetBalance(_FROM_TWO_->GetBalance() - _amount);
            _FROM_THREE_->SetBalance(_FROM_THREE_->GetBalance() - _amount);
            _FROM_FOUR_->SetBalance(_FROM_FOUR_->GetBalance() - _amount);
            _FROM_FIVE_->SetBalance(_FROM_FIVE_->GetBalance() - _amount);
            /*
             * From std::shared_ptr<BANK> to std::shared_ptr<OSTM> to store the memory spaces
             */
            _TO_OSTM = std::dynamic_pointer_cast<OSTM> (_TO_);
            _FROM_ONE_OSTM = std::dynamic_pointer_cast<OSTM> (_FROM_ONE_);
            _FROM_TWO_OSTM = std::dynamic_pointer_cast<OSTM> (_FROM_TWO_);
            _FROM_THREE_OSTM = std::dynamic_pointer_cast<OSTM> (_FROM_THREE_);
            _FROM_FOUR_OSTM = std::dynamic_pointer_cast<OSTM> (_FROM_FOUR_);
            _FROM_FIVE_OSTM = std::dynamic_pointer_cast<OSTM> (_FROM_FIVE_);
            /*
             * Store changes
             */
            tx->store(_TO_OSTM);
            tx->store(_FROM_ONE_OSTM);
            tx->store(_FROM_TWO_OSTM);
            tx->store(_FROM_THREE_OSTM);
            tx->store(_FROM_FOUR_OSTM);
            tx->store(_FROM_FIVE_OSTM);
            /*
             * Commit changes
             */
            done = tx->commit();
        }
    } catch (std::runtime_error& e) {
        std::cout << e.what() << std::endl;
    }
}

void _complex_transfer_(std::shared_ptr<OSTM> _from_, std::shared_ptr<OSTM> _from_two_,
    std::vector<std::shared_ptr<OSTM>> _customer_vec, TM& _tm, double _amount) {
    std::shared_ptr<TX> tx = _tm._get_tx();
    /* Register the two single account*/
    tx->_register(_from_);
    tx->_register(_from_two_);
    /* Declare required pointers */
    std::shared_ptr<OSTM> _FROM_OSTM_ONE_, _FROM_OSTM_TWO_, _TO_OSTM_;
    std::shared_ptr<BANK> _FROM_, _FROM_TWO_, _TO_;

    bool done = false;
    try {
        while (!done) {
            for (auto&& obj : _customer_vec) {
                /* Register customers accounts from the collection (vector) */
                tx->_register(obj);
                /* From std::shared_ptr<OSTM> to std::shared_ptr<BANK> to access the virtual methods  */
                _FROM_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_));
                _FROM_TWO_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_two_));
                _TO_ = std::dynamic_pointer_cast<BANK> (tx->load(obj));
                /* Make changes with the objects */
                _FROM_->SetBalance(_FROM_->GetBalance() - _amount);
                _FROM_TWO_->SetBalance(_FROM_TWO_->GetBalance() - _amount);
                _TO_->SetBalance(_TO_->GetBalance() + (_amount * 2));
                /* From std::shared_ptr<BANK> to std::shared_ptr<OSTM> to store the memory spaces */
                _FROM_OSTM_ONE_ = std::dynamic_pointer_cast<OSTM> (_FROM_);
                _FROM_OSTM_TWO_ = std::dynamic_pointer_cast<OSTM> (_FROM_TWO_);
                _TO_OSTM_ = std::dynamic_pointer_cast<OSTM> (_TO_);
                /* Store changes */
                tx->store(_FROM_OSTM_ONE_);
                tx->store(_FROM_OSTM_TWO_);
                tx->store(_TO_OSTM_);
            }
            /* Commit changes */
            done = tx->commit();
        }
    } catch (std::runtime_error& e) {
        std::cout << e.what() << std::endl;
    }
}


```

```
};

#endif /* CLIENT_H */

```

5.21 main.cpp File Reference

```
#include <cstdlib>
#include <cppunit/extensions/TestFactoryRegistry.h>
#include <cppunit/extensions/HelperMacros.h>
#include <cppunit/ui/text/TestRunner.h>
#include "MyTestCAse.h"
```

Include dependency graph for main.cpp:



Functions

- CPPUNIT_TEST_SUITE_REGISTRATION (MyTestCAse)
- int main (int argc, char **argv)

5.21.1 Function Documentation

5.21.1.1 CPPUNIT_TEST_SUITE_REGISTRATION (MyTestCAse)

5.21.1.2 int main (int *argc*, char ** *argv*)

Definition at line 13 of file main.cpp.

```
                                {
    TextUi::TestRunner runner;
    TestFactoryRegistry  &registry = TestFactoryRegistry::getRegistry();
    runner.addTest(registry.makeTest());
    runner.run();
    return (EXIT_SUCCESS);
}
```

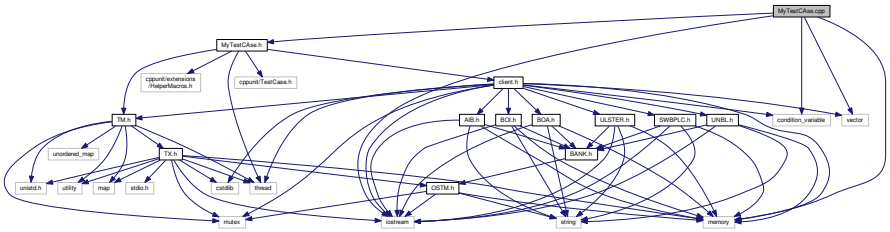
5.22 main.cpp

```
#include <cstdlib>
#include <cppunit/extensions/TestFactoryRegistry.h>
#include <cppunit/extensions/HelperMacros.h>
#include <cppunit/ui/text/TestRunner.h>

#include "MyTestCAse.h"

using namespace std;
using namespace CppUnit;

CPPUNIT_TEST_SUITE_REGISTRATION(MyTestCAse);

int main(int argc, char** argv) {
    TextUi::TestRunner runner;
    TestFactoryRegistry  &registry = TestFactoryRegistry::getRegistry();
    runner.addTest(registry.makeTest());
    runner.run();
    return (EXIT_SUCCESS);
}

```

5.23 MyTestCAse.cpp File Reference

```
#include "MyTestCAse.h"
#include <iostream>
#include <memory>
#include <condition_variable>
#include <vector>
```

Include dependency graph for MyTestCAse.cpp:



5.24 MyTestCAse.cpp

```


#include "MyTestCAse.h"
#include <iostream>
#include <memory>
#include <condition_variable>
#include <vector>

MyTestCAse::MyTestCAse(const MyTestCAse& orig) {
}

MyTestCAse::~MyTestCAse() {
}
void MyTestCAse::_collection_bject_(std::vector<std::shared_ptr<OSTM> >
_customer_vec, TM& _tm, double _amount, int loop){

     std::shared_ptr<TX> tx = _tm._get_tx();

    std::shared_ptr<OSTM> _TO_OSTM_;
    std::shared_ptr<BANK>  _TO_;

    bool done = false;
    try {
        while (!done) {
```

```
00033             for (int i = 0; i < loop; ++i) {
00034                 for (auto&& obj : _customer_vec) {
00038                     // auto&& obj = _customer_vec.at(i);
00039                     tx->_register(obj);
00043                     _TO_ = std::dynamic_pointer_cast<BANK> (tx->load(obj));
00047                     _TO_->SetBalance(_TO_->GetBalance() + (_amount));
00051                     _TO_OSTM_ = std::dynamic_pointer_cast<OSTM> (_TO_);
00055                     tx->store(_TO_OSTM_);
00056                 }
00057             }
00061             done = tx->commit();
00062         }
00063     } catch (std::runtime_error& e) {
00064         std::cout << e.what() << std::endl;
00065     }
00066 }
00067
00074 void MyTestCAse::_one_account_transfer_(std::shared_ptr<OSTM> _to_,
      TM& _tm, double _amount){
00075      std::shared_ptr<TX> tx = tm._get_tx();
00079     tx->_register(_to_);
00080
00081     std::shared_ptr<OSTM> _TO_OSTM_;
00082     std::shared_ptr<BANK> _TO_;
00083
00084     try {
00085         bool done = false;
00086         while (!done) {
00090             _TO_ = std::dynamic_pointer_cast<BANK> (tx->load(_to_));
00094             _TO_->SetBalance(_TO_->GetBalance() + (_amount ));
00095
00099             _TO_OSTM_ = std::dynamic_pointer_cast<OSTM> (_TO_);
00103             tx->store(_TO_OSTM_);
00104
00108             done = tx->commit();
00109         }
00110     } catch (std::runtime_error& e) {
00111         std::cout << e.what() << std::endl;
00112     }
00113
00114
00115
00116 }
00126 void MyTestCAse::_complex_transfer_(std::shared_ptr<OSTM> _from_,
      std::shared_ptr<OSTM> _from_two_, std::vector<std::shared_ptr<OSTM>> _customer_vec, TM& _tm, double _amount) {
00127     std::shared_ptr<TX> tx = _tm._get_tx();
00131     tx->_register(_from_);
00132     tx->_register(_from_two_);
00136     std::shared_ptr<OSTM> _FROM_OSTM_ONE_, _FROM_OSTM_TWO_, _TO_OSTM_;
00137     std::shared_ptr<BANK> _FROM_, _FROM_TWO_, _TO_;
00138
00139     bool done = false;
00140     try {
00141         while (!done) {
00142             // for (int i = 0; i < vector_number; ++i) {
00143             for (auto&& obj : _customer_vec) {
00147                 // auto&& obj = _customer_vec.at(i);
00148                 tx->_register(obj);
00152                 _FROM_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_));
00153                 _FROM_TWO_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_two_));
00154                 _TO_ = std::dynamic_pointer_cast<BANK> (tx->load(obj));
00158                 _FROM_->SetBalance(_FROM_->GetBalance() - _amount);
00159                 _FROM_TWO_->SetBalance(_FROM_TWO_->GetBalance() - _amount);
00160                 _TO_->SetBalance(_TO_->GetBalance() + (_amount * 2));
00164                 _FROM_OSTM_ONE_ = std::dynamic_pointer_cast<OSTM> (_FROM_);
00165                 _FROM_OSTM_TWO_ = std::dynamic_pointer_cast<OSTM> (_FROM_TWO_);
00166                 _TO_OSTM_ = std::dynamic_pointer_cast<OSTM> (_TO_);
00170                 tx->store(_FROM_OSTM_ONE_);
00171                 tx->store(_FROM_OSTM_TWO_);
00172                 tx->store(_TO_OSTM_);
00173             }
00177             done = tx->commit();
00178         }
00179     } catch (std::runtime_error& e) {
00180         std::cout << e.what() << std::endl;
00181     }
00182 }
00194 void MyTestCAse::_six_account_transfer_(std::shared_ptr<OSTM> _to_,
      std::shared_ptr<OSTM> _from_one_, std::shared_ptr<OSTM> _from_two_, std::shared_ptr<OSTM> _from_three_,
      std::shared_ptr<OSTM> _from_four_, std::shared_ptr<OSTM> _from_five_, TM& _tm, double _amount) {
00195     std::shared_ptr<TX> tx = _tm._get_tx();
00199     tx->_register(_to_);
00200     tx->_register(_from_one_);
00201     tx->_register(_from_two_);
00202     tx->_register(_from_three_);
00203     tx->_register(_from_four_);
00204     tx->_register(_from_five_);
```

```
00205
00209     std::shared_ptr<OSTM> _TO_OSTM, _FROM_ONE_OSTM, _FROM_TWO_OSTM, _FROM_THREE_OSTM, _FROM_FOUR_OSTM,
      _FROM_FIVE_OSTM;
00210     std::shared_ptr<BANK> _TO_, _FROM_ONE_, _FROM_TWO_, _FROM_THREE_, _FROM_FOUR_, _FROM_FIVE_;
00211     try {
00212         bool done = false;
00213         while (!done) {
00217             _TO_ = std::dynamic_pointer_cast<BANK> (tx->load(_to_));
00218             _FROM_ONE_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_one_));
00219             _FROM_TWO_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_two_));
00220             _FROM_THREE_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_three_));
00221             _FROM_FOUR_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_four_));
00222             _FROM_FIVE_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_five_));
00226             _TO_->SetBalance(_TO_->GetBalance() + (_amount * 5));
00227             _FROM_ONE_->SetBalance(_FROM_ONE_->GetBalance() - _amount);
00228             _FROM_TWO_->SetBalance(_FROM_TWO_->GetBalance() - _amount);
00229             _FROM_THREE_->SetBalance(_FROM_THREE_->GetBalance() - _amount);
00230             _FROM_FOUR_->SetBalance(_FROM_FOUR_->GetBalance() - _amount);
00231             _FROM_FIVE_->SetBalance(_FROM_FIVE_->GetBalance() - _amount);
00235             _TO_OSTM = std::dynamic_pointer_cast<OSTM> (_TO_);
00236             _FROM_ONE_OSTM = std::dynamic_pointer_cast<OSTM> (_FROM_ONE_);
00237             _FROM_TWO_OSTM = std::dynamic_pointer_cast<OSTM> (_FROM_TWO_);
00238             _FROM_THREE_OSTM = std::dynamic_pointer_cast<OSTM> (_FROM_THREE_);
00239             _FROM_FOUR_OSTM = std::dynamic_pointer_cast<OSTM> (_FROM_FOUR_);
00240             _FROM_FIVE_OSTM = std::dynamic_pointer_cast<OSTM> (_FROM_FIVE_);
00244             tx->store(_TO_OSTM);
00245             tx->store(_FROM_ONE_OSTM);
00246             tx->store(_FROM_TWO_OSTM);
00247             tx->store(_FROM_THREE_OSTM);
00248             tx->store(_FROM_FOUR_OSTM);
00249             tx->store(_FROM_FIVE_OSTM);
00253             done = tx->commit();
00254         }
00255     } catch (std::runtime_error& e) {
00256         std::cout << e.what() << std::endl;
00257     }
00258 }
00266  void MyTestCAse::_two_account_transfer_(std::shared_ptr<OSTM> _to_,
      std::shared_ptr<OSTM> _from_, TM& tm, double _amount) {
00267
00268     std::shared_ptr<TX> tx = tm._get_tx();
00272     tx->_register(_to_);
00273     tx->_register(_from_);
00277     std::shared_ptr<BANK> _TO_BANK_, _FROM_BANK_;
00278     std::shared_ptr<OSTM> _TO_OSTM_, _FROM_OSTM_;
00279
00280     bool done = false;
00281     try {
00282         while (!done) {
00286             _TO_BANK_ = std::dynamic_pointer_cast<BANK> (tx->load(_to_));
00287             _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_));
00291             _TO_BANK_->SetBalance(_TO_BANK_->GetBalance() + _amount);
00292             _FROM_BANK_->SetBalance(_FROM_BANK_->GetBalance() - _amount);
00296             _TO_OSTM_ = std::dynamic_pointer_cast<OSTM> (_TO_BANK_);
00297             _FROM_OSTM_ = std::dynamic_pointer_cast<OSTM> (_FROM_BANK_);
00301             tx->store(_TO_OSTM_);
00302             tx->store(_FROM_OSTM_);
00303
00307             done = tx->commit();
00308         }
00309     } catch (std::runtime_error& e) {
00310         std::cout << e.what() << std::endl;
00311     }
00312 }
00320 void MyTestCAse::_nesting_(std::shared_ptr<OSTM> _to_, std::shared_ptr<OSTM> _from_,
      TM& _tm, double _amount) {
00321     std::shared_ptr<TX> tx = _tm._get_tx();
00325     tx->_register(_to_);
00326     tx->_register(_from_);
00330     std::shared_ptr<BANK> _TO_BANK_, _FROM_BANK_;
00331     std::shared_ptr<OSTM> _TO_OSTM_, _FROM_OSTM_;
00332
00333
00334     bool done = false;
00335     try {
00336         while (!done) {
00340             _TO_BANK_ = std::dynamic_pointer_cast<BANK> (tx->load(_to_));
00341             _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (tx->load(_from_));
00345             _TO_BANK_->SetBalance(_TO_BANK_->GetBalance() + _amount);
00346             _FROM_BANK_->SetBalance(_FROM_BANK_->GetBalance() - _amount);
00350             _TO_OSTM_ = std::dynamic_pointer_cast<OSTM> (_TO_BANK_);
00351             _FROM_OSTM_ = std::dynamic_pointer_cast<OSTM> (_FROM_BANK_);
00355             tx->store(_TO_OSTM_);
00356             tx->store(_FROM_OSTM_);
00357
00361             std::shared_ptr<TX> txTwo = _tm._get_tx();
00362
```

```
            bool nestedDone = false;
            while (!nestedDone) {
                _TO_BANK_ = std::dynamic_pointer_cast<BANK> (txTwo->load(_to_));
                _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (txTwo->load(_from_));
                _TO_BANK_->SetBalance(_TO_BANK_->GetBalance() + _amount);
                _FROM_BANK_->SetBalance(_FROM_BANK_->GetBalance() - _amount);
                _TO_OSTM_ = std::dynamic_pointer_cast<OSTM> (_TO_BANK_);
                _FROM_OSTM_ = std::dynamic_pointer_cast<OSTM> (_FROM_BANK_);
                txTwo->store(_TO_OSTM_);
                txTwo->store(_FROM_OSTM_);
                _two_account_transfer_(_to_, _from_, _tm, _amount);

                nestedDone = txTwo->commit();
            }

            done = tx->commit();
        }
    } catch (std::runtime_error& e) {
        std::cout << e.what() << std::endl;
    }
}

/*
 * TEST DECLARATION FROM HERE *****************************************************
 */


void MyTestCAse::complex_threaded_functionality_hundred_threads(){
    tm._TX_EXIT();
     std::vector<std::shared_ptr < OSTM>>_customer_vec;
     for (int i = 0; i < 600; ++i) {
        if (i % 6 == 0) {
            std::shared_ptr<OSTM> sharedptr(new AIB(i, 50, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
            _customer_vec.push_back(std::move(sharedptr));
        } else if (i % 5 == 0) {
            std::shared_ptr<OSTM> sharedptr(new BOI(i, 50, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
            _customer_vec.push_back(std::move(sharedptr));
        } else if (i % 4 == 0) {
            std::shared_ptr<OSTM> sharedptr(new BOA(i, 50, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
            _customer_vec.push_back(std::move(sharedptr));
        } else if (i % 3 == 0) {
            std::shared_ptr<OSTM> sharedptr(new SWBPLC(i, 50, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
            _customer_vec.push_back(std::move(sharedptr));
        } else if (i % 2 == 0) {
            std::shared_ptr<OSTM> sharedptr(new ULSTER(i, 50, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
            _customer_vec.push_back(std::move(sharedptr));
        } else if (i % 1 == 0) {
            std::shared_ptr<OSTM> sharedptr(new UNBL(i, 50, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
            _customer_vec.push_back(std::move(sharedptr));
        }
    }
    std::shared_ptr<BANK> _FROM_BANK_;
    std::shared_ptr<BANK> _TO_BANK_;

    std::shared_ptr<OSTM> aib_ptr(new AIB(100, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> boi_ptr(new BOI(200, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));

    int transferAmount = 1;
    int threadArraySize = 100;
    std::thread thArray[threadArraySize];

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i] = std::thread(&MyTestCAse::_complex_transfer_, this, aib_ptr, boi_ptr, std::ref(_customer_vec),  std::ref(tm), transferAmount);
    }

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i].join();
    }

    _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (boi_ptr);
    _TO_BANK_ = std::dynamic_pointer_cast<BANK> (aib_ptr);

    CPPUNIT_ASSERT(_FROM_BANK_->GetBalance() == -59500);
    CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == -59500);

}

```

```
void MyTestCAse::complex_threaded_functionality_ten_threads(){
    tm._TX_EXIT();
    std::vector<std::shared_ptr < OSTM>>_customer_vec;
     for (int i = 0; i < 600; ++i) {
        if (i % 6 == 0) {
            std::shared_ptr<OSTM> sharedptr(new AIB(i, 50, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
            _customer_vec.push_back(std::move(sharedptr));
        } else if (i % 5 == 0) {
            std::shared_ptr<OSTM> sharedptr(new BOI(i, 50, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
            _customer_vec.push_back(std::move(sharedptr));
        } else if (i % 4 == 0) {
            std::shared_ptr<OSTM> sharedptr(new BOA(i, 50, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
            _customer_vec.push_back(std::move(sharedptr));
        } else if (i % 3 == 0) {
            std::shared_ptr<OSTM> sharedptr(new SWBPLC(i, 50, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
            _customer_vec.push_back(std::move(sharedptr));
        } else if (i % 2 == 0) {
            std::shared_ptr<OSTM> sharedptr(new ULSTER(i, 50, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
            _customer_vec.push_back(std::move(sharedptr));
        } else if (i % 1 == 0) {
            std::shared_ptr<OSTM> sharedptr(new UNBL(i, 50, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
            _customer_vec.push_back(std::move(sharedptr));
        }
    }
    std::shared_ptr<BANK> _FROM_BANK_;
    std::shared_ptr<BANK> _TO_BANK_;

    std::shared_ptr<OSTM> aib_ptr(new AIB(100, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> boi_ptr(new BOI(200, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));

    int transferAmount = 1;
    int threadArraySize = 10;
    std::thread thArray[threadArraySize];

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i] = std::thread(&MyTestCAse::_complex_transfer_, this, aib_ptr, boi_ptr, std::ref(_customer_vec),  std::ref(tm), transferAmount);
    }

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i].join();
    }

    _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (boi_ptr);
    _TO_BANK_ = std::dynamic_pointer_cast<BANK> (aib_ptr);

    CPPUNIT_ASSERT(_FROM_BANK_->GetBalance() == -5500);
    CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == -5500);

}


void MyTestCAse::threaded_functionality_hundred_threads()
{
    tm._TX_EXIT();
    std::shared_ptr<BANK> _FROM_BANK_;
    std::shared_ptr<BANK> _TO_BANK_;

    std::shared_ptr<OSTM> aib_ptr(new AIB(100, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> boi_ptr(new BOI(200, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));

    int transferAmount = 1;
    int threadArraySize = 100;
    std::thread thArray[threadArraySize];

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i] = std::thread(&MyTestCAse::_two_account_transfer_, this, aib_ptr, boi_ptr, std::ref(tm), transferAmount);
    }

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i].join();
    }

    _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (boi_ptr);
    _TO_BANK_ = std::dynamic_pointer_cast<BANK> (aib_ptr);
```

```
    CPPUNIT_ASSERT(_FROM_BANK_->GetBalance() == 400);
    CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == 600);

}

void MyTestCAse::threaded_functionality_thousand_threads(){
    tm._TX_EXIT();
    std::shared_ptr<BANK> _FROM_BANK_;
    std::shared_ptr<BANK> _TO_BANK_;

    std::shared_ptr<OSTM> aib_ptr(new AIB(100, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> boi_ptr(new BOI(200, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));

    int transferAmount = 1;
    int threadArraySize = 1000;
    std::thread thArray[threadArraySize];

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i] = std::thread(&MyTestCAse::_two_account_transfer_, this, aib_ptr, boi_ptr, std::ref(tm), transferAmount);
    }

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i].join();
    }

    _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (boi_ptr);
    _TO_BANK_ = std::dynamic_pointer_cast<BANK> (aib_ptr);

    CPPUNIT_ASSERT(_FROM_BANK_->GetBalance() == -500);
    CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == 1500);

}

void MyTestCAse::threaded_functionality_hundred_threads_six_account(){
    tm._TX_EXIT();
    std::shared_ptr<BANK> _FROM_BANK_ONE,_FROM_BANK_TWO,_FROM_BANK_THREE, _FROM_BANK_FOUR, _FROM_BANK_FIVE;
    std::shared_ptr<BANK> _TO_BANK_;

    std::shared_ptr<OSTM> aib_ptr(new AIB(100, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> boi_ptr(new BOI(200, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> boa_ptr(new BOA(100, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> ulster_ptr(new ULSTER(200, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> unbl_ptr(new UNBL(100, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> swplc_ptr(new SWBPLC(200, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));

    int transferAmount = 1;
    int threadArraySize = 100;
    std::thread thArray[threadArraySize];

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i] = std::thread(&MyTestCAse::_six_account_transfer_, this, aib_ptr, boi_ptr, boa_ptr, ulster_ptr, unbl_ptr, swplc_ptr, std::ref(tm), transferAmount);
    }

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i].join();
    }

    _FROM_BANK_ONE = std::dynamic_pointer_cast<BANK> (boi_ptr);
    _FROM_BANK_TWO = std::dynamic_pointer_cast<BANK> (boa_ptr);
    _FROM_BANK_THREE = std::dynamic_pointer_cast<BANK> (ulster_ptr);
    _FROM_BANK_FOUR = std::dynamic_pointer_cast<BANK> (unbl_ptr);
    _FROM_BANK_FIVE = std::dynamic_pointer_cast<BANK> (swplc_ptr);
    _TO_BANK_ = std::dynamic_pointer_cast<BANK> (aib_ptr);

    CPPUNIT_ASSERT(_FROM_BANK_ONE->GetBalance() == 400);
    CPPUNIT_ASSERT(_FROM_BANK_TWO->GetBalance() == 400);
    CPPUNIT_ASSERT(_FROM_BANK_THREE->GetBalance() == 400);
    CPPUNIT_ASSERT(_FROM_BANK_FOUR->GetBalance() == 400);
    CPPUNIT_ASSERT(_FROM_BANK_FIVE->GetBalance() == 400);
    CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == 1000);

}

void MyTestCAse::threaded_functionality_thousand_threads_six_account
```

```
    (){
    tm._TX_EXIT();
    std::shared_ptr<BANK> _FROM_BANK_ONE,_FROM_BANK_TWO,_FROM_BANK_THREE, _FROM_BANK_FOUR, _FROM_BANK_FIVE;
    std::shared_ptr<BANK> _TO_BANK_;

    std::shared_ptr<OSTM> aib_ptr(new AIB(100, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> boi_ptr(new BOI(200, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> boa_ptr(new BOA(100, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> ulster_ptr(new ULSTER(200, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> unbl_ptr(new UNBL(100, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> swplc_ptr(new SWBPLC(200, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));

    int transferAmount = 1;
    int threadArraySize = 1000;
    std::thread thArray[threadArraySize];

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i] = std::thread(&MyTestCAse::_six_account_transfer_, this, aib_ptr, boi_ptr, boa_ptr, ulster_ptr, unbl_ptr, swplc_ptr, std::ref(tm), transferAmount);
    }

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i].join();
    }

    _FROM_BANK_ONE = std::dynamic_pointer_cast<BANK> (boi_ptr);
    _FROM_BANK_TWO = std::dynamic_pointer_cast<BANK> (boa_ptr);
    _FROM_BANK_THREE = std::dynamic_pointer_cast<BANK> (ulster_ptr);
    _FROM_BANK_FOUR = std::dynamic_pointer_cast<BANK> (unbl_ptr);
    _FROM_BANK_FIVE = std::dynamic_pointer_cast<BANK> (swplc_ptr);
    _TO_BANK_ = std::dynamic_pointer_cast<BANK> (aib_ptr);

    CPPUNIT_ASSERT(_FROM_BANK_ONE->GetBalance() == -500);
    CPPUNIT_ASSERT(_FROM_BANK_TWO->GetBalance() == -500);
    CPPUNIT_ASSERT(_FROM_BANK_THREE->GetBalance() == -500);
    CPPUNIT_ASSERT(_FROM_BANK_FOUR->GetBalance() == -500);
    CPPUNIT_ASSERT(_FROM_BANK_FIVE->GetBalance() == -500);
    CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == 5500);

}

void MyTestCAse::nested_hundred_thread_functionality(){
    tm._TX_EXIT();
    std::shared_ptr<BANK> _FROM_BANK_;
    std::shared_ptr<BANK> _TO_BANK_;

    std::shared_ptr<OSTM> aib_ptr(new AIB(100, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> boi_ptr(new BOI(200, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));

    int transferAmount = 1;
    int threadArraySize = 100;
    std::thread thArray[threadArraySize];

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i] = std::thread(&MyTestCAse::_nesting_, this, aib_ptr, boi_ptr, std::ref(tm), transferAmount);
    }

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i].join();
    }

    _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (boi_ptr);
    _TO_BANK_ = std::dynamic_pointer_cast<BANK> (aib_ptr);

    CPPUNIT_ASSERT(_FROM_BANK_->GetBalance() == 200);
    CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == 800);

}


void MyTestCAse::nested_thousand_thread_functionality(){
    tm._TX_EXIT();
    std::shared_ptr<BANK> _FROM_BANK_;
    std::shared_ptr<BANK> _TO_BANK_;

    std::shared_ptr<OSTM> aib_ptr(new AIB(100, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> boi_ptr(new BOI(200, 500, "Joe", "Blog", "High street, Kilkenny,
```

```
Co.Kilkenny"));

    int transferAmount = 1;
    int threadArraySize = 1000;
    std::thread thArray[threadArraySize];

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i] = std::thread(&MyTestCAse::_nesting_, this, aib_ptr, boi_ptr, std::ref(tm), transferAmount);
    }

    for (int i = 0; i < threadArraySize; ++i) {
        thArray[i].join();
    }

    _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (boi_ptr);
    _TO_BANK_ = std::dynamic_pointer_cast<BANK> (aib_ptr);

    CPPUNIT_ASSERT(_FROM_BANK_->GetBalance() == -2500);
    CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == 3500);

}

void MyTestCAse::two_object_transfer_complete(){
    tm._TX_EXIT();
    std::shared_ptr<OSTM> aib_ptr(new AIB(100, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> boi_ptr(new BOI(200, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<BANK> _FROM_BANK_;
    std::shared_ptr<BANK> _TO_BANK_;
    a->_two_account_transfer_(aib_ptr, boi_ptr, tm, 20);
    _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (boi_ptr);
    _TO_BANK_ = std::dynamic_pointer_cast<BANK> (aib_ptr);

    CPPUNIT_ASSERT(_FROM_BANK_->GetBalance() == 480);
    CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == 520);

}
void MyTestCAse::two_object_transfer_state_change(){
    tm._TX_EXIT();
    std::shared_ptr<OSTM> boa_ptr(new BOA(300, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> swplc_ptr(new SWBPLC(400, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<BANK> _FROM_BANK_;
    std::shared_ptr<BANK> _TO_BANK_;
    a->_two_account_transfer_(swplc_ptr, boa_ptr, tm, 20);
    _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (boa_ptr);
    _TO_BANK_ = std::dynamic_pointer_cast<BANK> (swplc_ptr);


    CPPUNIT_ASSERT(!(_FROM_BANK_->GetBalance() == 500));
    CPPUNIT_ASSERT(!(_TO_BANK_->GetBalance() == 500));

}
void MyTestCAse::nested_transaction_object_test(){
    tm._TX_EXIT();
    std::shared_ptr<OSTM> ulster_ptr(new ULSTER(500, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<OSTM> unbl_ptr(new UNBL(600, 500, "Joe", "Blog", "High street, Kilkenny, Co.Kilkenny"));
    std::shared_ptr<BANK> _FROM_BANK_;
    std::shared_ptr<BANK> _TO_BANK_;
    a->_nesting_(ulster_ptr, unbl_ptr, tm, 20);
    _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (unbl_ptr);
    _TO_BANK_ = std::dynamic_pointer_cast<BANK> (ulster_ptr);

    CPPUNIT_ASSERT(_FROM_BANK_->GetBalance() == 440);
    CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == 560);

}
/*
 * \brief The library function throws runtime error if the client application tries to
 * register null pointer in the library. Runtime error should be thrown
 */
void MyTestCAse::register_null_pointer_throw_runtime_error(){

    std::shared_ptr<OSTM> null_ptr;
    std::shared_ptr<TX> tx = tm._get_tx();

    tx->_register(null_ptr);
}

void MyTestCAse::object_not_registered_throw_runtime_error(){
```

```
00819
00820     std::shared_ptr<OSTM> not_registered_object(new BOA(300, 500, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00821     std::shared_ptr<TX> tx = tm._get_tx();
00822
00823     tx->load(not_registered_object);
00824 }
00825
00831 void MyTestCAse::store_null_pointer_throw_runtime_error()
      {
00832
00833     std::shared_ptr<OSTM> null_ptr;
00834     std::shared_ptr<TX> tx = tm._get_tx();
00835
00836     tx->store(null_ptr);
00837 }
00841 void MyTestCAse::increase_nesting(){
00842     std::shared_ptr<TX> tx = tm._get_tx();
00843     tx->setTx_nesting_level(0);
00844     tx->_increase_tx_nesting();
00845     tx->_increase_tx_nesting();
00846     tx->_increase_tx_nesting();
00847
00848     CPPUNIT_ASSERT( tx->getTx_nesting_level() == 3);
00849 }
00853 void MyTestCAse::decrease_nesting(){
00854
00855     std::shared_ptr<TX> tx = tm._get_tx();
00856     tx->setTx_nesting_level(10);
00857     tx->_decrease_tx_nesting();
00858     tx->_decrease_tx_nesting();
00859     tx->_decrease_tx_nesting();
00860
00861     CPPUNIT_ASSERT( tx->getTx_nesting_level() == 7);
00862     tx->_decrease_tx_nesting();
00863     CPPUNIT_ASSERT( tx->getTx_nesting_level() == 6);
00864 }
00868 void MyTestCAse::increase_nesting_fail(){
00869     std::shared_ptr<TX> tx = tm._get_tx();
00870     tx->setTx_nesting_level(0);
00871     tx->_increase_tx_nesting();
00872     tx->_increase_tx_nesting();
00873     tx->_increase_tx_nesting();
00874
00875     CPPUNIT_ASSERT( !(tx->getTx_nesting_level() == 0));
00876
00877 }
00881 void MyTestCAse::decrease_nesting_fail(){
00882
00883     std::shared_ptr<TX> tx = tm._get_tx();
00884     tx->setTx_nesting_level(10);
00885     tx->_decrease_tx_nesting();
00886     tx->_decrease_tx_nesting();
00887     tx->_decrease_tx_nesting();
00888
00889     CPPUNIT_ASSERT( !(tx->getTx_nesting_level() == 10));
00890     tx->_decrease_tx_nesting();
00891     CPPUNIT_ASSERT( !(tx->getTx_nesting_level() == 12));
00892 }
00896 void MyTestCAse::compare_Transaction_Manager_singleton_instance
      (){
00897     TM& tm_copy = TM::Instance();
00898     CPPUNIT_ASSERT( tm == tm_copy );
00899 }
00903 void MyTestCAse::TM_get_thread_map(){
00904
00905     std::map< std::thread::id, int > localMap;
00906     std::map< std::thread::id, int > tmMap = tm.get_thread_Map();
00907
00908     CPPUNIT_ASSERT(localMap == tmMap);
00909 }
00910
00922 void MyTestCAse::multi_threaded_multiple_object_exchange_test
      (){
00923     //ten threads using same objects
00924     tm._TX_EXIT();
00925     std::shared_ptr<BANK> _FROM_BANK_;
00926     std::shared_ptr<BANK> _TO_BANK_;
00927
00928     std::shared_ptr<OSTM> aib_ptr(new AIB(100, 100, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00929     std::shared_ptr<OSTM> boi_ptr(new BOI(200, 100, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00930
00931     int transferAmount = 5;
00932     int threadArraySize = 10;
00933     std::thread thArray[threadArraySize];
```

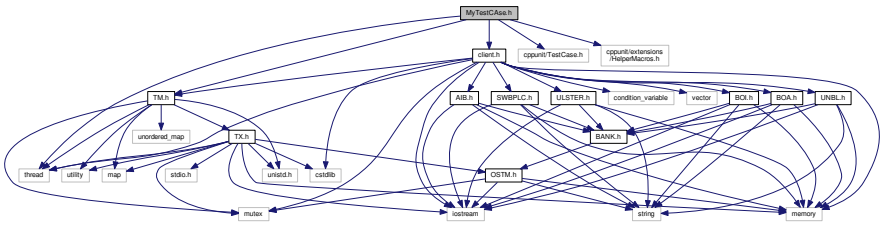
```
00934
00935     for (int i = 0; i < threadArraySize; ++i) {
00936         thArray[i] = std::thread(&MyTestCAse::_two_account_transfer_,
      this, aib_ptr, boi_ptr, std::ref(tm), transferAmount);
00937     }
00938
00939     for (int i = 0; i < threadArraySize; ++i) {
00940         thArray[i].join();
00941     }
00942
00943     _FROM_BANK_ = std::dynamic_pointer_cast<BANK> (boi_ptr);
00944     _TO_BANK_ = std::dynamic_pointer_cast<BANK> (aib_ptr);
00945
00946     CPPUNIT_ASSERT(_FROM_BANK_->GetBalance() == 50);
00947     CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == 150);
00948
00949 }
00950
00954 void MyTestCAse::multi_threaded_single_object_test_with_ten_threads
      (){
00955     //one object ten threads
00956     tm._TX_EXIT();
00957     std::shared_ptr<BANK> _TO_BANK_;
00958
00959     std::shared_ptr<OSTM> aib_ptr(new AIB(100, 100, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00960
00961     int transferAmount = 1;
00962     int threadArraySize = 10;
00963     std::thread thArray[threadArraySize];
00964
00965     for (int i = 0; i < threadArraySize; ++i) {
00966         thArray[i] = std::thread(&MyTestCAse::_one_account_transfer_,
      this, aib_ptr, std::ref(tm), transferAmount);
00967     }
00968
00969     for (int i = 0; i < threadArraySize; ++i) {
00970         thArray[i].join();
00971     }
00972
00973
00974     _TO_BANK_ = std::dynamic_pointer_cast<BANK> (aib_ptr);
00975
00976     CPPUNIT_ASSERT(_TO_BANK_->GetBalance() == 110);
00977
00978 }
00979
00983 void MyTestCAse::single_threaded_multiple_object_test(){
00984     //one transaction multiple objects six object function
00985     tm._TX_EXIT();
00986     std::vector<std::shared_ptr < OSTM>>_customer_vec;
00987      for (int i = 0; i < 10; ++i) {
00988         if (i % 6 == 0) {
00989             std::shared_ptr<OSTM> sharedptr(new AIB(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00990             _customer_vec.push_back(std::move(sharedptr));
00991         } else if (i % 5 == 0) {
00992             std::shared_ptr<OSTM> sharedptr(new BOI(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00993             _customer_vec.push_back(std::move(sharedptr));
00994         } else if (i % 4 == 0) {
00995             std::shared_ptr<OSTM> sharedptr(new BOA(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00996             _customer_vec.push_back(std::move(sharedptr));
00997         } else if (i % 3 == 0) {
00998             std::shared_ptr<OSTM> sharedptr(new SWBPLC(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
00999             _customer_vec.push_back(std::move(sharedptr));
01000         } else if (i % 2 == 0) {
01001             std::shared_ptr<OSTM> sharedptr(new ULSTER(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
01002             _customer_vec.push_back(std::move(sharedptr));
01003         } else if (i % 1 == 0) {
01004             std::shared_ptr<OSTM> sharedptr(new UNBL(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
01005             _customer_vec.push_back(std::move(sharedptr));
01006         }
01007     }
01008     std::shared_ptr<BANK> _one_, _two_, _three_, _four_, _five_, _six_;
01009     int loop = 5;
01010     int transferAmount = 1;
01011     int threadArraySize = 1;
01012     std::thread thArray[threadArraySize];
01013
01014     for (int i = 0; i < threadArraySize; ++i) {
01015         thArray[i] = std::thread(&MyTestCAse::_collection_bject_, this,
      std::ref(_customer_vec),  std::ref(tm), transferAmount , loop);
```

```
01016     }
01017
01018     for (int i = 0; i < threadArraySize; ++i) {
01019         thArray[i].join();
01020     }
01021
01022     _one_ = std::dynamic_pointer_cast<BANK> (_customer_vec[0]);
01023     _two_ = std::dynamic_pointer_cast<BANK> (_customer_vec[1]);
01024     _three_ = std::dynamic_pointer_cast<BANK> (_customer_vec[2]);
01025     _four_ = std::dynamic_pointer_cast<BANK> (_customer_vec[3]);
01026     _five_ = std::dynamic_pointer_cast<BANK> (_customer_vec[4]);
01027     _six_ = std::dynamic_pointer_cast<BANK> (_customer_vec[5]);
01028
01029     CPPUNIT_ASSERT(_one_->GetBalance() == 15);
01030     CPPUNIT_ASSERT(_two_->GetBalance() == 15);
01031     CPPUNIT_ASSERT(_three_->GetBalance() == 15);
01032     CPPUNIT_ASSERT(_four_->GetBalance() == 15);
01033     CPPUNIT_ASSERT(_five_->GetBalance() == 15);
01034     CPPUNIT_ASSERT(_six_->GetBalance() == 15);
01035 }
01036
01040 void MyTestCAse::multi_threaded_multiple_objects_test(){
01041     //three threads all thread use different objects
01042
01043     tm._TX_EXIT();
01044     std::vector<std::shared_ptr < OSTM>>_customer_vec;
01045      for (int i = 0; i < 10; ++i) {
01046         if (i % 6 == 0) {
01047             std::shared_ptr<OSTM> sharedptr(new AIB(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
01048             _customer_vec.push_back(std::move(sharedptr));
01049         } else if (i % 5 == 0) {
01050             std::shared_ptr<OSTM> sharedptr(new BOI(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
01051             _customer_vec.push_back(std::move(sharedptr));
01052         } else if (i % 4 == 0) {
01053             std::shared_ptr<OSTM> sharedptr(new BOA(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
01054             _customer_vec.push_back(std::move(sharedptr));
01055         } else if (i % 3 == 0) {
01056             std::shared_ptr<OSTM> sharedptr(new SWBPLC(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
01057             _customer_vec.push_back(std::move(sharedptr));
01058         } else if (i % 2 == 0) {
01059             std::shared_ptr<OSTM> sharedptr(new ULSTER(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
01060             _customer_vec.push_back(std::move(sharedptr));
01061         } else if (i % 1 == 0) {
01062             std::shared_ptr<OSTM> sharedptr(new UNBL(i, 10, "Joe", "Blog", "High street, Kilkenny,
       Co.Kilkenny"));
01063             _customer_vec.push_back(std::move(sharedptr));
01064         }
01065     }
01066     std::shared_ptr<BANK> _one_, _two_, _three_, _four_, _five_, _six_;
01067     int loop = 1;
01068     int transferAmount = 1;
01069     int threadArraySize = 10;
01070     std::thread thArray[threadArraySize];
01071
01072     for (int i = 0; i < threadArraySize; ++i) {
01073         thArray[i] = std::thread(&MyTestCAse::_collection_bject_, this,
      std::ref(_customer_vec),  std::ref(tm), transferAmount , loop);
01074     }
01075
01076     for (int i = 0; i < threadArraySize; ++i) {
01077         thArray[i].join();
01078     }
01079
01080     _one_ = std::dynamic_pointer_cast<BANK> (_customer_vec[0]);
01081     _two_ = std::dynamic_pointer_cast<BANK> (_customer_vec[1]);
01082     _three_ = std::dynamic_pointer_cast<BANK> (_customer_vec[2]);
01083     _four_ = std::dynamic_pointer_cast<BANK> (_customer_vec[3]);
01084     _five_ = std::dynamic_pointer_cast<BANK> (_customer_vec[4]);
01085     _six_ = std::dynamic_pointer_cast<BANK> (_customer_vec[5]);
01086
01087     CPPUNIT_ASSERT(_one_->GetBalance() == 20);
01088     CPPUNIT_ASSERT(_two_->GetBalance() == 20);
01089     CPPUNIT_ASSERT(_three_->GetBalance() == 20);
01090     CPPUNIT_ASSERT(_four_->GetBalance() == 20);
01091     CPPUNIT_ASSERT(_five_->GetBalance() == 20);
01092     CPPUNIT_ASSERT(_six_->GetBalance() == 20);
01093 }
```

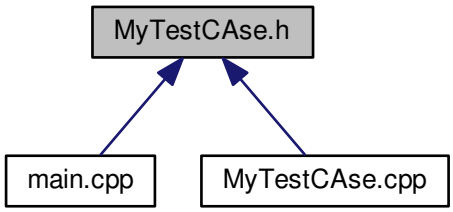
5.25 MyTestCAse.h File Reference

```
#include "client.h"
#include "TM.h"
#include <thread>
#include <cppunit/TestCase.h>
#include <cppunit/extensions/HelperMacros.h>
```

Include dependency graph for MyTestCAse.h:



This graph shows which files directly or indirectly include this file:



Classes

- class MyTestCAse

5.26 MyTestCAse.h

```
00001
00002
00003 #ifndef MYTESTCASE_H
00004 #define MYTESTCASE_H
00005
00006 //Connection between the library and the classes
00007 #include "client.h"
00008 #include "TM.h"
00009 #include <thread>
00010
00011 //#include "TestClient.cpp"
00012
00013 #include <cppunit/TestCase.h>
00014 #include <cppunit/extensions/HelperMacros.h>
00015
00016 using namespace CppUnit;
00017
```

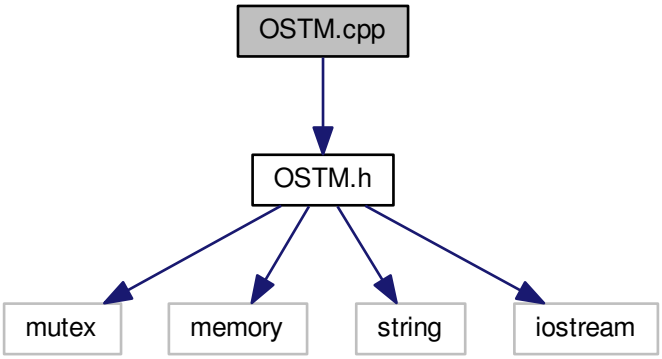
```
00018 class MyTestCAse : public TestCase{
00019     CPPUNIT_TEST_SUITE(MyTestCAse);
00020     /*
00021      * Complex library tests.
00022      */
00023     CPPUNIT_TEST(threaded_functionality_hundred_threads);
00024     CPPUNIT_TEST(threaded_functionality_thousand_threads);
00025     CPPUNIT_TEST(threaded_functionality_hundred_threads_six_account);
00026     CPPUNIT_TEST(threaded_functionality_thousand_threads_six_account);
00027     CPPUNIT_TEST(nested_hundred_thread_functionality);
00028     CPPUNIT_TEST(nested_thousand_thread_functionality);
00029     CPPUNIT_TEST(complex_threaded_functionality_hundred_threads);
00030     CPPUNIT_TEST(complex_threaded_functionality_ten_threads);
00031     CPPUNIT_TEST(two_object_transfer_complete);
00032     CPPUNIT_TEST(two_object_transfer_state_change);
00033     CPPUNIT_TEST(nested_transaction_object_test);
00034     /*
00035      * Design Manual document based tests
00036      */
00037     CPPUNIT_TEST(multi_threaded_multiple_object_exchange_test);
00038     CPPUNIT_TEST(multi_threaded_single_object_test_with_ten_threads);
00039     CPPUNIT_TEST(single_threaded_multiple_object_test);
00040     CPPUNIT_TEST(multi_threaded_multiple_objects_test);
00041     /*
00042      * OSTM library-API functions tests, private & public methods
00043      */
00044     CPPUNIT_TEST(increase_nesting);
00045     CPPUNIT_TEST(increase_nesting_fail);
00046     CPPUNIT_TEST(decrease_nesting);
00047     CPPUNIT_TEST(decrease_nesting_fail);
00048     CPPUNIT_TEST(two_object_transfer_state_change);
00049     CPPUNIT_TEST_EXCEPTION(register_null_pointer_throw_runtime_error, std::runtime_error);
00050     CPPUNIT_TEST_EXCEPTION(object_not_registered_throw_runtime_error, std::runtime_error);
00051     CPPUNIT_TEST_EXCEPTION(store_null_pointer_throw_runtime_error, std::runtime_error);
00052     CPPUNIT_TEST(compare_Transaction_Manager_singleton_instance);
00053     CPPUNIT_TEST(TM_get_thread_map);
00054
00055
00056
00057
00058
00059     CPPUNIT_TEST_SUITE_END();
00060 public:
00061     MyTestCAse(){};
00062     MyTestCAse(const MyTestCAse& orig);
00063     virtual ~MyTestCAse();
00064
00065     TM& tm = TM::Instance();
00066     std::shared_ptr<OSTM> aib_ptr;
00067     std::shared_ptr<OSTM> boi_ptr;
00068     std::shared_ptr<OSTM> boa_ptr;
00069     std::shared_ptr<OSTM> swplc_ptr;
00070     std::shared_ptr<OSTM> ulster_ptr;
00071     std::shared_ptr<OSTM> unbl_ptr;
00072
00073     void complex_threaded_functionality_hundred_threads();
00074     void complex_threaded_functionality_ten_threads();
00075     void threaded_functionality_hundred_threads();
00076     void threaded_functionality_thousand_threads();
00077     void threaded_functionality_hundred_threads_six_account();
00078     void threaded_functionality_thousand_threads_six_account();
00079     void nested_hundred_thread_functionality();
00080     void nested_thousand_thread_functionality();
00081     void two_object_transfer_complete();
00082     void two_object_transfer_state_change();
00083
00084     void multi_threaded_multiple_object_exchange_test();
00085     void multi_threaded_single_object_test_with_ten_threads();
00086     void single_threaded_multiple_object_test();
00087     void multi_threaded_multiple_objects_test();
00088
00089     void register_null_pointer_throw_runtime_error();
00090     void object_not_registered_throw_runtime_error();
00091     void store_null_pointer_throw_runtime_error();
00092     void increase_nesting();
00093     void decrease_nesting();
00094     void increase_nesting_fail();
00095     void decrease_nesting_fail();
00096     void compare_Transaction_Manager_singleton_instance();
00097     void TM_get_thread_map();
00098     void nested_transaction_object_test();
00099
00100
00101     void _two_account_transfer_(std::shared_ptr<OSTM> _to_, std::shared_ptr<OSTM> _from_,
      TM& tm, double _amount);
00102     void _nesting_(std::shared_ptr<OSTM> _to_, std::shared_ptr<OSTM> _from_, TM& _tm, double _amount);
00103     void _six_account_transfer_(std::shared_ptr<OSTM> _to_, std::shared_ptr<OSTM> _from_one_,
```

```
std::shared_ptr<OSTM> _from_two_, std::shared_ptr<OSTM> _from_three_, std::shared_ptr<OSTM> _from_four_,
std::shared_ptr<OSTM> _from_five_, TM& _tm, double _amount);
    void _complex_transfer_(std::shared_ptr<OSTM> _from_, std::shared_ptr<OSTM> _from_two_, std::vector<
std::shared_ptr<OSTM>> _customer_vec, TM& _tm, double _amount);
    void _one_account_transfer_(std::shared_ptr<OSTM> _to_, TM& _tm, double _amount);
    void _collection_bject_(std::vector<std::shared_ptr<OSTM>> _customer_vec, TM& _tm, double _amount,
int loop);

  void setUp()
  {
      a = new client(1);
      b = new client(1);
      c = new client(1);

  }

  void tearDown()
  {
      delete a;
      delete b;
      delete c;
  }


private:
    //Private pointer to use in the library
    client *a,*b,*c;

};

#endif /* MYTESTCASE_H */

```

5.27 OSTM.cpp File Reference

`#include "OSTM.h"`

Include dependency graph for OSTM.cpp:



5.28 OSTM.cpp

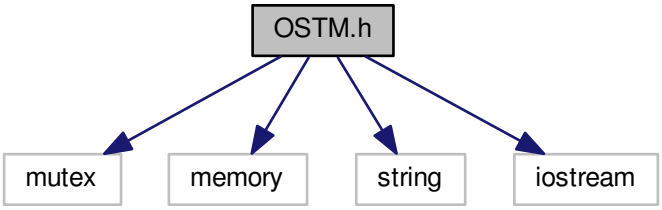
```
/*
 * File:   OSTM.cpp
 * Author: Zoltan Fuzesi
 *
 * Created on December 18, 2017, 2:09 PM
 * OSTM cpp file methods implementations
 */

```

```
#include "OSTM.h"

int OSTM::global_Unique_ID_Number = 0;

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


OSTM::OSTM(int _version_number_, int _unique_id_)
{
   // 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;
}

OSTM::~OSTM() {
    //std::cout << "[OSTM DELETE]" << std::endl;
}
int OSTM::Get_global_Unique_ID_Number() {
    if(global_Unique_ID_Number > 10000000)
        global_Unique_ID_Number = 0;
    return ++global_Unique_ID_Number;
}

void OSTM::Set_Unique_ID(int uniqueID) {
    this->uniqueID = uniqueID;
}
int OSTM::Get_Unique_ID() const
{
    return uniqueID;
}
void OSTM::Set_Version(int version)
{
    this->version = version;
}
int OSTM::Get_Version() const
{
    return version;
}
void OSTM::increase_VersionNumber()
{
    this->version += 1;
}
void OSTM::Set_Can_Commit(bool canCommit) {
    this->canCommit = canCommit;
}
bool OSTM::Is_Can_Commit() const {
    return canCommit;
}
void OSTM::Set_Abort_Transaction(bool abortTransaction) {
    this->abort_Transaction = abortTransaction;
}
bool OSTM::Is_Abort_Transaction() const {
    return abort_Transaction;
}
void OSTM::lock_Mutex() {
    this->mutex.lock();
}
void OSTM::unlock_Mutex() {
    this->mutex.unlock();
}
bool OSTM::is_Locked(){
    return this->mutex.try_lock();
}
```

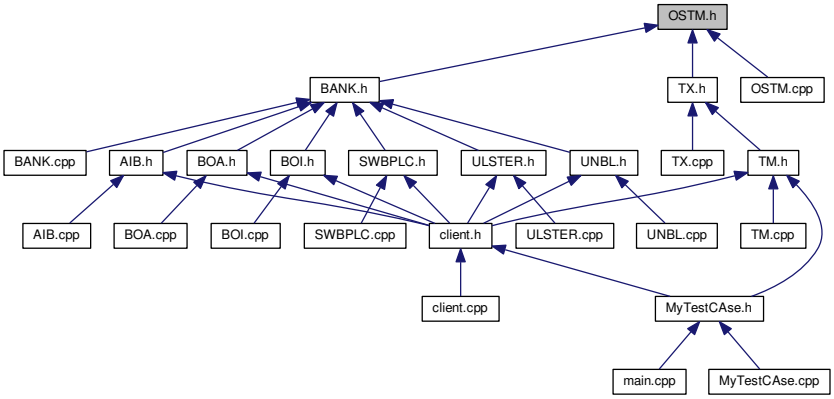
5.29 OSTM.h File Reference

```
#include <mutex>
#include <memory>
#include <string>
#include <iostream>
```

Include dependency graph for OSTM.h:



This graph shows which files directly or indirectly include this file:



Classes

- class OSTM

5.30 OSTM.h

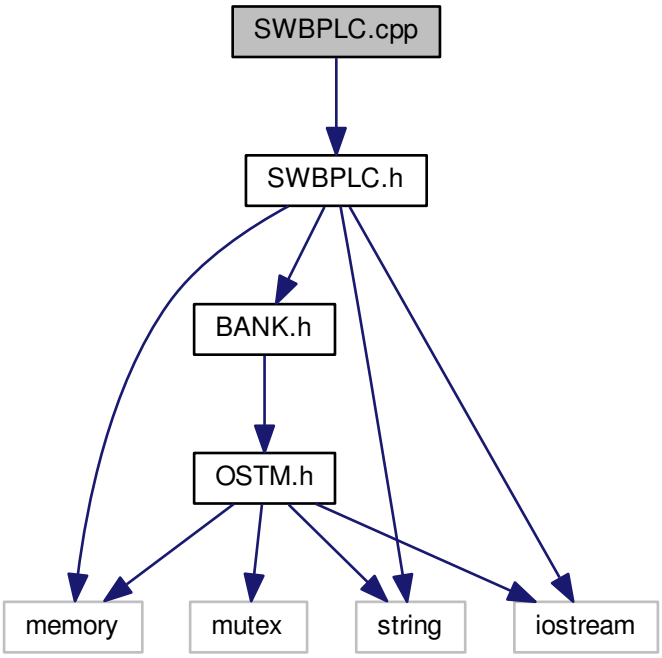
```
00001 /*
00002  * File:   OSTM.h
00003  * Author: Zoltan FUzesi
00004  *
00005  * Created on December 18, 2017, 2:09 PM
00006  * OSTM header file fields and  methods declarations
00007  */
00008
00009 #ifndef OSTM_H
00010 #define OSTM_H
00011 #include <mutex>
00012 #include <memory>
00013 #include <string>
00014 #include <iostream>
00015 #include <string>
00016
00017 class OSTM {
00018 public:
```

```
    OSTM();
    OSTM(int _version_number_, int _unique_id_);
    virtual ~OSTM();
    virtual void copy(std::shared_ptr<OSTM> from, std::shared_ptr<OSTM> to){};
    virtual std::shared_ptr<OSTM> getBaseCopy(std::shared_ptr<OSTM> object){};//std::cout <<
 "[OSTM GETBASECOPY]" << std::endl;};
    virtual void toString(){};
    void Set_Unique_ID(int uniqueID);
    int Get_Unique_ID() const;
    void Set_Version(int version);
    int Get_Version() const;
    void increase_VersionNumber();
    bool Is_Can_Commit() const;
    void Set_Can_Commit(bool canCommit);
    void Set_Abort_Transaction(bool abortTransaction);
    bool Is_Abort_Transaction() const;
    void lock_Mutex();
    void unlock_Mutex();
    bool is_Locked();

private:
    /*
     * \brief Object version number
     */
    int version;
    /*
     * \brief Object unique identifier
     */
    int uniqueID;
    /*
     * \brief Boolean value to check any other thread failed to commit
     */
    bool canCommit;
    /*
     * \brief Abort the transaction
     */
    bool abort_Transaction;
    static int global_Unique_ID_Number;
    const int ZERO = 0;
    std::mutex mutex;
    int Get_global_Unique_ID_Number();

};

#endif /* OSTM_H */
```

5.31 SWBPLC.cpp File Reference

```
#include "SWBPLC.h"
```

Include dependency graph for SWBPLC.cpp:



5.32 SWBPLC.cpp

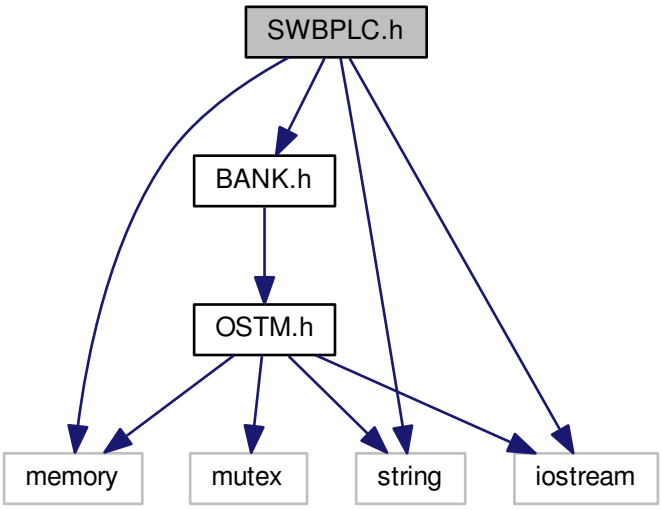
```
00001
00002 /*
00003  * File:   SWBPLC.cpp
00004  * Author: Zoltan Fuzesi
00005  * IT Carlow : C00197361
00006  *
00007  * Created on January 17, 2018, 8:02 PM
00008  */
00009
00010 #include "SWBPLC.h"
00011
00012 SWBPLC::SWBPLC(const SWBPLC& orig) {
00013 }
00014
00015 SWBPLC::~SWBPLC() {
00016 }
00017 /*
00018  * \brief getBaseCopy function, make deep copy of the object/pointer and Return a new std::shared_ptr<BANK>
       type object
00019  * \param objTO is a BANK type pointer for casting
00020  * \param obj is a std::shared_ptr<BANK> return type
00021  */
00022 std::shared_ptr<OSTM> SWBPLC::getBaseCopy(std::shared_ptr<OSTM> object)
00023 {
00024     std::shared_ptr<BANK> objTO = std::dynamic_pointer_cast<BANK>(object);
00025     std::shared_ptr<BANK> obj(new SWBPLC(objTO,object->Get_Version(),object->Get_Unique_ID()));
00026     std::shared_ptr<OSTM> ostm_obj = std::dynamic_pointer_cast<OSTM>(obj);

00027     return ostm_obj;
00028 }
00029 /*
00030  * \brief copy function, make deep copy of the object/pointer
00031  * \param objTO is a std::shared_ptr<BANK> type object casted back from std::shared_ptr<OSTM>
00032  * \param objFROM is a std::shared_ptr<BANK> type object casted back from std::shared_ptr<OSTM>
00033  */
```

```
00034 void SWBPLC::copy(std::shared_ptr<OSTM> to, std::shared_ptr<OSTM> from){
00035
00036     std::shared_ptr<SWBPLC> objTO = std::dynamic_pointer_cast<SWBPLC>(to);
00037     std::shared_ptr<SWBPLC> objFROM = std::dynamic_pointer_cast<SWBPLC>(from);
00038     objTO->Set_Unique_ID(objFROM->Get_Unique_ID());
00039     objTO->Set_Version(objFROM->Get_Version());
00040     objTO->SetAccountNumber(objFROM->GetAccountNumber());
00041     objTO->SetBalance(objFROM->GetBalance());
00042
00043
00044 }
00045 /*
00046  * \brief _cast, is use to cast bak the std::shared_ptr<OSTM> to the required type
00047  */
00048
00049 /*
00050  *  \brief toString function, displays the object values in formatted way
00051  */
00052 void SWBPLC::toString()
00053 {
00054      std::cout << "\nSWBPLC BANK" << "\nUnique ID : " << this->Get_Unique_ID() << "\nInt
       account : " << this->GetAccountNumber() << "\nDouble value : " << this->
      GetBalance() << "\nFirst name: " << this->GetFirstName() << "\nLast name : " <<
      this->GetLastName()  << "\nVersion number : " << this->Get_Version() << std::endl;
00055 }
00056
00057 void SWBPLC::SetAddress(std::string address) {
00058     this->address = address;
00059 }
00060
00061 std::string SWBPLC::GetAddress() const {
00062     return address;
00063 }
00064
00065 void SWBPLC::SetBalance(double balance) {
00066     this->balance = balance;
00067 }
00068
00069 double SWBPLC::GetBalance() const {
00070     return balance;
00071 }
00072
00073 void SWBPLC::SetAccountNumber(int accountNumber) {
00074     this->accountNumber = accountNumber;
00075 }
00076
00077 int SWBPLC::GetAccountNumber() const {
00078     return accountNumber;
00079 }
00080
00081 void SWBPLC::SetLastName(std::string lastName) {
00082     this->lastName = lastName;
00083 }
00084
00085 std::string SWBPLC::GetLastName() const {
00086     return lastName;
00087 }
00088
00089 void SWBPLC::SetFirstName(std::string firstName) {
00090     this->firstName = firstName;
00091 }
00092
00093 std::string SWBPLC::GetFirstName() const {
00094     return firstName;
00095 }
00096
00097 void SWBPLC::SetFullname(std::string fullname) {
00098     this->fullname = fullname;
00099 }
00100
00101 std::string SWBPLC::GetFullname() const {
00102     return fullname;
00103 }
00104
```

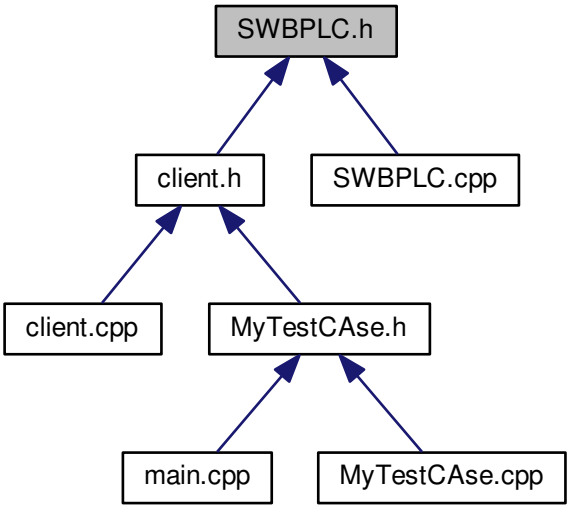
5.33 SWBPLC.h File Reference

```
#include "BANK.h"
#include <string>
#include <memory>
#include <iostream>
```

Include dependency graph for SWBPLC.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SWBPLC

5.34 SWBPLC.h

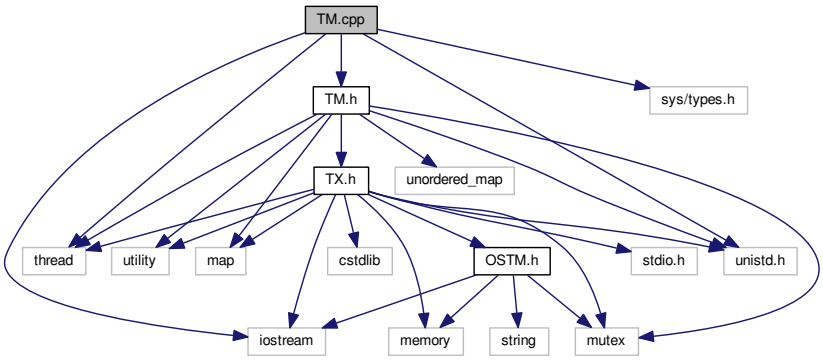
```
00001
00002 /*
00003  * File:   SWBPLC.h
00004  * Author: Zoltan Fuzesi
00005  * IT Carlow : C00197361
00006  *
00007  * Created on January 17, 2018, 8:02 PM
00008  */
00009
00010 #ifndef SWBPLC_H
00011 #define SWBPLC_H
00012 #include "BANK.h"
00013 #include <string>
00014 #include <memory>
00015 #include <iostream>
00016 /*
00017  * Inherit from BANK
00018  */
00019 class SWBPLC : public BANK {
00020 public:
00021     /*
00022      * Constructor
00023      */
00024     SWBPLC() : BANK() {
00025         this->accountNumber = 0;
00026         this->balance = 50;
00027         this->firstName = "Joe";
00028         this->lastName = "Blog";
00029         this->address = "High street, Carlow";
00030         this->fullname = firstName + " " + lastName;
00031     };
00032     /*
00033      * Custom constructor
00034      */
00035     SWBPLC(int accountNumber, double balance, std::string
      firstName, std::string lastName, std::string address) :
      BANK() {
00036         this->accountNumber = accountNumber;
00037         this->balance = balance;
00038         this->firstName = firstName;
00039         this->lastName = lastName;
00040         this->address = address;
00041         this->fullname = firstName + " " + lastName;
00042     };
00043     /*
00044      * Custom constructor, used by the library for deep copying
00045      */
00046     SWBPLC(std::shared_ptr<BANK> obj, int _version, int _unique_id) : BANK(_version, _unique_id)
      {
00047
00048         this->accountNumber = obj->GetAccountNumber();
00049         this->balance = obj->GetBalance();
00050         this->firstName = obj->GetFirstName();
00051         this->lastName = obj->GetLastName();
00052         this->address = obj->GetAddress();
00053         this->fullname = obj->GetFirstName() + " " + obj->GetLastName();
00054
00055     };
00056     /*
00057      * Copy constructor
00058      */
00059     SWBPLC(const SWBPLC& orig);
00060     /*
00061      * Operator
00062      */
00063     SWBPLC operator=(const SWBPLC& orig) {};
00064     /*
00065      * de-constructor
00066      */
00067     virtual ~SWBPLC();
00068
00069     /*
00070      * Implement OSTM virtual methods
00071      */
00072     //virtual std::shared_ptr<SWBPLC> _cast(std::shared_ptr<OSTM> _object);
00073     virtual void copy(std::shared_ptr<OSTM> to, std::shared_ptr<OSTM> from);
00074     virtual std::shared_ptr<OSTM> getBaseCopy(std::shared_ptr<OSTM> object);
00075     virtual void toString();
00076
00077     /*
00078      * Implement BANK virtual methods
00079      */
00080     virtual void SetAddress(std::string address);
00081     virtual std::string GetAddress() const;
```

```
    virtual void SetBalance(double balance);
    virtual double GetBalance() const;
    virtual void SetAccountNumber(int accountNumber);
    virtual int GetAccountNumber() const;
    virtual void SetLastName(std::string lastName);
    virtual std::string GetLastName() const;
    virtual void SetFirstName(std::string firstName);
    virtual std::string GetFirstName() const;
    virtual void SetFullname(std::string fullname);
    virtual std::string GetFullname() const;
private:
    std::string fullname;
    std::string firstName;
    std::string lastName;
    int accountNumber;
    double balance;
    std::string address;

};

#endif /* SWBPLC_H */
```

5.35 TM.cpp File Reference

```
#include "TM.h"
#include <thread>
#include <unistd.h>
#include <sys/types.h>
#include <iostream>
```

Include dependency graph for TM.cpp:



5.36 TM.cpp

```
/*
 * File:   TM.cpp
 * Author: Zoltan Fuzesi
 *
 * Created on December 18, 2017, 2:09 PM
 * Transaction Manager class methods implementation
 */
#include "TM.h"
#include <thread>
#include <unistd.h>
//#include <process.h>
#include <sys/types.h>
#include <iostream>

int TM::_tm_id;
std::map<int, std::map< std::thread::id, int >> TM::process_map_collection;
```

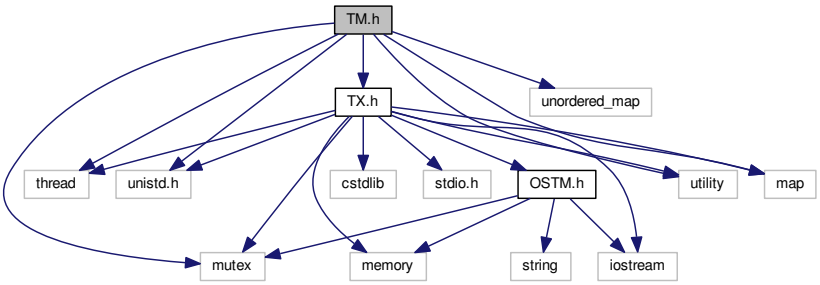
```
00028 TM& TM::Instance() {
00029     static TM _instance;
00030     _instance._tm_id = getpid();
00031
00032     return _instance;
00033 }
00034
00035 //TM Transaction managger checking the Process ID existence in the map
00036 //If not in the map then register
00043 void TM::registerTX()
00044 {
00045     std::lock_guard<std::mutex> guard(register_Lock);
00046     int ppid = getpid();
00047     std::map<int, std::map< std::thread::id, int >>::iterator process_map_collection_Iterator =
      TM::process_map_collection.find(ppid);
00048     if (process_map_collection_Iterator == TM::process_map_collection.end()) {
00049         /*
00050          * Register main process/application to the global map
00051          */
00052         std::map< std::thread::id, int >map = get_thread_Map();
00053         TM::process_map_collection.insert({ppid, map});
00054
00055     }
00056     std::map<std::thread::id, std::shared_ptr < TX>>::iterator it = txMap.find(
      std::this_thread::get_id());
00057     if (it == txMap.end()) {
00058         std::shared_ptr<TX> _transaction_object(new TX(std::this_thread::get_id()));
00059         txMap.insert({std::this_thread::get_id(), _transaction_object});
00060         /*
00061          * Get the map if registered first time
00062          */
00063         process_map_collection_Iterator = TM::process_map_collection.find(ppid);
00064         /*
00065          * Insert to the GLOBAL MAP as a helper to clean up at end of main process
00066          */
00067         process_map_collection_Iterator->second.insert({std::this_thread::get_id(), 1});
00068
00069     }
00070
00071 }
00072
00073
00079 std::shared_ptr<TX>const TM::_get_tx()
00080 {
00081     std::lock_guard<std::mutex> guard(get_Lock);
00082
00083     std::map<std::thread::id, std::shared_ptr<TX>>::iterator it = txMap.find(std::this_thread::get_id(
      ));
00084     if(it == txMap.end())
00085     {
00086        registerTX();
00087        it = txMap.find(std::this_thread::get_id());
00088
00089     } else {
00090         it->second->_increase_tx_nesting();
00091     }
00092     //it = txMap.find(std::this_thread::get_id());
00093
00094
00095     return it->second;
00096
00097 }
00102 void TM::_TX_EXIT(){
00103     TX tx(std::this_thread::get_id());
00104     int ppid = getpid();
00105     std::map<int, std::map< std::thread::id, int >>::iterator process_map_collection_Iterator =
      TM::process_map_collection.find(ppid);
00106     if (process_map_collection_Iterator != TM::process_map_collection.end()) {
00107
00108         for (auto current = process_map_collection_Iterator->second.begin(); current !=
      process_map_collection_Iterator->second.end(); ++current) {
00109             /*
00110              * Delete all transaction associated with the actual main process
00111              */
00112             txMap.erase(current->first);
00113         }
00114         TM::process_map_collection.erase(ppid);
00115
00116     }
00117     tx.ostm_exit();
00118 }
00122 void TM::print_all(){
00123     get_Lock.lock();
00124     for (auto current = txMap.begin(); current != txMap.end(); ++current) {
00125         std::cout << "KEY : " << current->first << std::endl;
00126     }
00127     get_Lock.unlock();
```

```
}

std::map< std::thread::id, int > TM::get_thread_Map() {
    std::map< std::thread::id, int > thread_Map;
    return thread_Map;
}
```

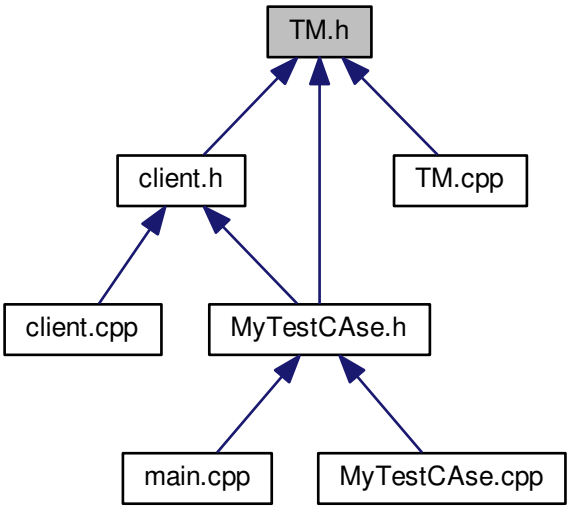
5.37 TM.h File Reference

```
#include <thread>
#include <unistd.h>
#include <mutex>
#include <unordered_map>
#include <utility>
#include <map>
#include "TX.h"
```

Include dependency graph for TM.h:



This graph shows which files directly or indirectly include this file:



Classes

- class TM

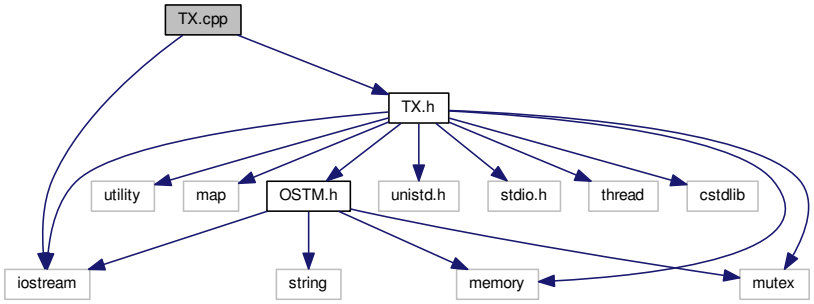
5.38 TM.h

```
/*
 * File:   TM.h
 * Author: Zoltan Fuzesi
 *
 * Created on December 18, 2017, 2:09 PM
 * Transaction Manager class fields and methods declarations
 */


#ifndef TM_H
#define TM_H

#include <thread>
#include <unistd.h>//used for pid_t
#include <mutex>
#include <unordered_map>
#include <utility>
#include <map>
#include "TX.h"

class TM {
private:
    TM() = default;
    ~TM() = default;
    TM(const TM&); //Modified for testing (= delete) removed
    TM& operator=(const TM&) = delete;
    std::map<std::thread::id, std::shared_ptr<TX>>txMap;
    static std::map<pid_t, std::map< std::thread::id, int >>
process_map_collection;
    //std::map< std::thread::id, int > get_thread_Map();
    void registerTX();
    std::mutex register_Lock;
    std::mutex get_Lock;
    static pid_t _tm_id;


public:

    static TM& Instance();
    std::shared_ptr<TX>const _get_tx();
    void _TX_EXIT();
    void print_all();
    /*
     * Added for testing
     */
    bool operator==(const TM &rhs) const {
        return &rhs == this;
    }
    //moved from private to public for testing purpose
    std::map< std::thread::id, int > get_thread_Map();




};


#endif // TM_H
```

5.39 TX.cpp File Reference

```
#include "TX.h"
#include <iostream>
```

Include dependency graph for TX.cpp:



5.40 TX.cpp

```
00001 /*
00002  * File:   TX.cpp
00003  * Author: Zoltan Fuzesi
00004  *
00005  * Created on December 18, 2017, 2:09 PM
00006  * TX cpp file methods implementations
00007  */
00008 #include "TX.h"
00009 #include <iostream>
00013 std::map<int, std::shared_ptr<OSTM> >TX::main_Process_Map_collection;
00017 std::map<int, std::map< int, int >> TX::process_map_collection;
00021 std::mutex TX::register_Lock;
00025 int TX::test_counter = 0;
00031 TX::TX(std::thread::id id) {
00032     this->transaction_Number = id;
00033     this->_tx_nesting_level = 0;
00034 }
00038 TX::~TX() {
00039
00040 }
00044 TX::TX(const TX& orig) {
00045
00046 }
00047
00052 void TX::th_exit() {
00053
00054     if (this->_tx_nesting_level > 0) {
00055         /*
00056          * Active nested transactions running in background, do not delete anything yet
00057          */
00058     } else {
00059         /*
00060          * Remove all elements map entries from transaction and clear the map
00061          */
00062         working_Map_collection.clear();
00063     }
00064 }
00065
00072 void TX::ostm_exit() {
00073     std::map<int, std::shared_ptr<OSTM>>::iterator main_Process_Map_collection_Iterator;
00074
00075     int ppid = getpid();
00076     std::map<int, 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
```

```
                 */
                TX::main_Process_Map_collection.erase(main_Process_Map_collection_Iterator->first);
            }
        }
        /*
         * Delete from Process_map_collection, Main process exits delete association with library
         */
        TX::process_map_collection.erase(process_map_collection_Iterator->first);
    }
}

void TX::_register(std::shared_ptr<OSTM> object) {
    /*
     * 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]") );
    }

    int ppid = getpid();
    std::map<int, 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()) {

        working_Map_collection.insert({object->Get_Unique_ID(), object->getBaseCopy(object)});
    }

}
std::shared_ptr<OSTM> TX::load(std::shared_ptr<OSTM> object) {

    std::map< int, std::shared_ptr<OSTM> >::iterator working_Map_collection_Object_Shared_Pointer_Iterator;
    /*
     * Check for null pointer !
     * Null pointer can cause segmentation fault!!!
     */
    if(object == nullptr){
        throw std::runtime_error(std::string("[RUNTIME ERROR : NULL POINTER IN LOAD FUNCTION]") );
    }

        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()) {

        return working_Map_collection_Object_Shared_Pointer_Iterator->second->getBaseCopy(working_Map_collection_Object_Shared_Pointer_Iterator->second);

    } else { throw std::runtime_error(std::string("[RUNTIME ERROR : NO OBJECT FOUND LOAD FUNCTION]") );}
```

```
}
void TX::store(std::shared_ptr<OSTM> object) {
    /*
     * 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; }
}
bool TX::commit() {

    bool can_Commit = true;

    /*
     * Dealing with nested transactions first
     */
    if (this->_tx_nesting_level > 0) {
        _decrease_tx_nesting();
        return true;
    }

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

    std::map<int, std::shared_ptr<OSTM>>::iterator main_Process_Map_collection_Iterator;
    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());
            /*
             * Throws runtime error if object can not find
             */
            if(main_Process_Map_collection_Iterator == TX::main_Process_Map_collection.end())
            {
                throw std::runtime_error(std::string("[RUNTIME ERROR : CAN'T FIND OBJECT COMMIT FUNCTION]") );
            }

        /*
         * Busy wait WHILE object locked by other thread
         */
        while(!(main_Process_Map_collection_Iterator->second)->is_Locked());

        if (main_Process_Map_collection_Iterator->second->Get_Version() > working_Map_collection_Object_Shared_Pointer_Iterator->second->Get_Version()) {

            working_Map_collection_Object_Shared_Pointer_Iterator->second->Set_Can_Commit(false);
            can_Commit = false;
            break;
        } else {

            working_Map_collection_Object_Shared_Pointer_Iterator->second->Set_Can_Commit(true);
        }
    }
    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();

```

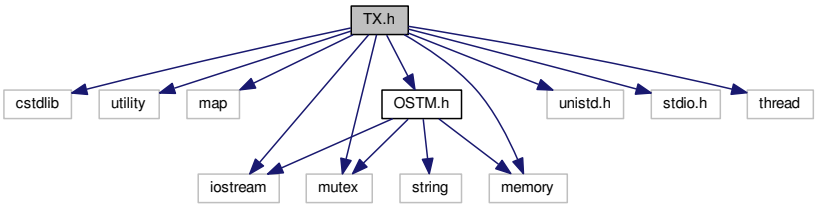
```
00254         return false;
00255     } else {
00256         /*
00257          * Commit changes
00258          */
00259         for (working_Map_collection_Object_Shared_Pointer_Iterator =
      working_Map_collection.begin(); working_Map_collection_Object_Shared_Pointer_Iterator
       != working_Map_collection.end();
      working_Map_collection_Object_Shared_Pointer_Iterator++) {
00260
00261                 main_Process_Map_collection_Iterator =
      TX::main_Process_Map_collection.find((
      working_Map_collection_Object_Shared_Pointer_Iterator->second)->Get_Unique_ID());
00262                 if (main_Process_Map_collection_Iterator !=
      TX::main_Process_Map_collection.end()) {
00263
00264                     (main_Process_Map_collection_Iterator->second)->copy(
      main_Process_Map_collection_Iterator->second, working_Map_collection_Object_Shared_Pointer_Iterator->second);
00265                     main_Process_Map_collection_Iterator->second->increase_VersionNumber();
00266
00267
00268                 } else {
00269                     throw std::runtime_error(std::string("[RUNTIME ERROR : CAN'T FIND OBJECT COMMIT
       FUNCTION]"));
00270
00271                 }
00272         }
00273
00274
00275         _release_object_lock();
00276         this->th_exit();
00277         return true;
00278     }
00279 }//Commit finish
00280
00286 void TX::_release_object_lock(){
00287
00288     std::map< int, std::shared_ptr<OSTM> >::iterator working_Map_collection_Object_Shared_Pointer_Iterator;
00289     std::map<int, std::shared_ptr<OSTM>>::iterator main_Process_Map_collection_Iterator;
00290     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++) {
00291
00292             main_Process_Map_collection_Iterator =
      TX::main_Process_Map_collection.find((
      working_Map_collection_Object_Shared_Pointer_Iterator->second)->Get_Unique_ID());
00293             if (main_Process_Map_collection_Iterator !=
      TX::main_Process_Map_collection.end()) {
00294                 /*
00295                  * Release object lock
00296                  */
00297                 (main_Process_Map_collection_Iterator)->second->unlock_Mutex();
00298
00299             }
00300         }
00301 }
00302
00307 void TX::_increase_tx_nesting() {
00308
00309     this->_tx_nesting_level += 1;
00310     // std::cout << "[this->_tx_nesting_level] = " << this->_tx_nesting_level << std::endl;
00311 }
00316 void TX::_decrease_tx_nesting() {
00317    // std::cout << "[this->_tx_nesting_level] = " << this->_tx_nesting_level << std::endl;
00318     this->_tx_nesting_level -= 1;
00319 ;
00320 }
00324 int TX::getTest_counter() {
00325     return TX::test_counter;
00326 }
00331 const std::thread::id TX::_get_tx_number() const {
00332     return transaction_Number;
00333 }
00338 std::map< int, int > TX::get_thread_Map() {
00339     std::map< int, int > thread_Map;
00340     return thread_Map;
00341 }
00342
00346 void TX::_print_all_tx() {
00347
00348     std::cout << "[PRINTALLTHREAD]" << std::endl;
00349     std::map< int, std::shared_ptr<OSTM> >::iterator it;
00350     /*
00351      * All registered thread id in the TX global
00352      */
00353     int ppid = getpid();
```

```
    std::map<int, 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;
            }


        }

    }
}

//Added only for testing
void TX::setTx_nesting_level(int _tx_nesting_level) {
    this->_tx_nesting_level = _tx_nesting_level;
}

int TX::getTx_nesting_level() const {
    return _tx_nesting_level;
}
```

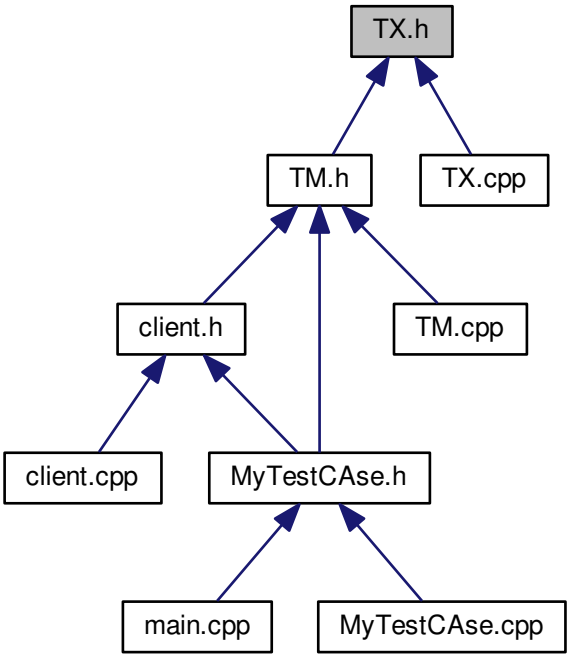
5.41 TX.h File Reference

```
#include <cstdlib>
#include <utility>
#include <map>
#include <iostream>
#include <mutex>
#include <unistd.h>
#include <memory>
#include <stdio.h>
#include <thread>
#include "OSTM.h"
```

Include dependency graph for TX.h:



This graph shows which files directly or indirectly include this file:



Classes

- class TX

5.42 TX.h

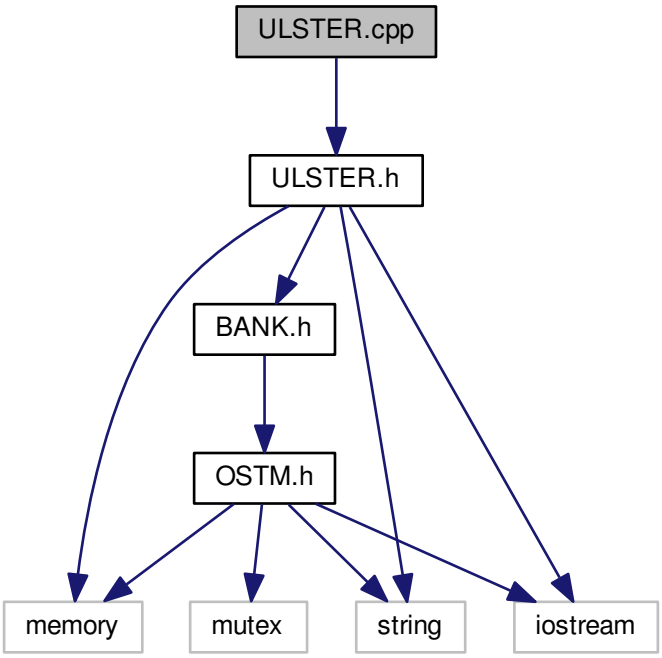
```
/*
 * File:   TX.h
 * Author: Zoltan Fuzesi
 *
 * Created on December 18, 2017, 2:09 PM
 * Transaction class fields and methods declarations
 */

#ifndef TX_H
#define TX_H
#include <cstdlib>
#include <utility>
#include <map>
#include <iostream>
#include <mutex>
#include <unistd.h>
#include <memory>
#include <stdio.h>
#include <thread>
#include "OSTM.h"

class TM;

class TX {
public:
    TX(std::thread::id id);
    ~TX();
```

```
    TX(const TX& orig);
    void ostm_exit();

    void _register(std::shared_ptr<OSTM>  object);
    std::shared_ptr<OSTM>  load(std::shared_ptr<OSTM>  object);
    void store(std::shared_ptr<OSTM>  object);
    bool commit();
    void _increase_tx_nesting();
    void _decrease_tx_nesting();
    friend class TM;
    /*
     * \brief ONLY FOR TESTING!!! returning the number of rollback happened during transactions
     */
    int getTest_counter();
    static int test_counter;
    /*
     * TESTING ONLY
     */
    void _print_all_tx() ;

    //Added only
    void setTx_nesting_level(int _tx_nesting_level);
    int getTx_nesting_level() const;


private:
    std::map< int, std::shared_ptr<OSTM>  > working_Map_collection;
    std::thread::id transaction_Number;
    int _tx_nesting_level;

    static std::map<int, std::shared_ptr<OSTM>  >main_Process_Map_collection;
    static std::map<pid_t, std::map< int, int >> process_map_collection;
    std::map< int , int > get_thread_Map();
    static std::mutex register_Lock;
    const std::thread::id _get_tx_number() const;

    void _release_object_lock();
    void th_exit();



};
#endif // _TX_H_
```

5.43 ULSTER.cpp File Reference

```
#include "ULSTER.h"
```

Include dependency graph for ULSTER.cpp:



5.44 ULSTER.cpp

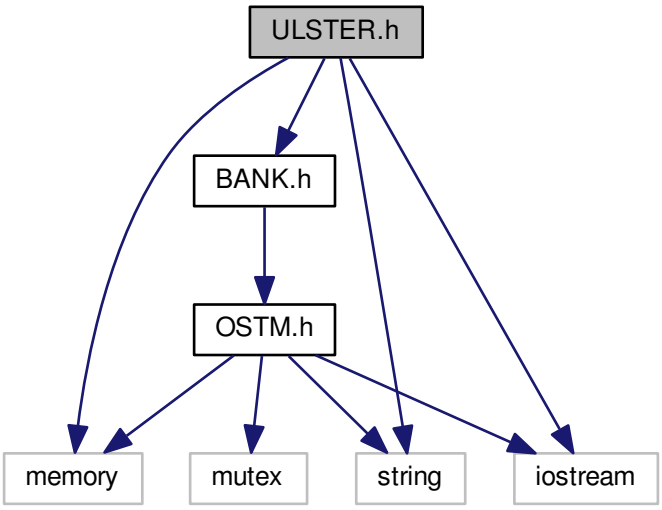
```
00001
00002 /*
00003  * File:   ULSTER.cpp
00004  * Author: Zoltan Fuzesi
00005  * IT Carlow : C00197361
00006  *
00007  * Created on January 17, 2018, 8:02 PM
00008  */
00009
00010 #include "ULSTER.h"
00011
00012
00013 ULSTER::ULSTER(const ULSTER& orig) {
00014 }
00015
00016 ULSTER::~ULSTER() {
00017 }
00018 /*
00019  * \brief getBaseCopy function, make deep copy of the object/pointer and Return a new std::shared_ptr<BANK>
       type object
00020  * \param objTO is a BANK type pointer for casting
00021  * \param obj is a std::shared_ptr<BANK> return type
00022  */
00023 std::shared_ptr<OSTM> ULSTER::getBaseCopy(std::shared_ptr<OSTM> object)
00024 {
00025     std::shared_ptr<BANK> objTO = std::dynamic_pointer_cast<BANK>(object);
00026     std::shared_ptr<BANK> obj(new ULSTER(objTO,object->Get_Version(),object->Get_Unique_ID()));
00027     std::shared_ptr<OSTM> ostm_obj = std::dynamic_pointer_cast<OSTM>(obj);

00028     return ostm_obj;
00029 }
00030 /*
00031  * \brief copy function, make deep copy of the object/pointer
00032  * \param objTO is a std::shared_ptr<BANK> type object casted back from std::shared_ptr<OSTM>
00033  * \param objFROM is a std::shared_ptr<BANK> type object casted back from std::shared_ptr<OSTM>
```

```
 */
void ULSTER::copy(std::shared_ptr<OSTM> to, std::shared_ptr<OSTM> from){

    std::shared_ptr<ULSTER> objTO = std::dynamic_pointer_cast<ULSTER>(to);
    std::shared_ptr<ULSTER> objFROM = std::dynamic_pointer_cast<ULSTER>(from);
    objTO->Set_Unique_ID(objFROM->Get_Unique_ID());
    objTO->Set_Version(objFROM->Get_Version());
    objTO->SetAccountNumber(objFROM->GetAccountNumber());
    objTO->SetBalance(objFROM->GetBalance());


}
/*
 * \brief _cast, is use to cast bak the std::shared_ptr<OSTM> to the required type
 */

/*
 *  \brief toString function, displays the object values in formatted way
 */
void ULSTER::toString()
{
   std::cout << "\nULSTER BANK" << "\nUnique ID : " << this->Get_Unique_ID() << "\nInt account
    : " << this->GetAccountNumber() << "\nDouble value : " << this->
GetBalance() << "\nFirst name: " << this->GetFirstName() << "\nLast name : " <<
this->GetLastName()  << "\nVersion number : " << this->Get_Version() << std::endl;
}

void ULSTER::SetAddress(std::string address) {
    this->address = address;
}

std::string ULSTER::GetAddress() const {
    return address;
}

void ULSTER::SetBalance(double balance) {
    this->balance = balance;
}

double ULSTER::GetBalance() const {
    return balance;
}

void ULSTER::SetAccountNumber(int accountNumber) {
    this->accountNumber = accountNumber;
}

int ULSTER::GetAccountNumber() const {
    return accountNumber;
}

void ULSTER::SetLastName(std::string lastName) {
    this->lastName = lastName;
}

std::string ULSTER::GetLastName() const {
    return lastName;
}

void ULSTER::SetFirstName(std::string firstName) {
    this->firstName = firstName;
}

std::string ULSTER::GetFirstName() const {
    return firstName;
}

void ULSTER::SetFullname(std::string fullname) {
    this->fullname = fullname;
}

std::string ULSTER::GetFullname() const {
    return fullname;
}

```

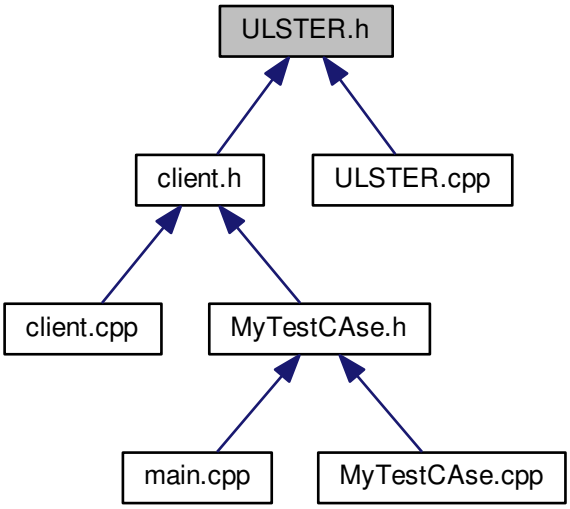
5.45 ULSTER.h File Reference

```
#include "BANK.h"
#include <string>
#include <memory>
#include <iostream>
```

Include dependency graph for ULSTER.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ULSTER

5.46 ULSTER.h

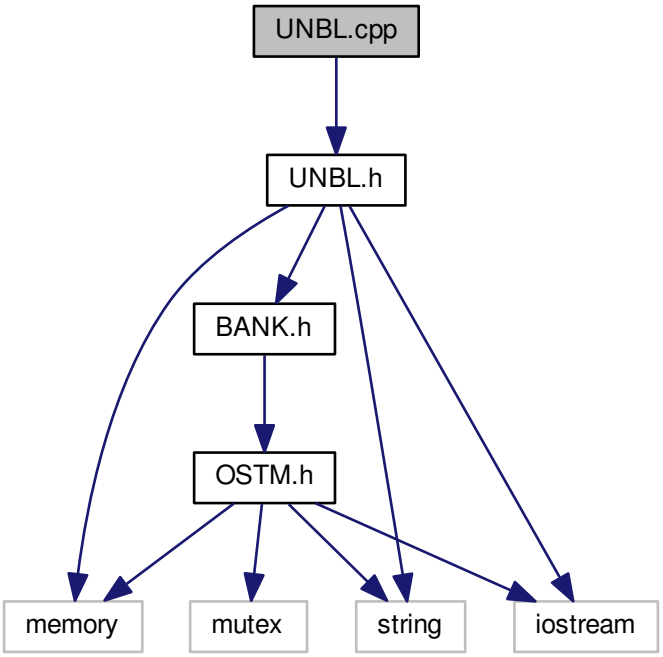
```
/*
 * File:   ULSTER.h
 * Author: Zoltan Fuzesi
 * IT Carlow : C00197361
 *
 * Created on January 17, 2018, 8:02 PM
 */

#ifndef ULSTER_H
#define ULSTER_H
#include "BANK.h"
#include <string>
#include <memory>
#include <iostream>
/*
 * Inherit from BANK
 */
class ULSTER : public BANK {
public:
    /*
     * Constructor
     */
    ULSTER() : BANK() {
        this->accountNumber = 0;
        this->balance = 50;
        this->firstName = "Joe";
        this->lastName = "Blog";
        this->address = "High street, Carlow";
        this->fullname = firstName + " " + lastName;
    };
    /*
     * Custom constructor
     */
    ULSTER(int accountNumber, double balance, std::string firstName, std::string lastName, std::string address) : BANK() {
        this->accountNumber = accountNumber;
        this->balance = balance;
        this->firstName = firstName;
        this->lastName = lastName;
        this->address = address;
        this->fullname = firstName + " " + lastName;
    };
    /*
     * Custom constructor, used by the library for deep copying
     */
    ULSTER(std::shared_ptr<BANK> obj, int _version, int _unique_id) : BANK(_version, _unique_id) {

        this->accountNumber = obj->GetAccountNumber();
        this->balance = obj->GetBalance();
        this->firstName = obj->GetFirstName();
        this->lastName = obj->GetLastName();
        this->address = obj->GetAddress();
        this->fullname = obj->GetFirstName() + " " + obj->GetLastName();
    };
    /*
     * Copy constructor
     */
    ULSTER(const ULSTER& orig);
    /*
     * Operator
     */
    ULSTER operator=(const ULSTER& orig) {};
    /*
     * de-constructor
     */
    virtual ~ULSTER();

    /*
     * Implement OSTM virtual methods
     */

    virtual void copy(std::shared_ptr<OSTM> to, std::shared_ptr<OSTM> from);
    virtual std::shared_ptr<OSTM> getBaseCopy(std::shared_ptr<OSTM> object);
    virtual void toString();

    /*
     * Implement BANK virtual methods
     */
    virtual void SetAddress(std::string address);
    virtual std::string GetAddress() const;
    virtual void SetBalance(double balance);
```

```
    virtual double GetBalance() const;
    virtual void SetAccountNumber(int accountNumber);
    virtual int GetAccountNumber() const;
    virtual void SetLastName(std::string lastName);
    virtual std::string GetLastName() const;
    virtual void SetFirstName(std::string firstName);
    virtual std::string GetFirstName() const;
    virtual void SetFullname(std::string fullname);
    virtual std::string GetFullname() const;
private:
    std::string fullname;
    std::string firstName;
    std::string lastName;
    int accountNumber;
    double balance;
    std::string address;

};

#endif /* ULSTER_H */

```

5.47 UNBL.cpp File Reference

`#include "UNBL.h"`

Include dependency graph for UNBL.cpp:



5.48 UNBL.cpp

```
/*
 * File:   UNBL.cpp
 * Author: Zoltan Fuzesi
 * IT Carlow : C00197361
 *
```

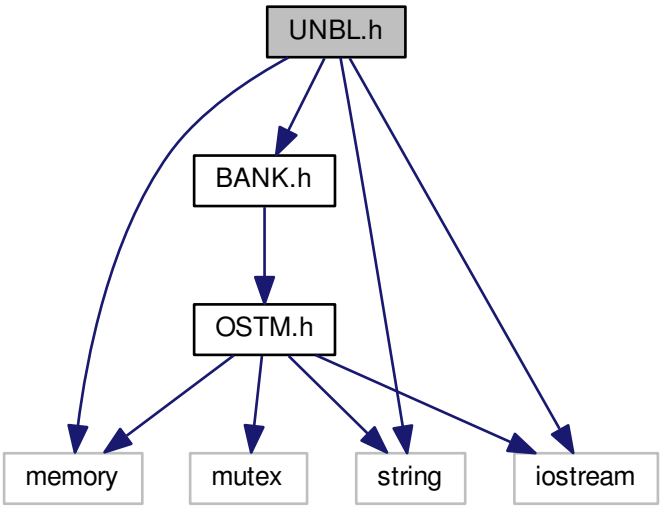
```
 * Created on January 17, 2018, 8:02 PM
 */

#include "UNBL.h"

UNBL::UNBL(const UNBL& orig) {
}

UNBL::~UNBL() {
}
/*
 * \brief getBaseCopy function, make deep copy of the object/pointer and Return a new std::shared_ptr<BANK> type object
 * \param objTO is a BANK type pointer for casting
 * \param obj is a std::shared_ptr<BANK> return type
 */
std::shared_ptr<OSTM> UNBL::getBaseCopy(std::shared_ptr<OSTM> object)
{
    std::shared_ptr<BANK> objTO = std::dynamic_pointer_cast<BANK>(object);
    std::shared_ptr<BANK> obj(new UNBL(objTO,object->Get_Version(),object->Get_Unique_ID()));
    std::shared_ptr<OSTM> ostm_obj = std::dynamic_pointer_cast<OSTM>(obj);

    return ostm_obj;
}
/*
 * \brief copy function, make deep copy of the object/pointer
 * \param objTO is a std::shared_ptr<BANK> type object casted back from std::shared_ptr<OSTM>
 * \param objFROM is a std::shared_ptr<BANK> type object casted back from std::shared_ptr<OSTM>
 */
void UNBL::copy(std::shared_ptr<OSTM> to, std::shared_ptr<OSTM> from){

    std::shared_ptr<UNBL> objTO = std::dynamic_pointer_cast<UNBL>(to);
    std::shared_ptr<UNBL> objFROM = std::dynamic_pointer_cast<UNBL>(from);
    objTO->Set_Unique_ID(objFROM->Get_Unique_ID());
    objTO->Set_Version(objFROM->Get_Version());
    objTO->SetAccountNumber(objFROM->GetAccountNumber());
    objTO->SetBalance(objFROM->GetBalance());

}
/*
 * \brief _cast, is use to cast bak the std::shared_ptr<OSTM> to the required type
 */

/*
 *  \brief toString function, displays the object values in formatted way
 */
void UNBL::toString()
{
   std::cout << "\nUNBL BANK" << "\nUnique ID : " << this->Get_Unique_ID() << "\nInt account : " << this->GetAccountNumber() << "\nDouble value : " << this->GetBalance() << "\nFirst name: " << this->GetFirstName() << "\nLast name : " << this->GetLastName()  << "\nVersion number : " << this->Get_Version() << std::endl;
}

void UNBL::SetAddress(std::string address) {
    this->address = address;
}

std::string UNBL::GetAddress() const {
    return address;
}

void UNBL::SetBalance(double balance) {
    this->balance = balance;
}

double UNBL::GetBalance() const {
    return balance;
}

void UNBL::SetAccountNumber(int accountNumber) {
    this->accountNumber = accountNumber;
}

int UNBL::GetAccountNumber() const {
    return accountNumber;
}

void UNBL::SetLastName(std::string lastName) {
    this->lastName = lastName;
}

std::string UNBL::GetLastName() const {
    return lastName;
}

void UNBL::SetFirstName(std::string firstName) {
```

```
    this->firstName = firstName;
}

std::string UNBL::GetFirstName() const {
    return firstName;
}

void UNBL::SetFullname(std::string fullname) {
    this->fullname = fullname;
}

std::string UNBL::GetFullname() const {
    return fullname;
}

```

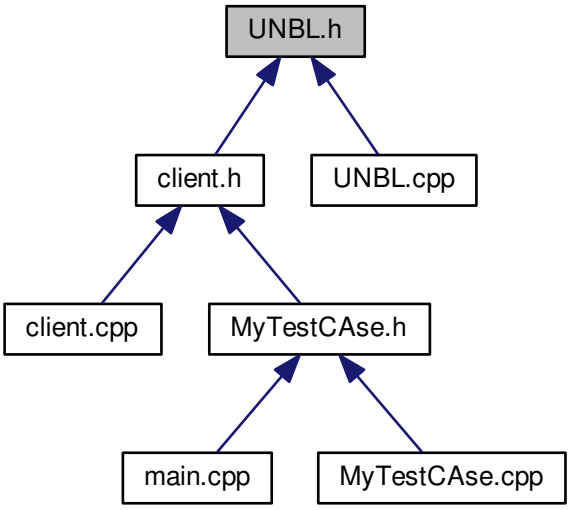
5.49 UNBL.h File Reference

```
#include "BANK.h"
#include <string>
#include <memory>
#include <iostream>
```

Include dependency graph for UNBL.h:



This graph shows which files directly or indirectly include this file:



Classes

- class UNBL

5.50 UNBL.h

```
00001
00002 /*
00003  * File:   UNBL.h
00004  * Author: Zoltan Fuzesi
00005  * IT Carlow : C00197361
00006  *
00007  * Created on January 17, 2018, 8:02 PM
00008  */
00009
00010 #ifndef UNBL_H
00011 #define UNBL_H
00012 #include "BANK.h"
00013 #include <string>
00014 #include <memory>
00015 #include <iostream>
00016
00017 class UNBL : public BANK {
00018 public:
00019     /*
00020      * Constructor
00021      */
00022     UNBL() : BANK() {
00023         this->accountNumber = 0;
00024         this->balance = 50;
00025         this->firstName = "Joe";
00026         this->lastName = "Blog";
00027         this->address = "High street, Carlow";
00028         this->fullname = firstName + " " + lastName;
00029     };
00030     /*
00031      * Custom constructor
00032      */
00033     UNBL(int accountNumber, double balance, std::string
      firstName, std::string lastName, std::string address) :
```

```
    BANK() {
00034         this->accountNumber = accountNumber;
00035         this->balance = balance;
00036         this->firstName = firstName;
00037         this->lastName = lastName;
00038         this->address = address;
00039         this->fullname = firstName + " " + lastName;
00040     };
00041     /*
00042      * Custom constructor, used by the library for deep copying
00043      */
00044     UNBL(std::shared_ptr<BANK> obj, int _version, int _unique_id) : BANK(_version, _unique_id) {
00045
00046         this->accountNumber = obj->GetAccountNumber();
00047         this->balance = obj->GetBalance();
00048         this->firstName = obj->GetFirstName();
00049         this->lastName = obj->GetLastName();
00050         this->address = obj->GetAddress();
00051         this->fullname = obj->GetFirstName() + " " + obj->GetLastName();
00052     };
00053     /*
00054      * Copy constructor
00055      */
00056     UNBL(const UNBL& orig);
00057     /*
00058      * Operator
00059      */
00060     UNBL operator=(const UNBL& orig) {};
00061     /*
00062      * de-constructor
00063      */
00064     virtual ~UNBL();
00065
00066     /*
00067      * Implement OSTM virtual methods
00068      */
00069     //virtual std::shared_ptr<UNBL> _cast(std::shared_ptr<OSTM> _object);
00070     virtual void copy(std::shared_ptr<OSTM> to, std::shared_ptr<OSTM> from);
00071     virtual std::shared_ptr<OSTM> getBaseCopy(std::shared_ptr<OSTM> object);
00072     virtual void toString();
00073
00074     /*
00075      * Implement BANK virtual methods
00076      */
00077     virtual void SetAddress(std::string address);
00078     virtual std::string GetAddress() const;
00079     virtual void SetBalance(double balance);
00080     virtual double GetBalance() const;
00081     virtual void SetAccountNumber(int accountNumber);
00082     virtual int GetAccountNumber() const;
00083     virtual void SetLastName(std::string lastName);
00084     virtual std::string GetLastName() const;
00085     virtual void SetFirstName(std::string firstName);
00086     virtual std::string GetFirstName() const;
00087     virtual void SetFullname(std::string fullname);
00088     virtual std::string GetFullname() const;
00089 private:
00090     std::string fullname;
00091     std::string firstName;
00092     std::string lastName;
00093     int accountNumber;
00094     double balance;
00095     std::string address;
00096
00097 };
00098
00099 #endif /* UNBL_H */
00100
```