SimpleTransactionalDatabase Reference Manual

1.0-beta

Generated by Doxygen 1.4.0

Thu May 12 17:58:08 2005

Contents

1 Simple Transactional Database **1**

1.1 Abstract 1

2 SimpleTransactionalDatabase Class Index **3**

2.1 SimpleTransactionalDatabase Class List 3

3 SimpleTransactionalDatabase Class Documentation **5**

3.1 Tstdb Class Reference 5

3.2 Tstdb::TAction Class Reference 26

Chapter 1

Simple Transactional Database

1.1 Abstract

Simple transactional database (STDB) is a free software library implementing a database engine based on transactions. Movements through this database are threated like "atomic transactions", so we are not allowed to introduce broken entries. This property can help us to maintain database coherence. STDB works using pairs key/value, so it can be implemented in a huge variety of applications.

This project was started by Andreu Moreno (E.U.S.S. University) and is now maintained by Alex Blanquer (pupil at the same University). The library is under GNU License, so you are welcome to copy, modify or redistribute it and/or its changes.

This documentation defines each detail of the library source.

Chapter 2

SimpleTransactionalDatabase Class Index

2.1 SimpleTransactionalDatabase Class List

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

Tstdb (Simple Transactional DataBase) 5
Tstdb::TAction (Class **TAction**(p. 26) to contain an action to be done) 26

Chapter 3

SimpleTransactionalDatabase Class Documentation

3.1 Tstdb Class Reference

Simple Transactional DataBase.

Collaboration diagram for Tstdb:

Public Member Functions

- **Tstdb** (const char *TCDBName, const char *TCDBLogName, const int ParMaxLogSize, const int ParMaxTime)
- void **Initialize** (void)

- bool **Insert** (char *key, unsigned int lkey, char *data, unsigned int ldata)
- bool **Update** (char *key, unsigned int lkey, char *data, unsigned int ldata)
- bool **Fetch** (char *key, unsigned int lkey, const char **data, unsigned int *ldata)
- bool **Delete** (char *key, unsigned int lkey)
- bool **Commit** (void)
- bool **Reorganize** ()
- bool **Rollback** (void)
- bool **Firstkey** (const char **key, unsigned int *lkey, const char **data, unsigned int *ldata)
- bool **Nextkey** (const char **key, unsigned int *lkey, const char **data, unsigned int *ldata)
- bool **Getstatus** (int *LogStatus, int *CDBStatus)
- void **Finalize** (void)
- ~**Tstdb** ()

Private Types

- typedef map< string, string, less< string > > **TDBMemory**
- typedef TDBMemory::value_type **TValue**
- enum { **LogOK**, **LogCORRUPTION**, **LogNOFILE** }
- enum { **CDBOK**, **CDBCORRUPTION**, **CDBNOFILE** }
- enum **TFunction** { **INSERT**, **UPDATE**, **DELETE** }

Private Member Functions

- bool **Getcdbline** (ifstream &in, **TAction** &action, bool &final)
- bool **Getlogline** (ifstream &in, **TAction** &action, bool &commit, int &size, bool &final)
- bool **Getline** (ifstream &in, **TAction** &action, int &size)
- bool **ParserLineOut** (string &line, **TAction** &action)
- bool **ParserLineDeleteOut** (string &line, **TAction** &action)
- bool **ReorganizeNecessary** ()

Private Attributes

- char * **db**
- char * **log**
- int **MaxLogSize**
- int **MaxTime**
- **TDBMemory DBMemory**
- TDBMemory::iterator **index**
- enum Tstdb:: { ... } **OpenLogStatus**
- enum Tstdb:: { ... } **OpenCDBStatus**
- int **ReadBytes**
- stack< **TAction**, vector< **TAction** > > **RollBack**
- queue< **TAction**, list< **TAction** > > **RedoLog**

Classes

- class **TAction**

  Class **TAction**(p. 26) *to contain an action to be done.*

3.1.1 Detailed Description

The Core Class: the database will be stored in memory inside an Tstdb object.

Definition at line 94 of file stdb.h.

3.1.2 Member Typedef Documentation

3.1.2.1 typedef map<string,string,less<string> > Tstdb::TDBMemory [private]

TDBMemory type definition.

The database is read from the db file to memory and it is stored into the object DBMemory, a TDBMemory map containing key/value pairs.

Definition at line 123 of file stdb.h.

3.1.2.2 typedef TDBMemory::value_type Tstdb::TValue [private]

TValue type definition.

This is an abstraction of every single object inside the DBMemory map.

Definition at line 129 of file stdb.h.

3.1.3 Member Enumeration Documentation

3.1.3.1 anonymous enum [private]

Log file possible status.

OpenLogStatus indicates the LOG file status, which could be correct corrupted or not existing, resp.

Definition at line 149 of file stdb.h.

```
149 {LogOK, LogCORRUPTION, LogNOFILE} OpenLogStatus;
```

3.1.3.2 anonymous enum [private]

Database file possible status.

OpenCDBStatus reports the DB file status: OK, CORRUPTED OR NOT EXISTING.

Definition at line 156 of file stdb.h.

```
156 {CDBOK, CDBCORRUPTION, CDBNOFILE} OpenCDBStatus;
```

3.1.3.3 enum Tstdb::TFunction [private]

These are possible **TAction**(p. 26) actions to be done.

Definition at line 166 of file stdb.h.

```
166 { INSERT, UPDATE, DELETE };
```

3.1.4 Constructor & Destructor Documentation

3.1.4.1 Tstdb::Tstdb (const char * *TCDBName*, const char * *TCDBLogName*, const int *ParMaxLogSize*, const int *ParMaxTime*)

Constructor Tstdb.

Features:

Truncate the log file if bad format founded

Autodetection of cdb remake operation interrumped

Parameters:

TCDBName Name of cdb file

TCDBLogName Name of log file

ParMaxLogSize Maximun log size to remake the cdbfile

ParMaxTime Max time between cdb and log files to remake db

Returns:

The Database Object

Definition at line 68 of file stdb.cpp.

References db, Initialize(), log, MaxLogSize, and MaxTime.

```
68                                                                                                  {
69
70      //  MaxLongLog
71      MaxLogSize = ParMaxLogSize;
72
73      //  MaxTime
74      MaxTime = ParMaxTime * 24 * 60 * 60; //In seconds
75
76      //  Save the path files
77      db=strdup(TCDBName);
78      log=strdup(TCDBLogName);
79
80      //  Transfer database information from files to memory
81      Initialize();
82 }
```

3.1.4.2 Tstdb::~Tstdb ()

Destructor Tstdb.

Definition at line 785 of file stdb.cpp.

```
785             {
786 }
```

3.1.5 Member Function Documentation

3.1.5.1 bool Tstdb::Commit (void)

Commit transaction.

Validate current transaction, beginning of the next one is implicit. It's essential to assure transaction integrity.

Returns:
False if anything's wrong

Definition at line 508 of file stdb.cpp.

References Tstdb::TAction::GetFunction(), log, ParserLineDeleteOut(), ParserLineOut(), RedoLog, ReorganizeNecessary(), and RollBack.

```
508                     {
509
510     //  Are there movements to log?
511     if(RedoLog.empty())
512         return true;
513
514     //  Transfer the redolog
515     //  queue to LOG file
516     ofstream logOut(log, ios::out|ios::app);
517     while(!RedoLog.empty()){
518         TAction &action = RedoLog.front();
519
520         switch(action.GetFunction()){
521
522             case INSERT:
523
524             case UPDATE:{
525                 //  Parser line out:
526                 //  Output formatting
527                 string buffer;
528                 ParserLineOut(buffer, action);
529                 //  Writting line into LOG file
530                 logOut << buffer;
531                 break;
532             }
533             case DELETE:{
534                 //  Parser Delete line out:
535                 //  Output formatting
536                 string buffer;
537                 ParserLineDeleteOut(buffer, action);
538                 //  Writting line into log file
539                 logOut << buffer;
540                 break;
541             }
542         }
543         RedoLog.pop();
544     }
545     //  Intro COMMIT constant before
546     //  closing LOG file
547     logOut << COMMIT << endl;
548
549     //  Flushing the buffer
550     logOut.flush();
551
552     //  Erase the RollBack stack
553     //  Pop all Actions from stack
554     while(!RollBack.empty())
555         RollBack.pop();
556
557     //  Verify if Reorganize() is necessary
558     ReorganizeNecessary();
559
560     return true;
561 }
```

3.1.5.2 bool Tstdb::Delete (char * *key*, unsigned int *lkey*)

Delete key/value pair from database.

Parameters:

key Key to be deleted

lkey Size of deleted key

Returns:

False if key do not exists

Definition at line 477 of file stdb.cpp.

References DBMemory, RedoLog, and RollBack.

```
477                                             {
478     string keys(key,key+lkey);
479     //  Search (key,data) into memory map
480     TDBMemory::iterator p = DBMemory.find(keys);
481     if (p != DBMemory.end()){
482         //  Key found
483         //  Pushing an Insert to the RollBack stack
484         TAction action1(INSERT, keys, p->second);
485         RollBack.push(action1);
486 
487         //  Pushing a Delete to the RedoLog queue
488         TAction action2(DELETE, keys, "");
489         RedoLog.push(action2);
490 
491         //  Erasing (key,data) from map object
492         DBMemory.erase(keys);
493 
494         return true;
495     }
496     else{
497         //  Not found
498         //  Nothing to do here
499         return false;
500     }
501 }
```

3.1.5.3 bool Tstdb::Fetch (char * *key*, unsigned int *lkey*, const char ** *data*, unsigned int * *ldata*)

Fetch the value for a given key from the database.

Parameters:

key Key to be fetched

lkey Size of the fetched key

data Returns a pointer to data

ldata Returns a pointer to size of data

Returns:

False if key do not exists

Definition at line 456 of file stdb.cpp.

References DBMemory.

```
456                                                                    {
457     string keys(key,key+lkey);
458     //  Search (key,data) into memory map
459     TDBMemory::iterator p = DBMemory.find(keys);
460     if (p != DBMemory.end()){
461         // Key has been found
462         *ldata = p->second.length();
463         *data = p->second.c_str();
464         return true;
465     }
466     else{
467         // Key has not been found
468         return false;
469     }
470 }
```

3.1.5.4 void Tstdb::Finalize (void)

Finalize: deletes the DB and LOG file.

It's not clear... why to delete these?

3.1.5.5 bool Tstdb::Firstkey (const char ** *key*, unsigned int * *lkey*, const char ** *data*, unsigned int * *ldata*)

Find first key in database.

Parameters:

key Returns a pointer to the first key

lkey Returns a pointer to the size of the first key

data Returns a pointer to the data of the first key

ldata Returns a pointer to size of data of the 1st key

Returns:

False if anything's wrong

Definition at line 730 of file stdb.cpp.

References DBMemory, and index.

```
730                                                                    {
731     //  Iterator at the beginning of the map
732     index = DBMemory.begin();
733
734     //  Verify if there are elements
735     if (index == DBMemory.end())
736         return false;
737
738     *lkey = index->first.length();
739     *key = index->first.c_str();
740
741     *ldata = index->second.length();
742     *data = index->second.c_str();
743
744     index++;
745
746     return true;
747 }
```

3.1.5.6 bool Tstdb::Getcdbline (ifstream & *in*, TAction & *action*, bool & *final*) [private]

Parser Get database file lines.

Getcdbline is designed to read an entry from the database file and recover the action it stores.

Parameters:

in The stream we are reading from

action The action returned by reference

final Indicates if it's the end

Returns:

True if DB line recovery went OK

Definition at line 234 of file stdb.cpp.

References Getline().

Referenced by Initialize().

```
234                                                     {
235
236     //  "+" check
237     char c;
238     in.get(c);
239     if (in.eof()) return false;
240     if (c=='\n'){
241         final=true;
242         in.get(c);
243         return true;
244     }
245     final=false;
246     if(c!='+')  return false;
247
248     int size;
249     return Getline(in, action, size);
250 }
```

3.1.5.7 bool Tstdb::Getline (ifstream & *in*, TAction & *action*, int & *size*) [private]

Parser generic Get line.

Getline is designed to read a formatted entry from both the DB and the LOG file. It is called from Getcdbline and Getlogline functions.

Parameters:

in The stream we are reading from

action The action returned by reference

size The size of read entry

Returns:

True if line recovery went OK

Definition at line 293 of file stdb.cpp.

Referenced by Getcdbline(), and Getlogline().

```
293                                                     {
294
295     //  Key long
296     char buffer[BUFFSIZE],c;
297     size=1;         //  Start with '+' character, counts one byte
298     in.get(buffer, BUFFSIZE, ',');
299     int lkey;
300     lkey = atol(buffer);
301     size+=strlen(buffer);
302     in.get(c);
303     if (in.eof())
304         return false;
305     if(c!=',')
306         return false;
307     size+=1;
308
309     //  Data long
310
311     int ldata;
312
313     in.get(buffer, BUFFSIZE, ':');
314     if (in.eof())
315         return false;
316     if(buffer[0]=='-'){
317         //  It's a deleted key
318         in.get(buffer[1]);
319         if (in.eof())
320             return false;
321         if(buffer[1]=='1'){
322             ldata=0;
323             size+=2;
324         }
325         else{
326             //  Format error
327             return false;
328         }
329     }
330     else{
331         //  Usual key
332         if (in.eof())
333             return false;
334         ldata = atol(buffer);
335         size+=strlen(buffer);
336     }
337
338     in.get(c);
339     if (in.eof())
340         return false;
341     if(c!=':')
342         return false;
343     size+=1;
344
345     //  Get key
346     in.read(buffer, lkey);
347     if (in.eof())
348         return false;
349     string keys(buffer, buffer+lkey);
350     size+=lkey;
351
352     //  Get "->"
353
354     in.get(c);
355     if (in.eof()) return false;
356     if(c!='-')  return false;
357     in.get(c);
358     if (in.eof()) return false;
359     if(c!='>')  return false;
```

```
360     size+=2;
361
362     //  Get data
363
364     if(ldata == 0){
365         //  Deleted key
366         TAction action1(DELETE,keys,"");
367         action=action1;
368     }
369     else{
370         //  Usual key
371         in.read(buffer, ldata);
372         string datas(buffer, buffer+ldata);
373         TAction action1(INSERT,keys,datas);
374         action=action1;
375     }
376     size+=ldata;
377
378     //  Get '\n', end of line/entry
379     in.get(c);
380     if (in.eof())
381         return false;
382     if(c!='\n')
383         return false;
384     size+=1;
385     return true;
386 }
```

3.1.5.8 bool Tstdb::Getlogline (ifstream & *in*, TAction & *action*, bool & *commit*, int & *size*, bool & *final*) [private]

Parser Get log file lines.

Getlogline is designed to read an entry from the LOG file and recover the action it stores.

Parameters:

in The stream we are reading from

action The action returned by reference

commit Indicates end of transaction

size Returns the size of the current entry

final Indicates if it's the end

Returns:

True if LOG line recovery went OK

Definition at line 257 of file stdb.cpp.

References Getline().

Referenced by Initialize().

```
257                                                                 {
258
259     //  "C" check
260     char c;
261     in.get(c);
262     if (in.eof()){
263         //  End of file detection
264         final = true;
```

```
265         in.get(c);
266         return true;
267     }
268     final = false;
269     if(c==COMMIT){
270         //  A Commit
271         commit = true;
272 
273         //  Get '\n', end of line
274         in.get(c);
275         if(in.eof()) return false;
276         if(c!='\n')  return false;
277         size = 2;
278         return true;
279     }
280     else{
281         //  Not a Commit
282         commit = false;
283         if(c!='+')  return false;
284         return Getline(in, action, size);
285     }
286 }
```

3.1.5.9 bool Tstdb::Getstatus (int * *LogStatus*, int * *CDBStatus*)

Return status.

Parameters:

LogStatus Returns a pointer to status of log file

CDBStatus Returns a pointer to status of db file

Returns:

False if anything's wrong

3.1.5.10 void Tstdb::Initialize (void)

Initialize: transfers DB from file to memory.

Initialize reads data from de LOG and DB file and store transactions inside a new DB in memory.

Definition at line 89 of file stdb.cpp.

References db, DBMemory, Getcdbline(), Tstdb::TAction::GetData(), Tstdb::TAction::GetFunction(), Tstdb::TAction::GetKey(), Getlogline(), log, OpenCDBStatus, OpenLogStatus, ReadBytes, Rollback(), and RollBack.

Referenced by Tstdb().

```
89                          {
90 
91     //  To truncate the log file if it founds format errors
92     int ReadBytesTran = 0;
93 
94     char dbtmp[100];
95     sprintf(dbtmp, "%s.tmp", db);
96 
97     //  Checking if TMP exists, because it means something went wrong
98     //  during last Reorganize. If it exists, it renames it from TMP
99     //  to DB
```

```
100     ifstream fdtmp(dbtmp, ios::in);
101     if(fdtmp)
102         rename(dbtmp,db);
103     fdtmp.close();
104 
105     //  Opening DB file in read-only mode. Now we're going to transfer
106     //  DB entries saved into the file to a new DBMemory object.
107 
108     ifstream cdbIn(db , ios::in);
109     if(!cdbIn){
110         //  If DB file doesn't exist
111         OpenCDBStatus=CDBNOFILE;
112     }
113     else{
114         //  DB file do exist
115         OpenCDBStatus=CDBOK;
116 
117         //  Loops DB file to fill the map object
118         //  It reads lines from DB file while we don't reach the end of file.
119         string buffer;
120         while (!cdbIn.eof()){
121             bool final;
122             TAction action;
123             if (!Getcdbline(cdbIn, action, final)){
124                 //  Format error encountered, file corrupted
125                 OpenCDBStatus=CDBCORRUPTION;
126                 break;
127             }
128             if(!final){
129                 //  Introduce (key,data) into map object
130                 DBMemory.insert(TValue(action.GetKey(),action.GetData()));
131             }
132         }
133     }
134 
135     //  Let's transfer the log file into memory map
136     ifstream logIn(log, ios::in);
137     if (!logIn){
138         //  LOG file doesn't exist
139         OpenLogStatus=LogNOFILE;
140     }
141     else{
142         //  LOG file do exist
143         ReadBytes=0;
144         //  Loop LOG entries/lines to fill map object
145         string buffer;
146         while (!logIn.eof()){
147             //  Read line
148             bool commit, final;
149             int size;
150             TAction action;
151 
152             if (!Getlogline(logIn, action, commit, size, final)){
153                 //  Format error
154                 OpenLogStatus=LogCORRUPTION;
155                 break;
156             }
157             if(final) continue;
158             if(commit){
159                 //  End of transaction
160                 ReadBytes+=ReadBytesTran+2;
161                 ReadBytesTran = 0;
162                 //  Pop all Actions from stack
163                 while(!RollBack.empty()){
164                     RollBack.pop();
165                 }
166             }
```

```
167             else{
168                 //  Not the end of transaction
169                 ReadBytesTran+=size;
170 
171                 //  Search (key,data) into memory map
172                 TDBMemory::iterator p = DBMemory.find(action.GetKey());
173 
174                 //  Difference between DELETE, UPDATE and INSERT and others
175                 switch(action.GetFunction()){
176                     case DELETE:
177                         //  DELETE case
178                         if (p != DBMemory.end()){
179                             //  Found
180                             //  Update the rollback with update
181                             TAction action2(INSERT, action.GetKey(), p->second);
182                             RollBack.push(action2);
183                             //  Erase (key,data) from map object
184                             DBMemory.erase(action.GetKey());
185                         }
186                         else{
187                             //  Not found
188                             //  Nothing to do here
189                         }
190                         break;
191 
192                     case UPDATE:
193                     case INSERT:
194                         //  Distinguish between INSERT and UPDATE
195                         if (p != DBMemory.end()){
196                             //  Found
197                             //  UPDATE case
198                             //  Update the rollback with update
199                             TAction action2(UPDATE, action.GetKey(), action.GetData());
200                             RollBack.push(action2);
201                             //  Update (key,data) into map object
202                             DBMemory[action.GetKey()] = action.GetData();
203                         }
204                         else{
205                             //  Not found
206                             //  INSERT case
207                             //  Update the rollback with delete
208                             TAction action2(DELETE, action.GetKey(), "");
209                             RollBack.push(action2);
210                             //  Introduce (key,data) into map object
211                             DBMemory.insert(TValue(action.GetKey(),action.GetData()));
212                         }
213                         break;
214                 }
215             }
216         }
217         //  If EOF and rollback stack is not empty -> LOG file corrupted
218         if (OpenLogStatus == LogCORRUPTION){
219             //  Rollback the current transaction
220             Rollback();
221             //  Truncate the log file
222             truncate(log, ReadBytes);
223         }
224         else
225             OpenLogStatus=LogOK;
226     }
227 }
```

3.1.5.11 bool Tstdb::Insert (char * *key*, unsigned int *lkey*, char * *data*, unsigned int *ldata*)

Insert a key/value pair to database.

Parameters:

key The key to be inserted

lkey The size of that key

data The data to be inserted

ldata The size of data

Returns:

False if key exists

Definition at line 425 of file stdb.cpp.

References DBMemory, RedoLog, and RollBack.

```
425                                                                    {
426     string keys(key,key+lkey);
427     string datas(data,data+ldata);
428     //  Search (key,data) into memory map
429     TDBMemory::iterator p = DBMemory.find(keys);
430     if (p != DBMemory.end()){
431         //  Key found,
432         //  not possible to Insert
433         return false;
434     }
435     else{
436         //  Not found
437         //  Pushing a Delete to the RollBack stack
438         TAction action1(DELETE, keys, "");
439         RollBack.push(action1);
440
441         //  Pushing an Insert to the RedoLog queue
442         TAction action2(INSERT, keys, datas);
443         RedoLog.push(action2);
444
445         //  Inserting (key,data) into map object
446         DBMemory.insert(TValue(keys,datas));
447         return true;
448     }
449 }
```

3.1.5.12 bool Tstdb::Nextkey (const char ** *key*, unsigned int * *lkey*, const char ** *data*, unsigned int * *ldata*)

Find next key in database.

Parameters:

key Returns a pointer to next key

lkey Returns a pointer to size of next key

data Returns a pointer of data

ldata: returns a pointer to size of data

Returns:
False if anything's wrong

Definition at line 754 of file stdb.cpp.

References DBMemory, and index.

```
754                                                                                 {
755
756     //  Verify if not elements
757     if (index == DBMemory.end())
758         return false;
759
760     *lkey = index->first.length();
761     *key = index->first.c_str();
762
763     *ldata = index->second.length();
764     *data = index->second.c_str();
765
766     index++;
767
768     return true;
769 }
```

3.1.5.13 bool Tstdb::ParserLineDeleteOut (string & *line*, TAction & *action*) [private]

Parser line delete out.

ParserLineDeleteOut is designed to read a DELETE action and generate a plain text line, formatted to fit into the LOG file.

Parameters:
line The output string

action The action to be parsed

Returns:
True if parsing went OK

Definition at line 596 of file stdb.cpp.

References Tstdb::TAction::GetKey().

Referenced by Commit().

```
596                                                         {
597     char buffer[100];
598
599     //  Output delete format defined here
600     sprintf(buffer, "+%lu,-1:", (unsigned long) action.GetKey().size());
601
602     char *d;
603     d=buffer+strlen(buffer);
604     memmove(d,action.GetKey().c_str(), action.GetKey().size());
605     d+=action.GetKey().size();
606     *(d++)='-';
607     *(d++)='>';
608     *(d++)='\n';
609
610     string linermp(buffer,d);
```

```
611     line = linermp;
612
613     return true;
614 }
```

3.1.5.14 bool Tstdb::ParserLineOut (string & *line*, TAction & *action*) [private]

Parser Line out.

ParserLineOut is designed to read a INSERT or UPDATE action an generate a plain text line, formatted to fit into the LOG file.

Parameters:

line The output string

action The action to be parsed

Returns:

True if parsing went OK

Definition at line 568 of file stdb.cpp.

References Tstdb::TAction::GetData(), and Tstdb::TAction::GetKey().

Referenced by Commit(), and Reorganize().

```
568                                                     {
569     char buffer[100];
570
571
572     //  Output format is defined here
573     sprintf(buffer, "+%lu,%lu:", (unsigned long)action.GetKey().size(),
574                                     (unsigned long)action.GetData().size());
575     char *d;
576     d=buffer+strlen(buffer);
577     memmove(d,action.GetKey().c_str(), action.GetKey().size());
578     d+=action.GetKey().size();
579     *(d++)='-';
580     *(d++)='>';
581     memmove(d,action.GetData().c_str(), action.GetData().size());
582     d+=action.GetData().size();
583     *(d++)='\n';
584
585     string linetmp(buffer,d);
586     line = linetmp;
587
588     return true;
589 }
```

3.1.5.15 bool Tstdb::Reorganize ()

Reorganize: transfers logged transactions to DB file.

Reorganize writes a new cdb incorporating all the logged changes, delete the log and reopen the new cdb.

Returns:

True if everything ok.

Definition at line 659 of file stdb.cpp.

References db, DBMemory, log, ParserLineOut(), and Rollback().

Referenced by ReorganizeNecessary().

```
659                     {
660     //  Rollback the current transaction if not finished
661     Rollback();
662
663     //  Rename the actual DB to DB.tmp
664     char dbtmp[100];
665     sprintf(dbtmp, "%s.tmp", db);
666     rename(db, dbtmp);
667
668     //  Checking if existing DB
669     ofstream cdbOut(db);
670     if (!cdbOut){
671         return false;
672     }
673
674     //  Iterator at the beginning of
675     //  the memory map
676     TDBMemory::iterator DBIter = DBMemory.begin();
677
678     //  Loop: rebuilding the DB file
679     while (DBIter != DBMemory.end()){
680         TAction action(INSERT,DBIter->first, DBIter->second);
681         //  Parser line out
682         string buffer;
683         ParserLineOut(buffer, action);
684         cdbOut << buffer;
685         DBIter++;
686     }
687
688     //  Closing DB file
689     cdbOut << endl;
690
691     //  Erasing the LOG file
692     unlink(log);
693
694     //  Erasing the TMP file
695     unlink(dbtmp);
696
697     return true;
698 }
```

3.1.5.16 bool Tstdb::ReorganizeNecessary () [private]

Reorganize: check if is necessary to remake the cdb file.

Depending on the value of MaxLogSize i MaxTime, Commit function will call to Reorganize automatically, to update and rebuild the DB file.

Definition at line 705 of file stdb.cpp.

References db, log, MaxLogSize, MaxTime, and Reorganize().

Referenced by Commit().

```
705                               {
706
707     struct stat file_cdb_stats;
708     struct stat file_log_stats;
```

```
709
710     if (stat(log, &file_log_stats)!=-1){
711         //  LOG exists
712         //  Verify max size
713         if (file_log_stats.st_size > MaxLogSize)
714             return Reorganize();
715
716         //  Verify max time cdb to log file
717         if (stat(db, &file_cdb_stats)!=-1){
718             if(difftime(file_log_stats.st_mtime, file_cdb_stats.st_mtime) > MaxTime)
719                 return Reorganize();
720         }
721     }
722     return true;
723 }
```

3.1.5.17 bool Tstdb::Rollback (void)

RollBack transaction.

It's like an Undo function.

Returns:
False if anything's wrong

Definition at line 621 of file stdb.cpp.

References DBMemory, Tstdb::TAction::GetData(), Tstdb::TAction::GetFunction(), Tstdb::TAction::GetKey(), RedoLog, and RollBack.

Referenced by Initialize(), and Reorganize().

```
621                         {
622     //  Redo the last transaction, using the RollBack stack
623     //  Transfer the RedoLog queue to LOG file
624     while(!RollBack.empty()){
625         TAction &action = RollBack.top();
626         switch(action.GetFunction()){
627
628             case INSERT:
629                 //  Inserting (key,data) into map object
630                 DBMemory.insert(TValue(action.GetKey(),action.GetData()));
631                 break;
632
633             case UPDATE:
634                 //  Updating (key,data) into map object
635                 DBMemory[action.GetKey()] = action.GetData();
636                 break;
637
638             case DELETE:
639                 //  Erasing (key,data) from map object
640                 DBMemory.erase(action.GetKey());
641                 break;
642         }
643         RollBack.pop();
644     }
645
646     //  Empty the RedoLog queue
647     //  Pop all Actions from queue
648     while(!RedoLog.empty())
649         RedoLog.pop();
650
651     return true;
652 }
```

3.1.5.18 bool Tstdb::Update (char * *key*, unsigned int *lkey*, char * *data*, unsigned int *ldata*)

Update a key/value pair to database.

Parameters:

key The key to be updated

lkey The size of updated key

data: The data to be updated

ldata: The size of the updated data

Returns:

False if not exists

Definition at line 393 of file stdb.cpp.

References DBMemory, RedoLog, and RollBack.

```
393                                                                    {
394     string keys(key,key+lkey);
395     string datas(data,data+ldata);
396     //  Search (key,data) into memory map
397     TDBMemory::iterator p = DBMemory.find(keys);
398     if (p != DBMemory.end()){
399         //  Found key
400         //  Update the rollback with update
401         TAction action1(UPDATE, keys, p->second);
402         RollBack.push(action1);
403
404         //  Update the redolog with update
405         TAction action2(UPDATE, keys, datas);
406         RedoLog.push(action2);
407
408         //  Update (key,data) into map object
409         DBMemory[keys] = datas;
410
411         return true;
412     }
413     else{
414         //  Key not found
415         //  Nothing to do
416         return false;
417     }
418 }
```

3.1.6 Member Data Documentation

3.1.6.1 char* Tstdb::db [private]

Filenames for the database file and the log file.

This "db" and "log" char arrays will contain the names of the database and log files, respectively.

Definition at line 102 of file stdb.h.

Referenced by Initialize(), Reorganize(), ReorganizeNecessary(), and Tstdb().

3.1.6.2 TDBMemory Tstdb::DBMemory [private]

Declaration of DBMemory map object.

This is the DBMemory object containg the database in memory.

Definition at line 135 of file stdb.h.

Referenced by Delete(), Fetch(), Firstkey(), Initialize(), Insert(), Nextkey(), Reorganize(), Rollback(), and Update().

3.1.6.3 TDBMemory::iterator Tstdb::index [private]

The index iterator.

An iterator is like a pointer designed for STL containers. The index iterator is designed to sequence objects stored into the DBMemory map.

Definition at line 142 of file stdb.h.

Referenced by Firstkey(), and Nextkey().

3.1.6.4 char * Tstdb::log [private]

Filenames for the database file and the log file.

This "db" and "log" char arrays will contain the names of the database and log files, respectively.

Definition at line 102 of file stdb.h.

Referenced by Commit(), Initialize(), Reorganize(), ReorganizeNecessary(), and Tstdb().

3.1.6.5 int Tstdb::MaxLogSize [private]

Maximum longitude of the log file to rebuild the database file.

The database should remake itself when the log file achieves this size, expressed in bytes.

Definition at line 109 of file stdb.h.

Referenced by ReorganizeNecessary(), and Tstdb().

3.1.6.6 int Tstdb::MaxTime [private]

Maximum time between cdb and log in days.

This integer variable stores the time between de database file and the log file.

Definition at line 116 of file stdb.h.

Referenced by ReorganizeNecessary(), and Tstdb().

3.1.6.7 enum { ... } Tstdb::OpenCDBStatus [private]

Database file possible status.

OpenCDBStatus reports the DB file status: OK, CORRUPTED OR NOT EXISTING.

Referenced by Initialize().

3.1.6.8 enum { ... } Tstdb::OpenLogStatus [private]

Log file possible status.

OpenLogStatus indicates the LOG file status, which could be correct corrupted or not existing, resp.

Referenced by Initialize().

3.1.6.9 int Tstdb::ReadBytes [private]

To truncate the log file if format error.

ReadBytes stores the current transaction lenght, in order to truncate the LOG file in case of data corruption.

Definition at line 163 of file stdb.h.

Referenced by Initialize().

3.1.6.10 queue<TAction, list<TAction> > Tstdb::RedoLog [private]

The RedoLog queue.

This queue is designed to store actions done to the database in order to store them into the LOG file.

Definition at line 250 of file stdb.h.

Referenced by Commit(), Delete(), Insert(), Rollback(), and Update().

3.1.6.11 stack<TAction, vector<TAction> > Tstdb::RollBack [private]

The RollBack stack.

This stack stores inverse actions in order to undo changes done to the database.

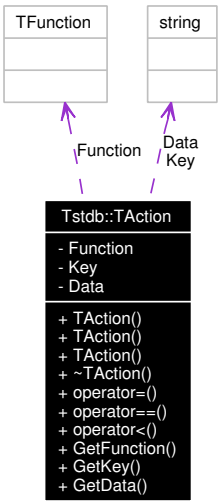
Definition at line 243 of file stdb.h.

Referenced by Commit(), Delete(), Initialize(), Insert(), Rollback(), and Update().

3.2 Tstdb::TAction Class Reference

Class **TAction**(p. 26) to contain an action to be done.

Collaboration diagram for Tstdb::TAction:



Public Member Functions

- **TAction** ()
- **TAction** (**TFunction** f, string k, string d)
- **TAction** (const **TAction** &arg)
- ~**TAction** ()
- void **operator**= (const **TAction** &copy)
- bool **operator**== (const **TAction** &arg) const
- bool **operator**< (const **TAction** &arg) const
- **TFunction GetFunction** ()
- string & **GetKey** ()
- string & **GetData** ()

Private Attributes

- **TFunction Function**
- string **Key**
- string **Data**

3.2.1 Detailed Description

TAction(p. 26) class is designed to contain actions to be stored into the RollBack stack or into the RedoLog queue.

Definition at line 174 of file stdb.h.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Tstdb::TAction::TAction () [inline]

Default constructor for **TAction**(p. 26).

Definition at line 189 of file stdb.h.

References Data, Function, and Key.

```
189 : Function(UPDATE), Key(""), Data("") { }
```

3.2.2.2 Tstdb::TAction::TAction (TFunction *f*, string *k*, string *d*) [inline]

Parametrized constructor for **TAction**(p. 26).

Definition at line 192 of file stdb.h.

References Data, Function, and Key.

```
192                                                 : Function(f), Key(k), Data(d)
193         { }
```

3.2.2.3 Tstdb::TAction::TAction (const TAction & *arg*) [inline]

Copy constructor for **TAction**(p. 26).

Definition at line 196 of file stdb.h.

References Data, Function, and Key.

```
196                                     {
197             Function=arg.Function;
198             Key=arg.Key;
199             Data=arg.Data;
200         }
```

3.2.2.4 Tstdb::TAction::~TAction () [inline]

Destructor for **TAction**(p. 26).

Definition at line 203 of file stdb.h.

```
203 { }
```

3.2.3 Member Function Documentation

3.2.3.1 string& Tstdb::TAction::GetData () [inline]

GetData returns actual **TAction**(p. 26) Data.

Definition at line 234 of file stdb.h.

References Data.

Referenced by Tstdb::Initialize(), Tstdb::ParserLineOut(), and Tstdb::Rollback().

```
234 { return Data;}
```

3.2.3.2 TFunction Tstdb::TAction::GetFunction () [inline]

Getfunction returns actual **TAction**(p. 26) Function.

Definition at line 228 of file stdb.h.

References Function.

Referenced by Tstdb::Commit(), Tstdb::Initialize(), and Tstdb::Rollback().

```
228 { return Function;}
```

3.2.3.3 string& Tstdb::TAction::GetKey () [inline]

GetKey returns actual **TAction**(p. 26) Key.

Definition at line 231 of file stdb.h.

References Key.

Referenced by Tstdb::Initialize(), Tstdb::ParserLineDeleteOut(), Tstdb::ParserLineOut(), and Tstdb::Rollback().

```
231 { return Key;}
```

3.2.3.4 bool Tstdb::TAction::operator< (const TAction & *arg*) const [inline]

Less than... operator.

Definition at line 221 of file stdb.h.

References Data, Function, and Key.

```
221                                                 {
222             return (Function < arg.Function) &&
223                         (Key < arg.Key) &&
224                         (Data < arg.Data);
225         }
```

3.2.3.5 void Tstdb::TAction::operator= (const TAction & *copy*) [inline]

Assignment operator.

Definition at line 206 of file stdb.h.

References Data, Function, and Key.

```
206                                             {
207             if (this == &copy) return;
208             Function=copy.Function;
209             Key=copy.Key;
210             Data=copy.Data;
211         }
```

3.2.3.6 bool Tstdb::TAction::operator== (const TAction & *arg*) const [inline]

Equality operator.

Definition at line 214 of file stdb.h.

References Data, Function, and Key.

```
214                                                 {
215             return (Function == arg.Function) &&
216                         (Key == arg.Key) &&
217                         (Data == arg.Data);
218         }
```

3.2.4 Member Data Documentation

3.2.4.1 string Tstdb::TAction::Data [private]

Data to apply the action.

Definition at line 184 of file stdb.h.

Referenced by GetData(), operator<(), operator=(), operator==(), and TAction().

3.2.4.2 TFunction Tstdb::TAction::Function [private]

The type of action it stores.

Definition at line 178 of file stdb.h.

Referenced by GetFunction(), operator<(), operator=(), operator==(), and TAction().

3.2.4.3 string Tstdb::TAction::Key [private]

The key to apply the action.

Definition at line 181 of file stdb.h.

Referenced by GetKey(), operator<(), operator=(), operator==(), and TAction().

Index

~TAction
    Tstdb::TAction, 27
~Tstdb
    Tstdb, 8

Commit
    Tstdb, 8

Data
    Tstdb::TAction, 29
db
    Tstdb, 23
DBMemory
    Tstdb, 23
Delete
    Tstdb, 9

Fetch
    Tstdb, 10
Finalize
    Tstdb, 11
Firstkey
    Tstdb, 11
Function
    Tstdb::TAction, 29

Getcdbline
    Tstdb, 11
GetData
    Tstdb::TAction, 27
GetFunction
    Tstdb::TAction, 28
GetKey
    Tstdb::TAction, 28
Getline
    Tstdb, 12
Getlogline
    Tstdb, 14
Getstatus
    Tstdb, 15

index
    Tstdb, 24
Initialize
    Tstdb, 15
Insert
    Tstdb, 17

Key
    Tstdb::TAction, 29

log
    Tstdb, 24

MaxLogSize
    Tstdb, 24
MaxTime
    Tstdb, 24

Nextkey
    Tstdb, 18

OpenCDBStatus
    Tstdb, 24
OpenLogStatus
    Tstdb, 24
operator<
    Tstdb::TAction, 28
operator=
    Tstdb::TAction, 28
operator==
    Tstdb::TAction, 28

ParserLineDeleteOut
    Tstdb, 19
ParserLineOut
    Tstdb, 20

ReadBytes
    Tstdb, 25
RedoLog
    Tstdb, 25
Reorganize
    Tstdb, 20
ReorganizeNecessary
    Tstdb, 21
RollBack
    Tstdb, 25
Rollback
    Tstdb, 22

TAction

Tstdb::TAction, 27
TDBMemory
Tstdb, 7
TFunction
Tstdb, 7
Tstdb, 5
~Tstdb, 8
Commit, 8
db, 23
DBMemory, 23
Delete, 9
Fetch, 10
Finalize, 11
Firstkey, 11
Getcdbline, 11
Getline, 12
Getlogline, 14
Getstatus, 15
index, 24
Initialize, 15
Insert, 17
log, 24
MaxLogSize, 24
MaxTime, 24
Nextkey, 18
OpenCDBStatus, 24
OpenLogStatus, 24
ParserLineDeleteOut, 19
ParserLineOut, 20
ReadBytes, 25
RedoLog, 25
Reorganize, 20
ReorganizeNecessary, 21
RollBack, 25
Rollback, 22
TDBMemory, 7
TFunction, 7
Tstdb, 8
TValue, 7
Update, 22
Tstdb::TAction, 26
~TAction, 27
Data, 29
Function, 29
GetData, 27
GetFunction, 28
GetKey, 28
Key, 29
operator<, 28
operator=, 28
operator==, 28
TAction, 27
TValue
Tstdb, 7
Update
Tstdb, 22