Doxygen Reference Manual

Generated by Doxygen 1.3.7

Thu Aug 24 14:45:24 2006

Contents

1 Doxygen Hierarchical Index **1**

1.1 Doxygen Class Hierarchy 1

2 Doxygen Class Index **3**

2.1 Doxygen Class List 3

3 Doxygen File Index **5**

3.1 Doxygen File List 5

4 Doxygen Class Documentation **7**

4.1 accessor Class Reference 7

4.2 basic Class Reference 9

4.3 column Struct Reference 11

4.4 datav Class Reference 12

4.5 dbCollection Struct Reference 13

4.6 dbDomain Class Reference 14

4.7 dbEnvList Class Reference 15

4.8 dbRow Class Reference 17

4.9 dbTable Class Reference 19

4.10 dbType Class Reference 21

4.11 elem Class Reference 22

4.12 indexNameVals Struct Reference 24

4.13 MysqlBin Class Reference 25

4.14 MysqlDb Class Reference 26

4.15 MysqlResult Class Reference 44

4.16 parseXmlString Class Reference 46

4.17 StDataBaseI Class Reference 50

4.18 StDbBuffer Class Reference 59

4.19 StDbBufferI Class Reference 75

4.20 StDbConfigNode Class Reference 82
4.21 StDbConfigNodeImpl Class Reference 93
4.22 StDbDefaults Class Reference 104
4.23 StDbElementIndex Class Reference 109
4.24 StDbFastSqlWriter Class Reference 113
4.25 StDbLogger Class Reference 121
4.26 StDbManager Class Reference 123
4.27 StDbManagerImpl Class Reference 135
4.28 StDbMessenger Class Reference 167
4.29 StDbMessService Class Reference 170
4.30 StDbModifier Class Reference 172
4.31 StDbNode Class Reference 179
4.32 StDbServer Class Reference 189
4.33 StDbServerImpl Class Reference 196
4.34 StDbSql Class Reference 202
4.35 StDbStoreInfo Class Reference 233
4.36 StDbTable Class Reference 236
4.37 StDbTableDescriptor Class Reference 268
4.38 StDbTableFactory Class Reference 280
4.39 StDbTableIter Class Reference 281
4.40 StDbTableIterImpl Class Reference 283
4.41 StDbTime Class Reference 286
4.42 StDbXmlReader Class Reference 289
4.43 StDbXmlWriter Class Reference 301
4.44 stsize Class Reference 312
4.45 StString Class Reference 313
4.46 StTableDescriptorI Class Reference 317
4.47 tableAcceptor Class Reference 320
4.48 tableDescriptor Struct Reference 322
4.49 typeAcceptor Class Reference 323

5 Doxygen File Documentation 327

5.1 dbCollection.h File Reference 327
5.2 dbStruct.hh File Reference 328
5.3 MysqlDb.cc File Reference 329
5.4 MysqlDb.h File Reference 333
5.5 parseXmlString.cc File Reference 335

5.6 parseXmlString.hh File Reference . . . 336
5.7 StDataBaseI.cc File Reference . . . 337
5.8 StDataBaseI.hh File Reference . . . 338
5.9 stdb_streams.h File Reference . . . 339
5.10 StDbBuffer.cc File Reference . . . 340
5.11 StDbBuffer.h File Reference . . . 346
5.12 StDbBufferI.h File Reference . . . 348
5.13 StDbConfigNode.cc File Reference . . . 349
5.14 StDbConfigNode.hh File Reference . . . 350
5.15 StDbConfigNodeImpl.cc File Reference . . . 351
5.16 StDbConfigNodeImpl.hh File Reference . . . 352
5.17 StDbDefaults.cc File Reference . . . 353
5.18 StDbDefaults.hh File Reference . . . 354
5.19 StDbDefs.hh File Reference . . . 355
5.20 StDbElementIndex.cc File Reference . . . 358
5.21 StDbElementIndex.hh File Reference . . . 359
5.22 StDbFastSqlWriter.cc File Reference . . . 360
5.23 StDbFastSqlWriter.h File Reference . . . 361
5.24 StDbLogger.hh File Reference . . . 362
5.25 StDbManager.cc File Reference . . . 363
5.26 StDbManager.hh File Reference . . . 364
5.27 StDbManagerImpl.cc File Reference . . . 365
5.28 StDbManagerImpl.hh File Reference . . . 367
5.29 StDbMessenger.cc File Reference . . . 369
5.30 StDbMessenger.hh File Reference . . . 370
5.31 StDbMessService.hh File Reference . . . 371
5.32 StDbModifier.cxx File Reference . . . 372
5.33 StDbModifier.h File Reference . . . 374
5.34 StDbNode.cc File Reference . . . 375
5.35 StDbNode.hh File Reference . . . 376
5.36 StDbServer.cc File Reference . . . 377
5.37 StDbServer.hh File Reference . . . 378
5.38 StDbServerImpl.cc File Reference . . . 379
5.39 StDbServerImpl.hh File Reference . . . 380
5.40 StDbSql.cc File Reference . . . 381
5.41 StDbSql.hh File Reference . . . 383

5.42 StDbStoreInfo.cc File Reference 384
5.43 StDbStoreInfo.hh File Reference 385
5.44 StDbTable.cc File Reference 386
5.45 StDbTable.h File Reference 387
5.46 StDbTableDescriptor.cc File Reference 388
5.47 StDbTableDescriptor.h File Reference 389
5.48 StDbTableFactory.cc File Reference 390
5.49 StDbTableFactory.hh File Reference 391
5.50 StDbTableIter.hh File Reference 392
5.51 StDbTableIterImpl.cc File Reference 393
5.52 StDbTableIterImpl.hh File Reference 394
5.53 StDbTime.h File Reference 395
5.54 StDbXmlReader.cc File Reference 396
5.55 StDbXmlReader.h File Reference 397
5.56 StDbXmlWriter.cc File Reference 398
5.57 StDbXmlWriter.h File Reference 399
5.58 StString.cc File Reference 400
5.59 StString.h File Reference 401
5.60 StTableDescriptorI.h File Reference 402
5.61 tableAcceptor.hh File Reference 403
5.62 typeAcceptor.hh File Reference 404

Chapter 1

Doxygen Hierarchical Index

1.1 Doxygen Class Hierarchy

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

- basic . . . 9
  - accessor . . . 7
    - dbRow . . . 17
  - datav . . . 12
  - dbTable . . . 19
  - elem . . . 22
  - stsize . . . 312
- column . . . 11
- dbCollection . . . 13
- dbDomain . . . 14
- dbEnvList . . . 15
- dbType . . . 21
- indexNameVals . . . 24
- MysqlBin . . . 25
- MysqlDb . . . 26
- MysqlResult . . . 44
- parseXmlString . . . 46
- StDataBaseI . . . 50
  - StDbSql . . . 202
- StDbBufferI . . . 75
  - StDbBuffer . . . 59
- StDbDefaults . . . 104
- StDbElementIndex . . . 109
- StDbLogger . . . 121
- StDbManager . . . 123
  - StDbManagerImpl . . . 135
- StDbMessService . . . 170
  - StDbMessenger . . . 167
- StDbModifier . . . 172
- StDbNode . . . 179
  - StDbConfigNode . . . 82

StDbConfigNodeImpl 93
StDbTable 236
StDbServer 189
StDbServerImpl 196
StDbStoreInfo 233
StDbTableFactory 280
StDbTableIter 281
StDbTableIterImpl 283
StDbTime 286
StString 313
StTableDescriptorI 317
StDbTableDescriptor 268
tableDescriptor 322
typeAcceptor 323
StDbXmlReader 289
tableAcceptor 320
StDbFastSqlWriter 113
StDbXmlWriter 301

Chapter 2

Doxygen Class Index

2.1 Doxygen Class List

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

accessor 7
basic 9
column 11
datav 12
dbCollection 13
dbDomain 14
dbEnvList 15
dbRow 17
dbTable 19
dbType 21
elem 22
indexNameVals 24
MysqlBin 25
MysqlDb 26
MysqlResult 44
parseXmlString 46
StDataBaseI 50
StDbBuffer 59
StDbBufferI 75
StDbConfigNode 82
StDbConfigNodeImpl 93
StDbDefaults 104
StDbElementIndex 109
StDbFastSqlWriter 113
StDbLogger 121
StDbManager 123
StDbManagerImpl 135
StDbMessenger 167
StDbMessService 170
StDbModifier 172
StDbNode 179
StDbServer 189
StDbServerImpl 196

StDbSql 202
StDbStoreInfo 233
StDbTable 236
StDbTableDescriptor 268
StDbTableFactory 280
StDbTableIter 281
StDbTableIterImpl 283
StDbTime 286
StDbXmlReader 289
StDbXmlWriter 301
stsize 312
StString 313
StTableDescriptorI 317
tableAcceptor 320
tableDescriptor 322
typeAcceptor 323

Chapter 3

Doxygen File Index

3.1 Doxygen File List

Here is a list of all files with brief descriptions:

dbCollection.h . . . 327
dbStruct.hh . . . 328
MysqlDb.cc . . . 329
MysqlDb.h . . . 333
parseXmlString.cc . . . 335
parseXmlString.hh . . . 336
StDataBaseI.cc . . . 337
StDataBaseI.hh . . . 338
stdb_streams.h . . . 339
StDbBuffer.cc . . . 340
StDbBuffer.h . . . 346
StDbBufferI.h . . . 348
StDbConfigNode.cc . . . 349
StDbConfigNode.hh . . . 350
StDbConfigNodeImpl.cc . . . 351
StDbConfigNodeImpl.hh . . . 352
StDbDefaults.cc . . . 353
StDbDefaults.hh . . . 354
StDbDefs.hh . . . 355
StDbElementIndex.cc . . . 358
StDbElementIndex.hh . . . 359
StDbFastSqlWriter.cc . . . 360
StDbFastSqlWriter.h . . . 361
StDbLogger.hh . . . 362
StDbManager.cc . . . 363
StDbManager.hh . . . 364
StDbManagerImpl.cc . . . 365
StDbManagerImpl.hh . . . 367
StDbMessenger.cc . . . 369
StDbMessenger.hh . . . 370
StDbMessService.hh . . . 371
StDbModifier.cxx . . . 372
StDbModifier.h . . . 374

StDbNode.cc . . . 375
StDbNode.hh . . . 376
StDbServer.cc . . . 377
StDbServer.hh . . . 378
StDbServerImpl.cc . . . 379
StDbServerImpl.hh . . . 380
StDbSql.cc . . . 381
StDbSql.hh . . . 383
StDbStoreInfo.cc . . . 384
StDbStoreInfo.hh . . . 385
StDbTable.cc . . . 386
StDbTable.h . . . 387
StDbTableDescriptor.cc . . . 388
StDbTableDescriptor.h . . . 389
StDbTableFactory.cc . . . 390
StDbTableFactory.hh . . . 391
StDbTableIter.hh . . . 392
StDbTableIterImpl.cc . . . 393
StDbTableIterImpl.hh . . . 394
StDbTime.h . . . 395
StDbXmlReader.cc . . . 396
StDbXmlReader.h . . . 397
StDbXmlWriter.cc . . . 398
StDbXmlWriter.h . . . 399
StString.cc . . . 400
StString.h . . . 401
StTableDescriptorI.h . . . 402
tableAcceptor.hh . . . 403
typeAcceptor.hh . . . 404

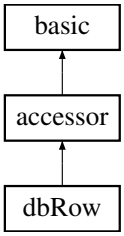
Chapter 4

Doxygen Class Documentation

4.1 accessor Class Reference

`#include <dbStruct.hh>`

Inheritance diagram for accessor::



Public Member Functions

- accessor ()
- virtual ~accessor ()

Public Attributes

- elemVec e
- int nelems

4.1.1 Constructor & Destructor Documentation

4.1.1.1 accessor::accessor () [inline]

Definition at line 105 of file dbStruct.hh.

```
105                 { setStartKey("<StDbAccessor>");
106                   setEndKey("</StDbAccessor>");};
```

4.1.1.2 virtual accessor::~accessor () [inline, virtual]

Definition at line 107 of file dbStruct.hh.

```
107 {};
```

4.1.2 Member Data Documentation

4.1.2.1 elemVec accessor::e

Definition at line 102 of file dbStruct.hh.

4.1.2.2 int accessor::nelems

Definition at line 103 of file dbStruct.hh.

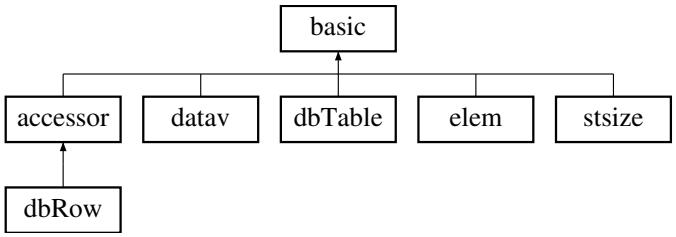
The documentation for this class was generated from the following file:

- dbStruct.hh

4.2 basic Class Reference

`#include <dbStruct.hh>`

Inheritance diagram for basic::



Public Member Functions

- virtual ~basic ()
- virtual void setStartKey (const char *key)
- virtual void setEndKey (const char *key)

Public Attributes

- char startKey [20]
- char endKey [20]
- int istart
- int iend

4.2.1 Constructor & Destructor Documentation

4.2.1.1 virtual basic::~basic () `[inline, virtual]`

Definition at line 44 of file dbStruct.hh.

```
44 {};
```

4.2.2 Member Function Documentation

4.2.2.1 virtual void basic::setEndKey (const char * *key*) `[inline, virtual]`

Definition at line 50 of file dbStruct.hh.

```
50 { strcpy(endKey,key); };
```

4.2.2.2 virtual void basic::setStartKey (const char * *key*) `[inline, virtual]`

Definition at line 49 of file dbStruct.hh.

```
49 { strcpy(startKey,key); };
```

4.2.3 Member Data Documentation

4.2.3.1 char basic::endKey[20]

Definition at line 46 of file dbStruct.hh.

4.2.3.2 int basic::iend

Definition at line 48 of file dbStruct.hh.

4.2.3.3 int basic::istart

Definition at line 47 of file dbStruct.hh.

4.2.3.4 char basic::startKey[20]

Definition at line 45 of file dbStruct.hh.

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

- dbStruct.hh

4.3 column Struct Reference

```
#include <StDbBuffer.h>
```

Public Attributes

- char * name
- enum myctype type
- char * val
- unsigned length

4.3.1 Member Data Documentation

4.3.1.1 unsigned column::length

Definition at line 68 of file StDbBuffer.h.

4.3.1.2 char* column::name

Definition at line 65 of file StDbBuffer.h.

4.3.1.3 enum myctype column::type

Definition at line 66 of file StDbBuffer.h.

4.3.1.4 char* column::val

Definition at line 67 of file StDbBuffer.h.

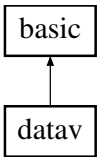
The documentation for this struct was generated from the following file:

- StDbBuffer.h

4.4 datav Class Reference

`#include <dbStruct.hh>`

Inheritance diagram for datav::



Public Member Functions

- datav ()

Public Attributes

- char * data

4.4.1 Constructor & Destructor Documentation

4.4.1.1 datav::datav () [inline]

Definition at line 69 of file dbStruct.hh.

```
69            { setStartKey("<value>");
70    setEndKey("</value>") ;};
```

4.4.2 Member Data Documentation

4.4.2.1 char* datav::data

Definition at line 68 of file dbStruct.hh.

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

- dbStruct.hh

4.5 dbCollection Struct Reference

```
#include <dbCollection.h>
```

Public Attributes

- char name [64]

4.5.1 Member Data Documentation

4.5.1.1 char dbCollection::name[64]

Definition at line 30 of file dbCollection.h.

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

- dbCollection.h

4.6 dbDomain Class Reference

```
#include <StDbManagerImpl.hh>
```

Public Member Functions

- dbDomain (StDbDomain adom, const char *aname)

Public Attributes

- StDbDomain domain
- char name [64]

4.6.1 Constructor & Destructor Documentation

4.6.1.1 dbDomain::dbDomain (StDbDomain *adom*, const char * *aname*) [inline]

Definition at line 67 of file StDbManagerImpl.hh.

```
67                                                  : domain(adom) {
68                                                    if(aname)strcpy(name,aname);};
```

4.6.2 Member Data Documentation

4.6.2.1 StDbDomain dbDomain::domain

Definition at line 65 of file StDbManagerImpl.hh.

4.6.2.2 char dbDomain::name[64]

Definition at line 66 of file StDbManagerImpl.hh.

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

- StDbManagerImpl.hh

4.7 dbEnvList Class Reference

```
#include <StDbManagerImpl.hh>
```

Public Member Functions

- dbEnvList ()
- ~dbEnvList ()

Public Attributes

- int num
- char * envVar [MAX_ENV_VARS]
- char * envDef [MAX_ENV_VARS]

4.7.1 Constructor & Destructor Documentation

4.7.1.1 dbEnvList::dbEnvList () [inline]

Definition at line 74 of file StDbManagerImpl.hh.

```
74 : num(0) {};
```

4.7.1.2 dbEnvList::~dbEnvList () [inline]

Definition at line 75 of file StDbManagerImpl.hh.

```
75                {
76     for(int i=0;i<num;i++){
77       delete [] envVar[i];
78       delete [] envDef[i];
79     }
80   }
```

4.7.2 Member Data Documentation

4.7.2.1 char* dbEnvList::envDef[MAX_ENV_VARS]

Definition at line 83 of file StDbManagerImpl.hh.

4.7.2.2 char* dbEnvList::envVar[MAX_ENV_VARS]

Definition at line 82 of file StDbManagerImpl.hh.

4.7.2.3 int dbEnvList::num

Definition at line 81 of file StDbManagerImpl.hh.

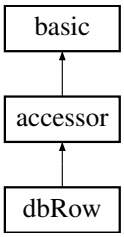
The documentation for this class was generated from the following file:

- StDbManagerImpl.hh

4.8 dbRow Class Reference

`#include <dbStruct.hh>`

Inheritance diagram for dbRow::



Public Member Functions

- dbRow ()
- virtual ~dbRow ()

Public Attributes

- int rowNumber
- int rowID

4.8.1 Constructor & Destructor Documentation

4.8.1.1 dbRow::dbRow () [inline]

Definition at line 120 of file dbStruct.hh.

```
120          { setStartKey("<TabRow>");
121            setEndKey("</TabRow>"); };
```

4.8.1.2 virtual dbRow::~dbRow () [inline, virtual]

Definition at line 122 of file dbStruct.hh.

```
122 {};
```

4.8.2 Member Data Documentation

4.8.2.1 int dbRow::rowID

Definition at line 118 of file dbStruct.hh.

4.8.2.2 int dbRow::rowNumber

Definition at line 117 of file dbStruct.hh.

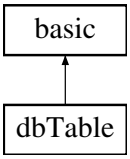
The documentation for this class was generated from the following file:

- dbStruct.hh

4.9 dbTable Class Reference

`#include <dbStruct.hh>`

Inheritance diagram for dbTable::



Public Member Functions

- dbTable ()
- virtual ~dbTable ()

Public Attributes

- accessor a
- rowVec row
- int curRow
- int numRows
- char * name

4.9.1 Constructor & Destructor Documentation

4.9.1.1 dbTable::dbTable () [inline]

Definition at line 143 of file dbStruct.hh.

```
143             { setStartKey("<StDbTable>");
144               setEndKey("</StDbTable>");};
```

4.9.1.2 virtual dbTable::~dbTable () [inline, virtual]

Definition at line 145 of file dbStruct.hh.

```
145 {};
```

4.9.2 Member Data Documentation

4.9.2.1 accessor dbTable::a

Definition at line 135 of file dbStruct.hh.

4.9.2.2 int dbTable::curRow

Definition at line 137 of file dbStruct.hh.

4.9.2.3 char* dbTable::name

Definition at line 141 of file dbStruct.hh.

4.9.2.4 int dbTable::numRows

Definition at line 138 of file dbStruct.hh.

4.9.2.5 rowVec dbTable::row

Definition at line 136 of file dbStruct.hh.

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

- dbStruct.hh

4.10 dbType Class Reference

```
#include <StDbManagerImpl.hh>
```

Public Member Functions

- dbType (StDbType atype, const char *aname)

Public Attributes

- StDbType type
- char name [64]

4.10.1 Constructor & Destructor Documentation

4.10.1.1 dbType::dbType (StDbType *atype*, const char * *aname*) [inline]

Definition at line 60 of file StDbManagerImpl.hh.

```
60                                          : type(atype) {
61                                           if(aname)strcpy(name,aname);};
```

4.10.2 Member Data Documentation

4.10.2.1 char dbType::name[64]

Definition at line 59 of file StDbManagerImpl.hh.

4.10.2.2 StDbType dbType::type

Definition at line 58 of file StDbManagerImpl.hh.

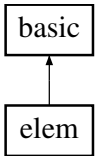
The documentation for this class was generated from the following file:

- StDbManagerImpl.hh

4.11 elem Class Reference

`#include <dbStruct.hh>`

Inheritance diagram for elem::



Public Member Functions

- elem ()

Public Attributes

- datav val
- char * name
- char * type
- stsize size

4.11.1 Constructor & Destructor Documentation

4.11.1.1 elem::elem () [inline]

Definition at line 84 of file dbStruct.hh.

```
84          { setStartKey("<db");
85            setEndKey("</db") ;};
```

4.11.2 Member Data Documentation

4.11.2.1 char* elem::name

Definition at line 80 of file dbStruct.hh.

4.11.2.2 stsize elem::size

Definition at line 82 of file dbStruct.hh.

4.11.2.3 char* elem::type

Definition at line 81 of file dbStruct.hh.

4.11.2.4 datav elem::val

Definition at line 79 of file dbStruct.hh.

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

- dbStruct.hh

4.12 indexNameVals Struct Reference

```
#include <StDbElementIndex.hh>
```

Public Attributes

- char iname [64]
- int ival

4.12.1 Member Data Documentation

4.12.1.1 char indexNameVals::iname[64]

Definition at line 40 of file StDbElementIndex.hh.

4.12.1.2 int indexNameVals::ival

Definition at line 41 of file StDbElementIndex.hh.

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

- StDbElementIndex.hh

4.13 MysqlBin Class Reference

```
#include <MysqlDb.h>
```

Public Member Functions

- MysqlBin ()
- virtual void Input (const unsigned long int aLen, char *aBin)

Private Attributes

- char * mBinData
- unsigned long int mLen

4.13.1 Constructor & Destructor Documentation

4.13.1.1 MysqlBin::MysqlBin () [inline]

Definition at line 180 of file MysqlDb.h.

```
180 {mBinData=0;mLen=0;};
```

4.13.2 Member Function Documentation

4.13.2.1 virtual void MysqlBin::Input (const unsigned long int *aLen*, char * *aBin*) [inline, virtual]

Definition at line 181 of file MysqlDb.h.

```
182     {mBinData=aBin;mLen=aLen;};
```

4.13.3 Member Data Documentation

4.13.3.1 char* MysqlBin::mBinData [private]

Definition at line 176 of file MysqlDb.h.

4.13.3.2 unsigned long int MysqlBin::mLen [private]

Definition at line 177 of file MysqlDb.h.

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

- MysqlDb.h

4.14 MysqlDb Class Reference

```
#include <MysqlDb.h>
```

Public Member Functions

- MysqlDb ()
- virtual ~MysqlDb ()
- virtual bool Connect (const char *aHost, const char *aUser, const char *aPasswd, const char *aDb, const int aPort=0)
- virtual bool reConnect ()
- virtual unsigned NbRows ()
- virtual unsigned NbFields ()
- virtual void Release ()
- virtual char * printQuery ()
- virtual bool InputStart (const char *aName, StDbBuffer *aBuff, const char *colList, int nRows, bool &hasBinary)
- virtual bool InputRow (StDbBuffer *aBuff, int row)
- virtual bool InputEnd ()
- virtual bool Input (const char *aName, StDbBuffer *aBuff)
- virtual bool Output (StDbBuffer *aBuff)
- MysqlDb & operator<< (const char *c)
- MysqlDb & operator<< (const string s)
- MysqlDb & operator<< (const MysqlBin *aBin)
- MysqlDb & operator<< (const short aq)
- MysqlDb & operator<< (const unsigned short aq)
- MysqlDb & operator<< (const int aq)
- MysqlDb & operator<< (const unsigned int aq)
- MysqlDb & operator<< (const long long aq)
- MysqlDb & operator<< (const float aq)
- MysqlDb & operator<< (const double aq)
- char ** DecodeStrArray (char *strinput, int &aLen)
- char * CodeStrArray (char **strarr, int aLen)
- virtual int GetLastInsertID ()
- virtual bool QueryStatus ()
- virtual bool checkForTable (const char *tableName)
- virtual void Close ()
- virtual bool IsConnected ()
- virtual bool setDefaultDb (const char *dbName)

Public Attributes

- bool mlogTime
- StDbLogger mqueryLog
- StDbLogger msocketLog
- StDbLogger mconnectLog
- StDbManagerImpl * my_manager

Protected Member Functions

- virtual void RazQuery ()
- virtual bool ExecQuery ()

Private Member Functions

- std::vector< std::string >::iterator RecommendedServer (std::vector< std::string > *ListToUse, char *socket, int port)

Private Attributes

- MYSQL mData
- char * mQuery
- unsigned long int mQueryLen
- char * mQueryLast
- MysqlResult * mRes
- bool mqueryState
- bool mhasConnected
- bool mhasBinaryQuery
- char * mdbhost
- char * mdbName
- char * mdbuser
- char * mdbpw
- int mdbPort
- char * mdbServerVersion
- unsigned int mtimeout
- int jfields
- char * cnames [200]
- bool isBlob [200]
- bool isBinary [200]
- bool isSpecialType [200]

4.14.1 Constructor & Destructor Documentation

4.14.1.1 MysqlDb::MysqlDb ()

Definition at line 213 of file MysqlDb.cc.

```
213                 : mdbhost(0), mdbName(NULL), mdbuser(0), mdbpw(0), mdbPort(0),mdbServerVersion(0),mlog
214
215 mhasConnected=false;
216 mhasBinaryQuery=false;
217 mtimeout=1;
218 mQuery=0;
219 mQueryLast=0;
220 mRes= new MysqlResult;
221  for(int i=0;i<200;i++)cnames[i]=0;
222 }
```

4.14.1.2 MysqlDb::~MysqlDb () [virtual]

Definition at line 225 of file MysqlDb.cc.

```
225                 {
226 if(mQuery) delete [] mQuery;
227 if(mQueryLast) delete [] mQueryLast;
228 Release();
229 if(mRes) delete mRes;
230 if(mhasConnected)mysql_close(&mData);
231 if(mdbhost) delete [] mdbhost;
232 if(mdbuser) delete [] mdbuser;
233 if(mdbpw)   delete [] mdbpw;
234 if(mdbName)  delete [] mdbName;
235  if(mdbServerVersion) delete [] mdbServerVersion;
236
237 }
```

4.14.2 Member Function Documentation

4.14.2.1 bool MysqlDb::checkForTable (const char * *tableName*) [virtual]

Definition at line 429 of file MysqlDb.cc.

```
429                                                 {
430
431    mRes->Release();
432    mRes->mRes=mysql_list_tables(&mData,tableName);
433    if(mRes->mRes==NULL) return false;
434    mRes->Release();
435
436 return true;
437 };
```

4.14.2.2 void MysqlDb::Close () [inline, virtual]

Definition at line 286 of file MysqlDb.h.

```
286                          {
287   if(mhasConnected)mysql_close(&mData);
288   mhasConnected=false;
289 }
```

4.14.2.3 char * MysqlDb::CodeStrArray (char ** *strarr*, int *aLen*)

Definition at line 829 of file MysqlDb.cc.

```
829                                                 {
830   int tMaxLen=0;
831   int i;
832   for (i=0;i<aLen;i++) {
833     if (strarr[i]) tMaxLen=tMaxLen+strlen(strarr[i])*2;
834     tMaxLen++;
835   };
836   char* tTempVal=new char[tMaxLen+1];
837   char* tRead;
```

```
838   char* tWrite=tTempVal;
839   for (i=0;i<aLen;i++) {
840     if (strarr[i]){
841       int j;
842       tRead=strarr[i];
843       for (j=0;j<(int)strlen(strarr[i]);j++) {
844         if (*tRead=='\\'||*tRead==',') {
845           *tWrite='\\';
846           tWrite++;
847         };
848         *tWrite=*tRead;
849         tWrite++;
850         tRead++;
851       };
852     };
853     *tWrite=',';
854     tWrite++;
855   };
856   tWrite--;
857   *tWrite='\0';
858   char *tRetVal=new char[strlen(tTempVal)+1];
859   strcpy(tRetVal,tTempVal);
860   if (tTempVal) delete [] tTempVal;
861   return tRetVal;
862 };
```

4.14.2.4 bool MysqlDb::Connect (const char ∗ *aHost*, const char ∗ *aUser*, const char ∗ *aPasswd*, const char ∗ *aDb*, const int *aPort* = 0) [virtual]

Definition at line 328 of file MysqlDb.cc.

```
328
329 #define __METHOD__ "Connect(host,user,pw,database,port)"
330
331   if(mdbhost) delete [] mdbhost;
332   mdbhost  = new char[strlen(aHost)+1];
333 strcpy(mdbhost,aHost);
334
335   if(aUser){
336    if(mdbuser) delete [] mdbuser;
337    mdbuser  = new char[strlen(aUser)+1];   strcpy(mdbuser,aUser);
338    }
339   if(aPasswd){
340   if(mdbpw) delete [] mdbpw;
341     mdbpw    = new char[strlen(aPasswd)+1]; strcpy(mdbpw,aPasswd);
342   }
343   mdbPort  = aPort;
344   //  cout << "aHost = "<<mdbhost<<endl;
345   //cout << " Calling load balancer\n";
346   clock_t start,finish;
347  double lbtime;
348   start = clock();
349
350   std::vector<std::string>::iterator  myserver = RecommendedServer(&(my_manager->xmlServerList), NULL,
351   strcpy(mdbhost,(*myserver).c_str());
352
353   finish = clock();
354   lbtime = (double(finish)-double(start))/CLOCKS_PER_SEC*1000;
355   cout << " Load balancer took "<<lbtime<<" ms, will use "<<mdbhost<<" \n";
356
357   if(mdbName) {
358     delete [] mdbName;
359     mdbName=NULL;
360   }
```

```
361   if(aDb && (strcmp(aDb," ")!=0)){
362     mdbName  = new char[strlen(aDb)+1];
363     strcpy(mdbName,aDb);
364   }
365
366   bool tRetVal = false;
367   double t0=mqueryLog.wallTime();
368   if(mlogTime)mconnectLog.start();
369   if (!mysql_init(&mData))
370     return (bool) StDbManager::Instance()->printInfo("Mysql Init Error=",mysql_error(&mData),dbMErr,__
371
372   // char *connString;
373   StString cs;
374   if(reConnect()){
375     //  if(mysql_real_connect(&mData,aHost,aUser,aPasswd,bDb,aPort,NULL,0)){
376        t0=mqueryLog.wallTime()-t0;
377        cs<< "Server Connecting:"; if(mdbName)cs<<" DB=" << mdbName ;
378        cs<< "  Host=" << mdbhost <<":"<<aPort <<stendl;
379        cs<< " --> Connection Time="<<t0<<" sec   ";
380        if(mdbServerVersion)cs<<" MysqlVersion="<<mdbServerVersion;
381
382         StDbManager::Instance()->printInfo((cs.str()).c_str(),dbMConnect,__LINE__,__CLASS__,__METHOD__
383       tRetVal=true;
384   } else {
385       cs << "Making Connection to DataBase = " << aDb;
386       cs << " On Host = " << mdbhost <<":"<<aPort;
387       cs << " MySQL returned error " << mysql_error(&mData);
388       StDbManager::Instance()->printInfo((cs.str()).c_str(),dbMConnect,__LINE__,__CLASS__,__METHOD__);
389   }
390
391   if(mlogTime)mconnectLog.end();
392   mhasConnected = tRetVal;
393   return tRetVal;
394 #undef __METHOD__
395 }
```

4.14.2.5 char ** MysqlDb::DecodeStrArray (char * *strinput*, int & *aLen*)

Definition at line 767 of file MysqlDb.cc.

```
767                                                         {
768
769   if(!strinput){ // shouldn't happen - should have checked before here
770     //   cout<< "null input string from mysql " << endl;
771     char** tmparr = new char*[1];
772     aLen = 1;
773     *tmparr = new char[2];
774     strcpy(*tmparr,"0");
775     return tmparr;
776   }
777
778   char* tPnt=strinput;
779   aLen=0;
780   while (tPnt&&aLen<100) { // 100 is a limit on # comma separated values
781     tPnt=strpbrk( tPnt,"\\,");
782     if (tPnt!=0){
783       if (*tPnt==',') {
784         aLen++;tPnt++;
785       } else { //(*tPnt=='\\')
786         tPnt++;tPnt++;};
787     };
788   };
789   aLen++;
790   char** strarr=new char*[aLen];
```

```
  tPnt=strinput;
  char* tPntNew=tPnt;
  char *tBuff=new char[strlen(strinput)+1];
  char *tBuffInd=tBuff;
  int tCount=0;
  while (tPntNew) {
    tPntNew=strpbrk( tPnt,"\\,");
    if ((tPntNew==0)||*tPntNew==',') {
      if (tPntNew==0) {
        strcpy(tBuffInd,tPnt);

      } else {
        strncpy(tBuffInd,tPnt,tPntNew-tPnt);
        *(tBuffInd+(tPntNew-tPnt))='\0';
      }
      strarr[tCount]=new char[strlen(tBuff)+1];
      strcpy(strarr[tCount],tBuff);
      tBuffInd=tBuff;
      tPnt=tPntNew+1;
      tCount++;
    } else { //(*tPntNew=='\\')
      strncpy(tBuffInd,tPnt,tPntNew-tPnt);
      tBuffInd=tBuffInd+(tPntNew-tPnt);
      tPntNew++;
      if(*tPntNew=='\\'||*tPntNew==',') {
        *tBuffInd=*tPntNew;
        tBuffInd++;
      };
      *(tBuffInd)='\0';
      tPnt=tPntNew+1;
    };
  };
  delete [] tBuff;
  return strarr;
}
```

4.14.2.6 bool MysqlDb::ExecQuery () [protected, virtual]

Definition at line 441 of file MysqlDb.cc.

```
                        {
#define __METHOD__ "ExecQuery()"

mqueryState=false;

// cout<<"MysqlDb::ExecQuery() ";
// cout<<mQuery<<endl;

if(mlogTime)mqueryLog.start();

int status=mysql_real_query(&mData,mQuery,mQueryLen);

  if( (status!=0) && ( mysql_errno(&mData)==CR_SERVER_GONE_ERROR || mysql_errno(&mData)==CR_SERVER_LOS
       StDbManager::Instance()->printInfo(mysql_error(&mData)," Lost server, will try to reconnect",db
       if(reConnect())status=mysql_real_query(&mData,mQuery,mQueryLen);
  }

  if(status!=0)
    return StDbManager::Instance()->printInfo(" Query Failed ",mysql_error(&mData),dbMErr,__LINE__,__C

  if(mlogTime)mqueryLog.end();
  mRes->Release();

  if(mlogTime)msocketLog.start();
```

```
  mRes->mRes=mysql_store_result(&mData);
  if(mlogTime)msocketLog.end();

  return mqueryState=true;
}
```

4.14.2.7 virtual int MysqlDb::GetLastInsertID () [inline, virtual]

Definition at line 264 of file MysqlDb.h.

```
{ return (int)mysql_insert_id(&mData);}
```

4.14.2.8 bool MysqlDb::Input (const char * *aName*, StDbBuffer * *aBuff*) [virtual]

Definition at line 627 of file MysqlDb.cc.

```
                                                        {
  bool tRetVal=false;
  bool change=aBuff->IsClientMode();
  if (change) aBuff->SetStorageMode();
  aBuff->SetStorageMode();
  if (aBuff) {
    *this << "select * from " << table << " where null"<< endsql;
    *this << "insert into " << table << " set ";
    bool tFirst=true;
    char* tVal;tVal=0;
    int len;
    unsigned i;

 for (i=0;i<NbFields();i++) {
    if ((IS_BLOB(mRes->mRes->fields[i].flags) ) ||
        mRes->mRes->fields[i].type ==254) {
      if (mRes->mRes->fields[i].flags&BINARY_FLAG) { // Binary
            if (aBuff->ReadArray(tVal,len,mRes->mRes->fields[i].name)){
               if (tFirst) {
               tFirst=false;
           } else {
               *this << ",";
           };
                 *this << mRes->mRes->fields[i].name << "='" << Binary(len,(float*)tVal)<<"'";
           };
         }else{  // text types
       if(mRes->mRes->fields[i].type==254){
             if (aBuff->ReadScalar(tVal,mRes->mRes->fields[i].name)) {
               if (tFirst) {
               tFirst=false;
           } else {
               *this << ",";
           };
               *this << mRes->mRes->fields[i].name << "='" << tVal << "'";
             };
       } else {
             char** tVal2=0;
             if (aBuff->ReadArray(tVal2,len,mRes->mRes->fields[i].name)){
               tVal=CodeStrArray(tVal2,len);
               int j;for (j=0;j<len;j++) {if (tVal2[j]) delete [] tVal2[j];};
               delete [] tVal2;
               if (tFirst) {
               tFirst=false;
           } else {
               *this << ",";
```

```
            };
                *this << mRes->mRes->fields[i].name << "='" << tVal<<"'";
          }
        };
         };
   } else {  // not binary nor text
               if (aBuff->ReadScalar(tVal,mRes->mRes->fields[i].name)) {
                  if (tFirst) {
                  tFirst=false;
             } else {
                  *this << ",";
             };
                  *this << mRes->mRes->fields[i].name << "='" << tVal << "'";
               };
       };
       if (tVal) delete [] tVal;tVal=0;
     };
     if (tFirst) {
       RazQuery();
     } else {
       *this << endsql;
       if(mqueryState)tRetVal=true;
     };
   };
   //  if (!tRetVal) cout << "insert Failed"<< endl;
   if (change) aBuff->SetClientMode();
   aBuff->SetClientMode();
   return tRetVal;
 };
```

4.14.2.9 bool MysqlDb::InputEnd () [virtual]

Definition at line 617 of file MysqlDb.cc.

```
                      {

 bool tRetVal=false;
 *this<<")"<<endsql;
 if(mqueryState)tRetVal=true;
   return tRetVal;

 };
```

4.14.2.10 bool MysqlDb::InputRow (StDbBuffer * *aBuff*, int *row*) [virtual]

Definition at line 578 of file MysqlDb.cc.

```
                                                {

   char* tVal;tVal=0;
   int len;
   aBuff->SetStorageMode();
       if(row>0)*this<<"),(";
       int k;
        for(k=0;k<jfields;k++){
          if(k!=0)*this<<",";
          if(isBlob[k]){
            if(isBinary[k]){
              if(!aBuff->ReadArray(tVal,len,cnames[k]))break;
              *this<<"'"<<Binary(len,(float*)tVal)<<"'";
            } else if(isSpecialType[k]) {
```

```
592             if(!aBuff->ReadScalar(tVal,cnames[k]))break;
593             *this<<"'"<<tVal<<"'";
594           } else {
595             char** tVal2=0;
596             if(!aBuff->ReadArray(tVal2,len,cnames[k]))break;
597             tVal=CodeStrArray(tVal2,len);
598             for(int jj=0;jj<len;jj++)if(tVal2[jj])delete []tVal2[jj];
599             *this<<"'"<<tVal<<"'";
600           }
601         } else {
602           if(!aBuff->ReadScalar(tVal,cnames[k])) break;
603           *this<<"'"<<tVal<<"'";
604         }
605       }
606
607       aBuff->SetClientMode();
608
609     if(k!=jfields){
610       RazQuery();
611       return false;
612     }
613
614     return true;
615 }
```

4.14.2.11 bool MysqlDb::InputStart (const char * *aName*, StDbBuffer * *aBuff*, const char * *colList*, int *nRows*, bool & *hasBinary*) [virtual]

Definition at line 519 of file MysqlDb.cc.

```
519
520
521   bool tRetVal=false;
522   if(!table || !aBuff || !colList) return tRetVal;
523
524   bool change=aBuff->IsClientMode();
525   if(change) aBuff->SetStorageMode();
526
527   *this << "select * from " << table << " where null"<< endsql;
528   *this << "insert delayed into " << table << " ("<<colList<<") VALUES(";
529   int i;
530
531     char* tmpString=new char[strlen(colList)+1];
532     strcpy(tmpString,colList);
533     char *ptr1,*ptr2;
534     jfields=0;
535     bool done = false;
536     ptr1=tmpString;
537
538     while(!done){
539       if((ptr2=strstr(ptr1,",")){
540         *ptr2='\0';
541       } else {
542         done=true;
543       }
544       if(*ptr1==' ')ptr1++;
545       if(cnames[jfields]) delete [] cnames[jfields];
546       cnames[jfields]=new char[strlen(ptr1)+1];
547       strcpy(cnames[jfields],ptr1);
548       if(!done){
549         ptr1=ptr2+1;
550         *ptr2=',';
551       }
552       jfields++;
```

```
553     }
554     delete [] tmpString;
555     int nfields=NbFields();
556     int fcount=0;
557     hasBinary=false;
558
559     for(int k=0;k<jfields;k++){
560       for(i=0;i<nfields;i++)
561         if(strcmp(mRes->mRes->fields[i].name,cnames[k])==0)break;
562
563       if(i==nfields)continue;
564       fcount++;
565       isBlob[k]=( (IS_BLOB(mRes->mRes->fields[i].flags)) ||
566                   (mRes->mRes->fields[i].type ==254) );
567       isBinary[k]= (mRes->mRes->fields[i].flags&BINARY_FLAG);
568       isSpecialType[k]=(mRes->mRes->fields[i].type ==254);
569
570       if(isBinary[k])hasBinary=true;
571     }
572     if(fcount!=jfields) done=false;
573
574     return done;
575
576 };
```

4.14.2.12 virtual bool MysqlDb::IsConnected () [inline, virtual]

Definition at line 268 of file MysqlDb.h.

```
268 { return mhasConnected; }
```

4.14.2.13 virtual unsigned MysqlDb::NbFields () [inline, virtual]

Definition at line 235 of file MysqlDb.h.

```
235 {if(mqueryState)return mRes->NbFields(); return 0;};
```

4.14.2.14 virtual unsigned MysqlDb::NbRows () [inline, virtual]

Definition at line 234 of file MysqlDb.h.

```
234 {if(mqueryState)return mRes->NbRows(); return 0;};
```

4.14.2.15 MysqlDb & MysqlDb::operator<< (const double *aq*) [inline]

Definition at line 325 of file MysqlDb.h.

```
325                                                    {
326   StString ts;  ts<<aq;
327   return *this<<ts.str();
328 }
```

4.14.2.16 MysqlDb & MysqlDb::operator<< (const float *aq*) [inline]

Definition at line 320 of file MysqlDb.h.

```
320                                                 {
321   StString ts;  ts<<aq;
322   return *this<<ts.str();
323 }
```

4.14.2.17 MysqlDb & MysqlDb::operator<< (const long long *aq*) [inline]

Definition at line 315 of file MysqlDb.h.

```
315                                                     {
316   StString ts;  ts<<aq;
317   return *this<<ts.str();
318 }
```

4.14.2.18 MysqlDb & MysqlDb::operator<< (const unsigned int *aq*) [inline]

Definition at line 310 of file MysqlDb.h.

```
310                                                       {
311   StString ts;  ts<<aq;
312   return *this<<ts.str();
313 }
```

4.14.2.19 MysqlDb & MysqlDb::operator<< (const int *aq*) [inline]

Definition at line 305 of file MysqlDb.h.

```
305                                                 {
306   StString ts;  ts<<aq;
307   return *this<<ts.str();
308 }
```

4.14.2.20 MysqlDb & MysqlDb::operator<< (const unsigned short *aq*) [inline]

Definition at line 300 of file MysqlDb.h.

```
300                                                       {
301   StString ts;  ts<<aq;
302   return *this<<ts.str();
303 }
```

4.14.2.21 MysqlDb & MysqlDb::operator<< (const short *aq*) [inline]

Definition at line 295 of file MysqlDb.h.

```
295                                                 {
296   StString ts;  ts<<aq;
297   return *this<<ts.str();
298 }
```

4.14.2.22 MysqlDb & MysqlDb::operator<< (const MysqlBin * *aBin*)

Definition at line 502 of file MysqlDb.cc.

```
502                                                 {
503
504   mhasBinaryQuery=true;
505
506   char *tQuery = new char[mQueryLen+aBin->mLen+1];
507   memcpy(tQuery,mQuery,mQueryLen);
508   memcpy(&tQuery[mQueryLen],aBin->mBinData,aBin->mLen+1); // mBinData included null terminator
509   //  tQuery[mQueryLen+aBin->mLen]='\0';
510   if(mQuery)delete [] mQuery;
511   mQuery=tQuery;
512   mQueryLen=mQueryLen+aBin->mLen;  // always not include null terminator
513
514   return *this;
515 };
```

4.14.2.23 MysqlDb & MysqlDb::operator<< (const string *s*) [inline]

Definition at line 291 of file MysqlDb.h.

```
291                                                 {
292   return *this<<s.c_str();
293 }
```

4.14.2.24 MysqlDb & MysqlDb::operator<< (const char * *c*)

Definition at line 473 of file MysqlDb.cc.

```
473                                                 {
474
475   if (strcmp(aQuery,";")==0){
476     ExecQuery();
477     RazQuery();
478   } else {
479
480     if(!mQuery){
481       mQueryLen=strlen(aQuery);
482       mQuery = new char[mQueryLen+1];
483       strcpy(mQuery,aQuery);
484     } else {
485      char* tQuery = new char[strlen(mQuery)+1];
486      strcpy(tQuery,mQuery);
487      delete [] mQuery;
488      mQuery = new char[mQueryLen+strlen(aQuery)+1];
489      memcpy(mQuery,tQuery,mQueryLen);
490      strcpy(&mQuery[mQueryLen],aQuery);
491      delete [] tQuery;
492      mQueryLen=mQueryLen+strlen(aQuery);
493     }
494
495   }
496
497   return *this;
498 }
```

4.14.2.25 bool MysqlDb::Output (StDbBuffer * *aBuff*) [virtual]

Definition at line 704 of file MysqlDb.cc.

```
704                                     {
705
706   if(mlogTime)msocketLog.start();
707   MYSQL_ROW tRow=mysql_fetch_row(mRes->mRes);
708   if(!tRow) return false;
709   unsigned long * lengths=mysql_fetch_lengths(mRes->mRes);
710   unsigned tNbFields=NbFields();
711   if(mlogTime)msocketLog.end();
712   int i;
713   bool tRetVal=false;
714   bool change=aBuff->IsClientMode();
715   if (change) aBuff->SetStorageMode();
716   aBuff->SetStorageMode();
717
718   for (i=0;i<(int)tNbFields;i++){
719     if(tRow[i]){
720
721       //      cout <<mRes->mRes->fields[i].name<<" = "<< tRow[i] << endl;
722       if (IS_BLOB(mRes->mRes->fields[i].flags)) {
723             if (mRes->mRes->fields[i].flags&BINARY_FLAG) {
724                aBuff->WriteArray((char*)tRow[i],lengths[i],mRes->mRes->fields[i].name);
725             }else {
726                char** tStrPtr;
727                int len;
728                tStrPtr=DecodeStrArray((char*)tRow[i],len);
729                aBuff->WriteArray(tStrPtr,len,mRes->mRes->fields[i].name);
730            for(int k=0;k<len;k++)delete [] tStrPtr[k];
731            delete [] tStrPtr;
732
733            // something of a cludge: "," are used in "Decode.."
734            // as array delimeters. But we also want to have
735            // char arrays for comment structures - so write'em both
736            // to the buffer.
737
738            //char commentName[1024];
739            StString cn;
740            cn<<mRes->mRes->fields[i].name<<".text";
741                aBuff->WriteScalar((char*)tRow[i],(cn.str()).c_str());
742             };
743       } else {
744                aBuff->WriteScalar((char*)tRow[i],mRes->mRes->fields[i].name);
745       };
746     }
747
748     /*
749        else {
750
751       StString nd;
752       nd<<"null data returned from table = ";
753       nd<<mRes->mRes->fields[i].table<<" column="<<mRes->mRes->fields[i].name;
754       StDbManager::Instance()->printInfo((nd.str()).c_str(),dbMWarn,__LINE__,"MysqlDb","Output(StDbBuf
755     }
756     */
757
758     tRetVal=true;
759   };
760   if (change) aBuff->SetClientMode();
761   aBuff->SetClientMode();
762   return tRetVal;
763 }
```

4.14.2.26 char * MysqlDb::printQuery () [virtual]

Definition at line 401 of file MysqlDb.cc.

```
401 { return mQueryLast; };
```

4.14.2.27 virtual bool MysqlDb::QueryStatus () [inline, virtual]

Definition at line 265 of file MysqlDb.h.

```
265 { return mqueryState; }
```

4.14.2.28 void MysqlDb::RazQuery () [protected, virtual]

Definition at line 404 of file MysqlDb.cc.

```
404                       {
405
406   if (mQueryLast){
407     delete [] mQueryLast;
408     mQueryLast=0;
409   }
410   if(mhasBinaryQuery){
411      mQueryLast = new char[strlen(binaryMessage)+1];
412      strcpy(mQueryLast,binaryMessage);
413      if(mQuery)delete [] mQuery;
414   } else {
415      if(mQuery){
416        mQueryLast=new char[strlen(mQuery)+1];
417        strcpy(mQueryLast,mQuery);
418        delete [] mQuery;
419      }
420   }
421
422   mQuery = 0;//new char[1];
423   mQueryLen=0;
424   //  strcpy(mQuery,"");
425
426   mhasBinaryQuery=false;
427 }
```

4.14.2.29 std::vector<std::string>::iterator MysqlDb::RecommendedServer (std::vector< std::string > * *ListToUse*, char * *socket*, int *port*) [private]

4.14.2.30 bool MysqlDb::reConnect () [virtual]

Definition at line 287 of file MysqlDb.cc.

```
287                       {
288 #define __METHOD__ "reConnect()"
289
290   bool connected=false;
291   unsigned int timeOutConnect=mtimeout;
292   while(!connected && timeOutConnect<600){
293     mysql_options(&mData,MYSQL_OPT_CONNECT_TIMEOUT,(const char*)&timeOutConnect);
```

```
294     if(mysql_real_connect(&mData,mdbhost,mdbuser,mdbpw,mdbName,mdbPort,NULL,0))
295     connected=true;
296     if(!connected){
297       timeOutConnect*=2;
298       StString wm;
299       wm<<" Connection Failed with MySQL on "<<mdbhost<<":"<<mdbPort<<stendl;
300       wm<<" Returned error =";
301       wm<<mysql_error(&mData)<<".  Will re-try with timeout set at \n==> ";
302       wm<<timeOutConnect<<" seconds <==";
303       StDbManager::Instance()->printInfo((wm.str()).c_str(),dbMConnect,__LINE__,__CLASS__,__METHOD__);
304     }
305   }
306
307   if(connected){
308     if(mdbServerVersion) delete [] mdbServerVersion;
309  //MPD added 4/28/04 check for valid (not Null) mData.server_version
310     if(!mData.server_version){
311             StString smm;
312             smm<<" No Server version - most likely incompatable libraries \n CONTACT ADMIN";
313             StDbManager::Instance()->printInfo((smm.str()).c_str(),dbMConnect,__LINE__,__CLASS__,__MET
314             assert(mData.server_version);
315     }
316   //MPD end addition
317
318     mdbServerVersion=new char[strlen(mData.server_version)+1];
319     strcpy(mdbServerVersion,mData.server_version);
320   }
321
322   return connected;
323
324 #undef __METHOD__
325 }
```

4.14.2.31 virtual void MysqlDb::Release () [inline, virtual]

Definition at line 236 of file MysqlDb.h.

```
236 {mRes->Release();};
```

4.14.2.32 bool MysqlDb::setDefaultDb (const char * *dbName*) [virtual]

Definition at line 867 of file MysqlDb.cc.

```
867                                         {
868
869   if(!dbName || strlen(dbName)==0)return false;
870 if(mdbName) delete [] mdbName;
871 mdbName=new char[strlen(dbName)+1];
872 strcpy(mdbName,dbName);
873 if(strcmp(dbName," ")==0)return true;
874
875 bool tOk=false;
876 unsigned int mysqlError;
877
878  if(mysql_select_db(&mData,dbName)){
879
880     mysqlError = mysql_errno(&mData);
881     if(mysqlError==CR_SERVER_GONE_ERROR || mysqlError==CR_SERVER_LOST){
882        reConnect();
883        if(mysql_select_db(&mData,dbName)){
884          cerr<< "Error selecting database=" << dbName << endl;
```

```
885          tOk=false;
886        } else {
887          tOk=true;
888        }
889     } else {
890        cerr<< "Error selecting database=" << dbName << endl;
891        tOk=false;
892     }
893  } else {
894    tOk=true;
895  }
896
897 return tOk;
898 }
```

4.14.3 Member Data Documentation

4.14.3.1 char* MysqlDb::cnames[200] [private]

Definition at line 213 of file MysqlDb.h.

4.14.3.2 bool MysqlDb::isBinary[200] [private]

Definition at line 215 of file MysqlDb.h.

4.14.3.3 bool MysqlDb::isBlob[200] [private]

Definition at line 214 of file MysqlDb.h.

4.14.3.4 bool MysqlDb::isSpecialType[200] [private]

Definition at line 216 of file MysqlDb.h.

4.14.3.5 int MysqlDb::jfields [private]

Definition at line 212 of file MysqlDb.h.

4.14.3.6 StDbLogger MysqlDb::mconnectLog

Definition at line 226 of file MysqlDb.h.

4.14.3.7 MYSQL MysqlDb::mData [private]

Definition at line 192 of file MysqlDb.h.

4.14.3.8 char* MysqlDb::mdbhost [private]

Definition at line 201 of file MysqlDb.h.

4.14.3.9 char* MysqlDb::mdbName [private]

Definition at line 202 of file MysqlDb.h.

4.14.3.10 int MysqlDb::mdbPort [private]

Definition at line 205 of file MysqlDb.h.

4.14.3.11 char* MysqlDb::mdbpw [private]

Definition at line 204 of file MysqlDb.h.

4.14.3.12 char* MysqlDb::mdbServerVersion [private]

Definition at line 206 of file MysqlDb.h.

4.14.3.13 char* MysqlDb::mdbuser [private]

Definition at line 203 of file MysqlDb.h.

4.14.3.14 bool MysqlDb::mhasBinaryQuery [private]

Definition at line 199 of file MysqlDb.h.

4.14.3.15 bool MysqlDb::mhasConnected [private]

Definition at line 198 of file MysqlDb.h.

4.14.3.16 bool MysqlDb::mlogTime

Definition at line 223 of file MysqlDb.h.

4.14.3.17 char* MysqlDb::mQuery [private]

Definition at line 193 of file MysqlDb.h.

4.14.3.18 char* MysqlDb::mQueryLast [private]

Definition at line 195 of file MysqlDb.h.

4.14.3.19 unsigned long int MysqlDb::mQueryLen [private]

Definition at line 194 of file MysqlDb.h.

4.14.3.20 StDbLogger MysqlDb::mqueryLog

Definition at line 224 of file MysqlDb.h.

4.14.3.21 bool MysqlDb::mqueryState [private]

Definition at line 197 of file MysqlDb.h.

4.14.3.22 MysqlResult* MysqlDb::mRes [private]

Definition at line 196 of file MysqlDb.h.

4.14.3.23 StDbLogger MysqlDb::msocketLog

Definition at line 225 of file MysqlDb.h.

4.14.3.24 unsigned int MysqlDb::mtimeout [private]

Definition at line 208 of file MysqlDb.h.

4.14.3.25 StDbManagerImpl* MysqlDb::my_manager

Definition at line 271 of file MysqlDb.h.

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

- MysqlDb.h
- MysqlDb.cc

4.15 MysqlResult Class Reference

```
#include <MysqlDb.h>
```

Public Member Functions

- MysqlResult ()
- virtual unsigned NbRows ()
- virtual unsigned NbFields ()
- virtual void Release ()
- MysqlResult & operator++ ()

Private Attributes

- MYSQL_RES * mRes
- MYSQL_ROW mRow
- char mSep

4.15.1 Constructor & Destructor Documentation

4.15.1.1 MysqlResult::MysqlResult () [inline]

Definition at line 159 of file MysqlDb.h.

```
159 {mRes=0;};
```

4.15.2 Member Function Documentation

4.15.2.1 virtual unsigned MysqlResult::NbFields () [inline, virtual]

Definition at line 163 of file MysqlDb.h.

```
163 {return mysql_num_fields(mRes);};
```

4.15.2.2 virtual unsigned MysqlResult::NbRows () [inline, virtual]

Definition at line 162 of file MysqlDb.h.

```
162 {return mysql_num_rows(mRes);};
```

4.15.2.3 MysqlResult& MysqlResult::operator++ () [inline]

Definition at line 165 of file MysqlDb.h.

```
165 {mRow=mysql_fetch_row(mRes);return *this;};
```

4.15.2.4 virtual void MysqlResult::Release () [inline, virtual]

Definition at line 164 of file MysqlDb.h.

```
164 {if(mRes)mysql_free_result(mRes);mRes=0;};
```

4.15.3 Member Data Documentation

4.15.3.1 MYSQL_RES* MysqlResult::mRes [private]

Definition at line 154 of file MysqlDb.h.

4.15.3.2 MYSQL_ROW MysqlResult::mRow [private]

Definition at line 155 of file MysqlDb.h.

4.15.3.3 char MysqlResult::mSep [private]

Definition at line 156 of file MysqlDb.h.

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

- MysqlDb.h

4.16 parseXmlString Class Reference

`#include <parseXmlString.hh>`

Public Member Functions

- parseXmlString ()
- ~parseXmlString ()
- char * getString (char *line, char *key1, char *key2)
- char * getStringAfter (char *line, char *key)
- char * getStringBefore (char *line, char *key)
- int getIndexAfter (char *line, char *key)
- int getIndexBefore (char *line, char *key)
- char * removeBlankEnds (char *line)

4.16.1 Constructor & Destructor Documentation

4.16.1.1 parseXmlString::parseXmlString () [inline]

Definition at line 31 of file parseXmlString.hh.

```
31 {};
```

4.16.1.2 parseXmlString::~parseXmlString () [inline]

Definition at line 32 of file parseXmlString.hh.

```
32 {};
```

4.16.2 Member Function Documentation

4.16.2.1 int parseXmlString::getIndexAfter (char * *line*, char * *key*)

Definition at line 130 of file parseXmlString.cc.

```
130                                                     {
131
132  if(!line || !key)return 0;
133  char* id = strstr(line,key);
134  if(!id)return 0;  //use as bool
135
136  int i = id - line;
137  i += strlen(key)+1;
138  return i;
139 }
```

4.16.2.2 int parseXmlString::getIndexBefore (char * *line*, char * *key*)

Definition at line 144 of file parseXmlString.cc.

```
144                                                     {
145
146  if(!line || !key) return 0;
147  char* id = strstr(line,key);
148  if(!id) return 0; //use as bool
149  int i = id - line;
150  return i--;
151
152 }
```

4.16.2.3 char * parseXmlString::getString (char * *line*, char * *key1*, char * *key2*)

Definition at line 49 of file parseXmlString.cc.

```
49                                                          {
50
51
52 char* p1 = 0;
53 char* p = &line[0];
54
55 int i1 = getIndexAfter(line, key1);
56 if(!i1) return p1;
57 int i2 = getIndexBefore(line, key2);
58 if(!i2) return p1;
59
60
61 int size = i2-i1+1;
62 if(i2<i1){
63   cerr << " Missing Key " << endl;
64   return p1;
65 }
66
67 for (int i=0; i<i1-1; i++)p++;
68
69 p1 = new char[size+1];
70 strncpy(p1,p,size);
71 p1[size]='\0';
72
73 // cout << p1 << endl;
74  char* retVal = removeBlankEnds(p1);
75  delete [] p1;
76  return retVal;
77  // return removeBlankEnds(p1);
78 }
```

4.16.2.4 char * parseXmlString::getStringAfter (char * *line*, char * *key*)

Definition at line 83 of file parseXmlString.cc.

```
83                                                      {
84
85  int i = getIndexAfter(line,key);
86  char* tmp =0;
87
88  if(!i)return tmp;
```

```

char* p1 = &line[i];

int k = strlen(line);
int size = k-i;

tmp = new char[size+1];
strncpy(tmp,p1,size);

tmp[size] = '\0';
char* retVal = removeBlankEnds(tmp);
 delete [] tmp;
 return retVal;
 // return removeBlankEnds(tmp);

}
```

4.16.2.5 char ∗ parseXmlString::getStringBefore (char ∗ *line*, char ∗ *key*)

Definition at line 109 of file parseXmlString.cc.

```
109                                                    {

 int i = getIndexBefore(line,key);

 char * tmp = 0;
 if(!i)return tmp;
 char* p1 = &line[0];
 int size = i;
 tmp = new char[size+1];
 strncpy(tmp,p1,size);
 tmp[size] = '\0';

 char* retVal = removeBlankEnds(tmp);
 delete [] tmp;
 return retVal;
 // return removeBlankEnds(tmp);
}
```

4.16.2.6 char ∗ parseXmlString::removeBlankEnds (char ∗ *line*)

Definition at line 157 of file parseXmlString.cc.

```
157                                          {

 char* p1 = &line[0];
 int k = strlen(line);
 int i,j;

 for(i=0; i<k; i++){
   if(line[i]!=' ')break;
   p1++;
  }

 for(j=k; j>0; j--){
   if(line[j]=='\0')continue;
   if(line[j]!=' ')break;
   }

 int size = j-i+1;
 char* tmp = new char[size+1];
```

```
175  strncpy(tmp,p1,size);
176  tmp[size] = '\0';
177
178  //delete [] line;
179  return tmp;
180 }
```

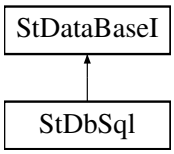
The documentation for this class was generated from the following files:

- parseXmlString.hh
- parseXmlString.cc

4.17 StDataBaseI Class Reference

`#include <StDataBaseI.hh>`

Inheritance diagram for StDataBaseI::



Public Member Functions

- StDataBaseI ()
- StDataBaseI (StDbType type, StDbDomain domain)
- StDataBaseI (const char *typeName, const char *domainName)
- virtual ~StDataBaseI ()
- virtual void setDataBase (StDbType type, StDbDomain domain)
- virtual void setDataBase (const char *typeName, const char *domainName)
- virtual void setDataBase (const char *dbName)
- virtual void setDbType (StDbType type)
- virtual void setDbDomain (StDbDomain domain)
- virtual StDbType getDbType () const
- virtual StDbDomain getDbDomain () const
- virtual void setDbName (const char *dbName)
- virtual void setTypeName (const char *typeName)
- virtual void setDomainName (const char *domainName)
- virtual char * getDbName () const
- virtual char * getTypeName () const
- virtual char * getDomainName () const
- virtual char * printDbName ()
- virtual char * printTypeName ()
- virtual char * printDomainName ()
- virtual bool checkDbName (const char *name)
- virtual bool checkDbType (const char *type)
- virtual bool checkDbDomain (const char *domain)
- virtual StDbStoreType getDbStoreType () const
- virtual void setDbStoreType (StDbStoreType type)
- virtual void use ()=0
- virtual void close ()=0
- virtual int QueryDb (StDbConfigNode *node)=0
- virtual int QueryDb (StDbNode *node)=0
- virtual int QueryDb (StDbTable *table, unsigned int reqTime)=0
- virtual int QueryDb (StDbTable *table, const char *whereClause)=0
- virtual unsigned int * QueryDbTimes (StDbTable *table, const char *whereClause, int opt=0)=0
- virtual int QueryDbFunction (StDbTable *table, const char *whereClause, char *funcName)=0
- virtual int QueryDescriptor (StDbTable *table)=0
- virtual int WriteDb (StDbTable *table, unsigned int storeTime)=0
- virtual int WriteDb (StDbConfigNode *node, int parentID, int &configID)=0

- virtual int storeConfigNode (StDbConfigNode *node)=0
- virtual int storeTableNode (StDbTable *table)=0
- virtual bool insertNodeRelation (int configID, int parent, int child)=0
- virtual bool rollBack (StDbNode *node)=0
- virtual bool rollBack (StDbTable *table)=0
- virtual unsigned int getUnixTime (const char *time)=0
- virtual char * getDateTime (unsigned int time)=0
- virtual int * selectElements (const char *elementName, StDbElementIndex *inval, int &numElements)=0

Protected Attributes

- StDbType mdbType
- StDbDomain mdbDomain
- char * mdbName
- char * mtypeName
- char * mdomainName
- StDbStoreType mdbStore

4.17.1 Constructor & Destructor Documentation

4.17.1.1 StDataBaseI::StDataBaseI ()

Definition at line 44 of file StDataBaseI.cc.

```
44 : mdbType(dbStDb), mdbDomain(dbDomainUnknown), mdbName(0), mtypeName(0), mdomainName(0), mdbStore(dbV00
```

4.17.1.2 StDataBaseI::StDataBaseI (StDbType *type*, StDbDomain *domain*)

Definition at line 46 of file StDataBaseI.cc.

```
46 : mdbName(0), mtypeName(0), mdomainName(0), mdbStore(dbV00) { setDataBase(type,domain); }
```

4.17.1.3 StDataBaseI::StDataBaseI (const char * *typeName*, const char * *domainName*)

Definition at line 48 of file StDataBaseI.cc.

```
48 : mdbName(0), mtypeName(0), mdomainName(0), mdbStore(dbV00) { setDataBase(typeName,domainName); }
```

4.17.1.4 StDataBaseI::~StDataBaseI () [virtual]

Definition at line 51 of file StDataBaseI.cc.

```
51                          {
52   if(mdbName)     delete [] mdbName;
53   if(mtypeName)   delete [] mtypeName;
54   if(mdomainName) delete [] mdomainName;
55 }
```

4.17.2 Member Function Documentation

4.17.2.1 bool StDataBaseI::checkDbDomain (const char * *domain*) [virtual]

Definition at line 170 of file StDataBaseI.cc.

```
170                                                 {
171   return (strcmp(name,mdomainName)==0) ? true : false;
172 }
```

4.17.2.2 bool StDataBaseI::checkDbName (const char * *name*) [virtual]

Definition at line 164 of file StDataBaseI.cc.

```
164                                                 {
165   return (strcmp(name,mdbName)==0) ? true : false;
166 }
```

4.17.2.3 bool StDataBaseI::checkDbType (const char * *type*) [virtual]

Definition at line 167 of file StDataBaseI.cc.

```
167                                                 {
168   return (strcmp(name,mtypeName)==0) ? true : false;
169 }
```

4.17.2.4 virtual void StDataBaseI::close () [pure virtual]

Implemented in StDbSql.

4.17.2.5 virtual char* StDataBaseI::getDateTime (unsigned int *time*) [pure virtual]

Implemented in StDbSql.

4.17.2.6 StDbDomain StDataBaseI::getDbDomain () const [inline, virtual]

Definition at line 141 of file StDataBaseI.hh.

```
141 { return mdbDomain; };
```

4.17.2.7 char * StDataBaseI::getDbName () const [virtual]

Definition at line 125 of file StDataBaseI.cc.

```
125                                 {
126
127  char* retVal=0;
128  if(mdbName) {
```

```
129     retVal = new char[strlen(mdbName)+1];
130     strcpy(retVal,mdbName);
131  }
132 return retVal;
133 }
```

4.17.2.8 StDbStoreType StDataBaseI::getDbStoreType () const [inline, virtual]

Definition at line 142 of file StDataBaseI.hh.

```
142 {return mdbStore;};
```

4.17.2.9 StDbType StDataBaseI::getDbType () const [inline, virtual]

Definition at line 140 of file StDataBaseI.hh.

```
140 { return mdbType; };
```

4.17.2.10 char * StDataBaseI::getDomainName () const [virtual]

Definition at line 149 of file StDataBaseI.cc.

```
149                                 {
150
151  char* retVal=0;
152  if(mdomainName) {
153     retVal = new char[strlen(mdomainName)+1];
154     strcpy(retVal,mdomainName);
155  }
156 return retVal;
157 }
```

4.17.2.11 char * StDataBaseI::getTypeName () const [virtual]

Definition at line 137 of file StDataBaseI.cc.

```
137                                 {
138
139  char* retVal=0;
140  if(mtypeName) {
141     retVal = new char[strlen(mtypeName)+1];
142     strcpy(retVal,mtypeName);
143  }
144 return retVal;
145 }
```

4.17.2.12 virtual unsigned int StDataBaseI::getUnixTime (const char * *time*) [pure virtual]

Implemented in StDbSql.

4.17.2.13 virtual bool StDataBaseI::insertNodeRelation (int *configID*, int *parent*, int *child*) [pure virtual]

Implemented in StDbSql.

4.17.2.14 char * StDataBaseI::printDbName () [virtual]

Definition at line 161 of file StDataBaseI.cc.

```
161 { return mdbName; }
```

4.17.2.15 char * StDataBaseI::printDomainName () [virtual]

Definition at line 163 of file StDataBaseI.cc.

```
163 { return mdomainName; }
```

4.17.2.16 char * StDataBaseI::printTypeName () [virtual]

Definition at line 162 of file StDataBaseI.cc.

```
162 { return mtypeName; }
```

4.17.2.17 virtual int StDataBaseI::QueryDb (StDbTable * *table*, const char * *whereClause*) [pure virtual]

Implemented in StDbSql.

4.17.2.18 virtual int StDataBaseI::QueryDb (StDbTable * *table*, unsigned int *reqTime*) [pure virtual]

Implemented in StDbSql.

4.17.2.19 virtual int StDataBaseI::QueryDb (StDbNode * *node*) [pure virtual]

Implemented in StDbSql.

4.17.2.20 virtual int StDataBaseI::QueryDb (StDbConfigNode * *node*) [pure virtual]

Implemented in StDbSql.

4.17.2.21 virtual int StDataBaseI::QueryDbFunction (StDbTable * *table*, const char * *whereClause*, char * *funcName*) [pure virtual]

Implemented in StDbSql.

4.17.2.22 virtual unsigned int* StDataBaseI::QueryDbTimes (StDbTable * *table*, const char * *whereClause*, int *opt* = 0) [pure virtual]

Implemented in StDbSql.

4.17.2.23 virtual int StDataBaseI::QueryDescriptor (StDbTable * *table*) [pure virtual]

Implemented in StDbSql.

4.17.2.24 virtual bool StDataBaseI::rollBack (StDbTable * *table*) [pure virtual]

Implemented in StDbSql.

4.17.2.25 virtual bool StDataBaseI::rollBack (StDbNode * *node*) [pure virtual]

Implemented in StDbSql.

4.17.2.26 virtual int* StDataBaseI::selectElements (const char * *elementName*, StDbElementIndex * *inval*, int & *numElements*) [pure virtual]

Implemented in StDbSql.

4.17.2.27 void StDataBaseI::setDataBase (const char * *dbName*) [virtual]

Definition at line 83 of file StDataBaseI.cc.

```
83                                          {
84    setDbName(dbName);
85    StDbManager::Instance()->getDataBaseInfo(dbName,mtypeName,mdomainName);
86    setDbType(StDbManager::Instance()->getDbType(mtypeName));
87    setDbDomain(StDbManager::Instance()->getDbDomain(mdomainName));
88 }
```

4.17.2.28 void StDataBaseI::setDataBase (const char * *typeName*, const char * *domainName*) [virtual]

Definition at line 72 of file StDataBaseI.cc.

```
72                                                                  {
73    setTypeName(typeName);
74    setDomainName(domainName);
75    char* name = StDbManager::Instance()->getDbName(mtypeName,mdomainName);
76    setDbName(name); if(name) delete [] name;
77    setDbType(StDbManager::Instance()->getDbType(typeName));
78    setDbDomain(StDbManager::Instance()->getDbDomain(domainName));
79 }
```

4.17.2.29 void StDataBaseI::setDataBase (StDbType *type*, StDbDomain *domain*) [virtual]

Definition at line 59 of file StDataBaseI.cc.

```
59                                                      {
60    setDbType(type);
61    setDbDomain(domain);
62    char* name= StDbManager::Instance()->getDbTypeName(type);
63    setTypeName(name); if(name) delete [] name;
64    name= StDbManager::Instance()->getDbDomainName(domain);
65    setDomainName(name); if(name) delete [] name;
66    name=StDbManager::Instance()->getDbName(mtypeName,mdomainName);
67    setDbName(name); if(name) delete [] name;
68 }
```

4.17.2.30 void StDataBaseI::setDbDomain (StDbDomain *domain*) [inline, virtual]

Definition at line 139 of file StDataBaseI.hh.

```
139 {mdbDomain = domain; }
```

4.17.2.31 void StDataBaseI::setDbName (const char * *dbName*) [virtual]

Definition at line 92 of file StDataBaseI.cc.

```
92                                          {
93
94   if(dbName){
95     if(mdbName) delete [] mdbName;
96     mdbName = new char[strlen(dbName)+1];
97     strcpy(mdbName,dbName);
98   }
99 }
```

4.17.2.32 void StDataBaseI::setDbStoreType (StDbStoreType *type*) [inline, virtual]

Definition at line 143 of file StDataBaseI.hh.

```
143 {mdbStore=type; };
```

4.17.2.33 void StDataBaseI::setDbType (StDbType *type*) [inline, virtual]

Definition at line 138 of file StDataBaseI.hh.

```
138 {mdbType = type; };
```

4.17.2.34 void StDataBaseI::setDomainName (const char * *domainName*) [virtual]

Definition at line 114 of file StDataBaseI.cc.

```
114                                                 {
115
116   if(domainName){
117     if(mdomainName) delete [] mdomainName;
118     mdomainName = new char[strlen(domainName)+1];
119     strcpy(mdomainName,domainName);
120   }
121 }
```

4.17.2.35 void StDataBaseI::setTypeName (const char * *typeName*) [virtual]

Definition at line 103 of file StDataBaseI.cc.

```
103                                             {
104
105   if(typeName){
106     if(mtypeName) delete [] mtypeName;
107     mtypeName = new char[strlen(typeName)+1];
108     strcpy(mtypeName,typeName);
109   }
110 }
```

4.17.2.36 virtual int StDataBaseI::storeConfigNode (StDbConfigNode * *node*) [pure virtual]

Implemented in StDbSql.

4.17.2.37 virtual int StDataBaseI::storeTableNode (StDbTable * *table*) [pure virtual]

Implemented in StDbSql.

4.17.2.38 virtual void StDataBaseI::use () [pure virtual]

Implemented in StDbSql.

4.17.2.39 virtual int StDataBaseI::WriteDb (StDbConfigNode * *node*, int *parentID*, int & *configID*) [pure virtual]

Implemented in StDbSql.

4.17.2.40 virtual int StDataBaseI::WriteDb (StDbTable * *table*, unsigned int *storeTime*) [pure virtual]

Implemented in StDbSql.

4.17.3 Member Data Documentation

4.17.3.1 StDbDomain StDataBaseI::mdbDomain [protected]

Definition at line 59 of file StDataBaseI.hh.

4.17.3.2 char* StDataBaseI::mdbName [protected]

Definition at line 67 of file StDataBaseI.hh.

4.17.3.3 StDbStoreType StDataBaseI::mdbStore [protected]

Definition at line 71 of file StDataBaseI.hh.

4.17.3.4 StDbType StDataBaseI::mdbType [protected]

Definition at line 58 of file StDataBaseI.hh.

4.17.3.5 char* StDataBaseI::mdomainName [protected]

Definition at line 69 of file StDataBaseI.hh.

4.17.3.6 char* StDataBaseI::mtypeName [protected]

Definition at line 68 of file StDataBaseI.hh.

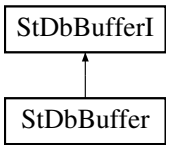
The documentation for this class was generated from the following files:

- StDataBaseI.hh
- StDataBaseI.cc

4.18 StDbBuffer Class Reference

`#include <StDbBuffer.h>`

Inheritance diagram for StDbBuffer::



Public Member Functions

- StDbBuffer ()
- StDbBuffer (unsigned int anum_fields, column *acol)
- void Init (unsigned int anum_fields, column *acol)
- ~StDbBuffer ()
- virtual void SetClientMode ()
- virtual void SetStorageMode ()
- virtual bool IsClientMode ()
- virtual bool IsStorageMode ()
- void Raz ()
- void Print ()
- virtual char ** WhatsIn ()
- virtual bool ReadScalar (char &c, const char *aName)
- virtual bool ReadScalar (unsigned char &c, const char *)
- virtual bool ReadScalar (short &h, const char *)
- virtual bool ReadScalar (unsigned short &h, const char *)
- virtual bool ReadScalar (int &i, const char *)
- virtual bool ReadScalar (unsigned int &i, const char *)
- virtual bool ReadScalar (long &l, const char *)
- virtual bool ReadScalar (unsigned long &l, const char *)
- virtual bool ReadScalar (long long &l, const char *)
- virtual bool ReadScalar (float &f, const char *)
- virtual bool ReadScalar (double &d, const char *)
- virtual bool ReadScalar (char *&c, const char *)
- virtual bool WriteScalar (const char c, const char *)
- virtual bool WriteScalar (const unsigned char c, const char *)
- virtual bool WriteScalar (const short h, const char *)
- virtual bool WriteScalar (const unsigned short h, const char *)
- virtual bool WriteScalar (const int i, const char *)
- virtual bool WriteScalar (const unsigned int i, const char *)
- virtual bool WriteScalar (const long l, const char *)
- virtual bool WriteScalar (const unsigned long l, const char *)
- virtual bool WriteScalar (const long long l, const char *)
- virtual bool WriteScalar (const float f, const char *)
- virtual bool WriteScalar (const double d, const char *)
- virtual bool WriteScalar (const char *c, const char *)
- virtual bool ReadArray (char *&c, int &len, const char *)

- virtual bool ReadArray (unsigned char *&c, int &len, const char *)
- virtual bool ReadArray (short *&c, int &len, const char *)
- virtual bool ReadArray (unsigned short *&c, int &len, const char *)
- virtual bool ReadArray (int *&c, int &len, const char *)
- virtual bool ReadArray (unsigned int *&c, int &len, const char *)
- virtual bool ReadArray (long *&c, int &len, const char *)
- virtual bool ReadArray (unsigned long *&c, int &len, const char *)
- virtual bool ReadArray (long long *&c, int &len, const char *)
- virtual bool ReadArray (float *&c, int &len, const char *)
- virtual bool ReadArray (double *&c, int &len, const char *)
- virtual bool ReadArray (char **&c, int &len, const char *)
- virtual bool WriteArray (char *c, int len, const char *)
- virtual bool WriteArray (unsigned char *c, int len, const char *)
- virtual bool WriteArray (short *c, int len, const char *)
- virtual bool WriteArray (unsigned short *c, int len, const char *)
- virtual bool WriteArray (int *c, int len, const char *)
- virtual bool WriteArray (unsigned int *c, int len, const char *)
- virtual bool WriteArray (long *c, int len, const char *)
- virtual bool WriteArray (unsigned long *c, int len, const char *)
- virtual bool WriteArray (long long *c, int len, const char *)
- virtual bool WriteArray (float *c, int len, const char *)
- virtual bool WriteArray (double *c, int len, const char *)
- virtual bool WriteArray (char **c, int len, const char *)

Protected Member Functions

- bool Find_Col (const char *aName)
- void AddField (const char *aName, const myctype aTpe, const void *aVal, const int aLen)
- void ChangeField (const myctype aTpe, const void *aVal, const int aLen)
- bool WriteMem (char *s, void *aVal, myctype type)
- bool WriteMem (unsigned char *s, void *aVal, myctype type)
- bool WriteMem (short *s, void *aVal, myctype type)
- bool WriteMem (unsigned short *s, void *aVal, myctype type)
- bool WriteMem (int *s, void *aVal, myctype type)
- bool WriteMem (unsigned int *s, void *aVal, myctype type)
- bool WriteMem (long *s, void *aVal, myctype type)
- bool WriteMem (unsigned long *s, void *aVal, myctype type)
- bool WriteMem (long long *s, void *aVal, myctype type)
- bool WriteMem (float *s, void *aVal, myctype type)
- bool WriteMem (double *s, void *aVal, myctype type)
- bool WriteMem (char **s, void *aVal, myctype type)
- void StrConv (char *aVal, char &s)
- void StrConv (char *aVal, unsigned char &s)
- void StrConv (char *aVal, short &s)
- void StrConv (char *aVal, unsigned short &s)
- void StrConv (char *aVal, int &s)
- void StrConv (char *aVal, unsigned int &s)
- void StrConv (char *aVal, long &s)
- void StrConv (char *aVal, unsigned long &s)
- void StrConv (char *aVal, long long &s)

- void StrConv (char *aVal, float &s)
- void StrConv (char *aVal, double &s)
- void StrConv (char *aVal, char *&s)
- void MemSwapCpy (char *where, char *from, int len, int swaplen, BuffMode mode)

Private Member Functions

- void zeroColumn (int istart, int iend)

Private Attributes

- int mMax
- int mCur
- int mLast
- column * mCol
- BuffMode mMode

4.18.1 Constructor & Destructor Documentation

4.18.1.1 StDbBuffer::StDbBuffer () [inline]

Definition at line 87 of file StDbBuffer.h.

```
87              {
88     mCur=0;
89     mLast=-1;
90     mMax=100;
91     mCol= new column[mMax+1];
92     zeroColumn(0,mMax);
93     mMode=Storage;
94   };
```

4.18.1.2 StDbBuffer::StDbBuffer (unsigned int *anum_fields*, column * *acol*) [inline]

Definition at line 96 of file StDbBuffer.h.

```
96                                                  {
97     Init( anum_fields,acol);}
```

4.18.1.3 StDbBuffer::~StDbBuffer () [inline]

Definition at line 104 of file StDbBuffer.h.

```
104 {Raz();delete [] mCol;  };
```

4.18.2 Member Function Documentation

4.18.2.1 void StDbBuffer::AddField (const char ∗ *aName*, const myctype *aTpe*, const void ∗ *aVal*, const int *aLen*) [protected]

Definition at line 208 of file StDbBuffer.cc.

```
208                                                                                                          {
209   /*  when the field aName doesn't exist int the buffer, this function is call
210       to initialise a new field. if needed, it increase the array mCol.
211       it store the name, and then call ChangeField to store the value */
212
213   if (mLast+1>mMax) {
214     column *tCol=new column[mMax*2+1];
215     memcpy(tCol,mCol,(mMax+1)*sizeof(column));
216     if (mCol) delete [] mCol;
217     mCol=tCol;
218     zeroColumn(mMax+1,2*mMax);
219     mMax=mMax*2;
220   }
221   mLast++;
222   mCur=mLast;
223   if(mCol[mCur].name) delete [] mCol[mCur].name;
224   mCol[mCur].name=new char[strlen(aName)+1];
225   strcpy(mCol[mCur].name,aName);
226   mCol[mCur].val=0;
227   ChangeField(aTpe,aVal,aLen);
228 };
```

4.18.2.2 void StDbBuffer::ChangeField (const myctype *aTpe*, const void ∗ *aVal*, const int *aLen*) [protected]

Definition at line 231 of file StDbBuffer.cc.

```
231                                                                                          {
232   if (mCol[mCur].val){
233    if(mCol[mCur].type>=_ascii){
234         char ** tTextVal= (char**)mCol[mCur].val;
235         int j;
236         for(j=0;j<(int)mCol[mCur].length;j++) {
237           if (tTextVal[j]) delete [] tTextVal[j];
238         };
239         delete [] tTextVal ; mCol[mCur].val=0;
240     }
241   }
242   mCol[mCur].type=aTpe;
243   mCol[mCur].length=aLen;
244   if (aTpe<_ascii) {
245     mCol[mCur].val=new char[aLen*mycsize[aTpe]];
246     MemSwapCpy(mCol[mCur].val,(char*)aVal,aLen*mycsize[aTpe],mycswapl[aTpe],Auto);
247   } else {
248     mCol[mCur].val=(char*) new char*[aLen];
249     char** tVal=(char**)aVal;
250     char** tStoreVal=(char**)mCol[mCur].val;
251     int i;
252     for (i=0;i<aLen;i++) {
253       if (tVal[i]==0) {
254         (tStoreVal)[i]=0;
255       } else {
256         tStoreVal[i]=new char[strlen(tVal[i])+1];
257         strcpy(tStoreVal[i],tVal[i]);
258       };
```

```
    };
  };


};
```

4.18.2.3 bool StDbBuffer::Find_Col (const char * *aName*) [protected]

Definition at line 191 of file StDbBuffer.cc.

```
                                          {
  // int tCount=0;

  if (mLast==-1) return false;

  for (int tCount=0;tCount<mLast+1;tCount++){
    mCur++;
    if (mCur>mLast) mCur=0;

    if (!strcmp(mCol[mCur].name,aName)) return true;
  };

  return false;
}
```

4.18.2.4 void StDbBuffer::Init (unsigned int *anum_fields*, column * *acol*) [inline]

Definition at line 99 of file StDbBuffer.h.

```
                                                  {
    mCur=0;
    mLast=anum_fields;
    mCol=acol;};
```

4.18.2.5 virtual bool StDbBuffer::IsClientMode () [inline, virtual]

Implements StDbBufferI.

Definition at line 108 of file StDbBuffer.h.

```
{return (mMode==Client) ? true : false;};
```

4.18.2.6 virtual bool StDbBuffer::IsStorageMode () [inline, virtual]

Implements StDbBufferI.

Definition at line 109 of file StDbBuffer.h.

```
{return (mMode==Storage) ? true : false;};
```

4.18.2.7 void StDbBuffer::MemSwapCpy (char * *where*, char * *from*, int *len*, int *swaplen*, BuffMode *mode*) [protected]

Definition at line 266 of file StDbBuffer.cc.

```
266                                                                                                       {
267   if (mode==Auto) mode=mMode;
268   if (swaplen<=1||mode==Storage) {
269     memcpy(where,from,len);
270   } else {
271     if (len%swaplen!=0) {
272       cerr << "memswapcy: len not in agreement with swapping - binary  truncate " << endl;
273     };
274     int tNbCpy=len/swaplen;
275     int i;
276     for (i=0;i<tNbCpy;i++){
277       where+=swaplen;
278       int j;
279       for (j=0;j<swaplen;j++){
280         where--;
281         *where=*from;
282         from++;
283       }
284       where+=swaplen;
285     };
286   };
287 };
```

4.18.2.8 void StDbBuffer::Print ()

Definition at line 122 of file StDbBuffer.cc.

```
122                       {
123   if (mCol) {
124     int i;
125     for (i=0; i<=mLast;i++) {
126       if (mCol[i].type==_char) {
127         char* tVal;int len;
128         ReadArray(tVal,len,mCol[i].name);
129         cout << mCol[i].name <<"=" << "#BIN (" << len << ") bytes"  <<endl;
130       }else {
131         char** tVal; int len;
132         ReadArray(tVal,len,mCol[i].name);
133         int j;
134         for (j=0;j<len;j++)
135         {cout << mCol[i].name <<"["<< j << "]=" << tVal[j]  <<endl;}
136       };
137     };
138   };
139 };
```

4.18.2.9 void StDbBuffer::Raz ()

Definition at line 151 of file StDbBuffer.cc.

```
151                     {
152
153   int i;
154   for (i=0;i<=mLast;i++) {
```

```
    if (mCol[i].val)  {
      if (mCol[i].type>=_ascii){
              char ** tTextVal= (char**)mCol[i].val;
              int j;
              for(j=0;j<(int)mCol[i].length;j++) {

                  if (tTextVal[j]) delete [] tTextVal[j];
              };
              delete [] tTextVal;
      } else {
          delete [] mCol[i].val ;
      };
    };
    if (mCol[i].name) delete []  mCol[i].name;
    mCol[i].val=0;
    mCol[i].name=0;
    mCol[i].type=_char;
    mCol[i].length=0;
  };
  mCur=0;
  mLast=-1;
}
```

4.18.2.10 virtual bool StDbBuffer::ReadArray (char **& *c*, int & *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.11 virtual bool StDbBuffer::ReadArray (double *& *c*, int & *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.12 virtual bool StDbBuffer::ReadArray (float *& *c*, int & *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.13 virtual bool StDbBuffer::ReadArray (long long *& *c*, int & *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.14 virtual bool StDbBuffer::ReadArray (unsigned long *& *c*, int & *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.15 virtual bool StDbBuffer::ReadArray (long *& *c*, int & *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.16 virtual bool StDbBuffer::ReadArray (unsigned int *& *c*, int & *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.17 virtual bool StDbBuffer::ReadArray (int *& *c*, int & *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.18 virtual bool StDbBuffer::ReadArray (unsigned short *& *c*, int & *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.19 virtual bool StDbBuffer::ReadArray (short *& *c*, int & *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.20 virtual bool StDbBuffer::ReadArray (unsigned char *& *c*, int & *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.21 bool StDbBuffer::ReadArray (char *& *c*, int & *len*, const char *) [virtual]

Implements StDbBufferI.

Definition at line 482 of file StDbBuffer.cc.

```
483 { bool tRetVal=false;
484  if (Find_Col(aName)) {
485    len=mCol[mCur].length*mycsize[mCol[mCur].type];
486    s=new char[len];
487    MemSwapCpy((char*)s,(char*)mCol[mCur].val,len,mycswapl[mCol[mCur].type],Auto);
488    tRetVal=true;
489  } else {
490    s=0;
491    //   cerr << "WARNING:: field " << aName << " doesnt exist in this Buffer" << endl;
492  };
493  return tRetVal;\
494 };
```

4.18.2.22 virtual bool StDbBuffer::ReadScalar (char *& *c*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.23 virtual bool StDbBuffer::ReadScalar (double & *d*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.24 virtual bool StDbBuffer::ReadScalar (float & *f*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.25 virtual bool StDbBuffer::ReadScalar (long long & *l*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.26 virtual bool StDbBuffer::ReadScalar (unsigned long & *l*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.27 virtual bool StDbBuffer::ReadScalar (long & *l*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.28 virtual bool StDbBuffer::ReadScalar (unsigned int & *i*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.29 virtual bool StDbBuffer::ReadScalar (int & *i*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.30 virtual bool StDbBuffer::ReadScalar (unsigned short & *h*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.31 virtual bool StDbBuffer::ReadScalar (short & *h*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.32 virtual bool StDbBuffer::ReadScalar (unsigned char & *c*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.33 virtual bool StDbBuffer::ReadScalar (char & *c*, const char * *aName*) [virtual]

Implements StDbBufferI.

4.18.2.34 virtual void StDbBuffer::SetClientMode () [inline, virtual]

Implements StDbBufferI.

Definition at line 106 of file StDbBuffer.h.

```
106 {mMode=Client;};
```

4.18.2.35 virtual void StDbBuffer::SetStorageMode () [inline, virtual]

Implements StDbBufferI.

Definition at line 107 of file StDbBuffer.h.

```
107 {mMode=Storage;};
```

4.18.2.36 void StDbBuffer::StrConv (char * *aVal*, char *& *s*) [protected]

Definition at line 306 of file StDbBuffer.cc.

```
306 {s=new char[strlen(aVal)+1];strcpy(s,aVal);};
```

4.18.2.37 void StDbBuffer::StrConv (char * *aVal*, double & *s*) [protected]

Definition at line 305 of file StDbBuffer.cc.

```
305 {s=atof(aVal);};
```

4.18.2.38 void StDbBuffer::StrConv (char * *aVal*, float & *s*) [protected]

Definition at line 304 of file StDbBuffer.cc.

```
304 {s=(float) atof(aVal);};
```

4.18.2.39 void StDbBuffer::StrConv (char * *aVal*, long long & *s*) [protected]

Definition at line 300 of file StDbBuffer.cc.

```
300 {s=atoll(aVal);};
```

4.18.2.40 void StDbBuffer::StrConv (char * *aVal*, unsigned long & *s*) [protected]

Definition at line 298 of file StDbBuffer.cc.

```
298 {s=atol(aVal);};
```

4.18.2.41 void StDbBuffer::StrConv (char * *aVal*, long & *s*) [protected]

Definition at line 297 of file StDbBuffer.cc.

```
297 {s=atoi(aVal);};
```

4.18.2.42 void StDbBuffer::StrConv (char * *aVal*, unsigned int & *s*) [protected]

Definition at line 296 of file StDbBuffer.cc.

```
296 {s=atol(aVal);};
```

4.18.2.43 void StDbBuffer::StrConv (char * *aVal*, int & *s*) [protected]

Definition at line 295 of file StDbBuffer.cc.

```
295 {s=atoi(aVal);};
```

4.18.2.44 void StDbBuffer::StrConv (char * *aVal*, unsigned short & *s*) [protected]

Definition at line 294 of file StDbBuffer.cc.

```
294 {s=(unsigned short) atoi(aVal);};
```

4.18.2.45 void StDbBuffer::StrConv (char * *aVal*, short & *s*) [protected]

Definition at line 293 of file StDbBuffer.cc.

```
293 {s=(short)atoi(aVal);};
```

4.18.2.46 void StDbBuffer::StrConv (char * *aVal*, unsigned char & *s*) [protected]

Definition at line 292 of file StDbBuffer.cc.

```
292 {s=(unsigned char)atoi(aVal);};
```

4.18.2.47 void StDbBuffer::StrConv (char * *aVal*, char & *s*) [protected]

Definition at line 291 of file StDbBuffer.cc.

```
291 {s=aVal[0];};
```

4.18.2.48 char ** StDbBuffer::WhatsIn () [virtual]

Definition at line 179 of file StDbBuffer.cc.

```
179                          {
180   char **tIn;
181   tIn=new char*[mLast+2];
182   int i;
183   for (i=0;i<=mLast;i++){
184     tIn[i]=mCol[i].name;
185   };
186   tIn[mLast+1]=0;
187   return tIn;
188 }
```

4.18.2.49 virtual bool StDbBuffer::WriteArray (char ** *c*, int *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.50 virtual bool StDbBuffer::WriteArray (double * *c*, int *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.51 virtual bool StDbBuffer::WriteArray (float * *c*, int *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.52 virtual bool StDbBuffer::WriteArray (long long * *c*, int *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.53 virtual bool StDbBuffer::WriteArray (unsigned long * *c*, int *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.54 virtual bool StDbBuffer::WriteArray (long * *c*, int *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.55 virtual bool StDbBuffer::WriteArray (unsigned int * *c*, int *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.56 virtual bool StDbBuffer::WriteArray (int * *c*, int *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.57 virtual bool StDbBuffer::WriteArray (unsigned short * *c*, int *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.58 virtual bool StDbBuffer::WriteArray (short * *c*, int *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.59 virtual bool StDbBuffer::WriteArray (unsigned char * *c*, int *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.60 virtual bool StDbBuffer::WriteArray (char * *c*, int *len*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.61 bool StDbBuffer::WriteMem (char ** *s*, void * *aVal*, myctype *type*) [protected]

Definition at line 344 of file StDbBuffer.cc.

```
344                                                                 {
345   bool tRetVal=true;
346
347   switch (type) {
348     castcasest(_char,char);
349     castcasest(_uchar,unsigned char);
350     castcasest(_short,short);
351     castcasest(_ushort,unsigned short);
352     castcasest(_int,int);
353     castcasest(_uint,unsigned int);
354     castcasest(_long,long);
355     castcasest(_ulong,unsigned long);
356     castcasest(_longlong,long long);
357     castcasest(_float,float);
358     castcasest(_double,double);
359   case _string: {char** tVal=(char**)aVal;
360                  *s=new char[strlen(*(char**)aVal)+1];
361                  strcpy(*s,*tVal);
362                  }
363                  break;
364   default: cout <<"wrong type" << endl;tRetVal=false;\
365   };\
366   return tRetVal;\
367 }
```

4.18.2.62 bool StDbBuffer::WriteMem (double * *s*, void * *aVal*, myctype *type*) [protected]

4.18.2.63 bool StDbBuffer::WriteMem (float * *s*, void * *aVal*, myctype *type*) [protected]

4.18.2.64 bool StDbBuffer::WriteMem (long long * *s*, void * *aVal*, myctype *type*) [protected]

4.18.2.65 bool StDbBuffer::WriteMem (unsigned long * *s*, void * *aVal*, myctype *type*) [protected]

4.18.2.66 bool StDbBuffer::WriteMem (long * *s*, void * *aVal*, myctype *type*) [protected]

4.18.2.67 bool StDbBuffer::WriteMem (unsigned int * *s*, void * *aVal*, myctype *type*) [protected]

4.18.2.68 bool StDbBuffer::WriteMem (int * *s*, void * *aVal*, myctype *type*) [protected]

4.18.2.69 bool StDbBuffer::WriteMem (unsigned short * *s*, void * *aVal*, myctype *type*) [protected]

4.18.2.70 bool StDbBuffer::WriteMem (short * *s*, void * *aVal*, myctype *type*) [protected]

4.18.2.71 bool StDbBuffer::WriteMem (unsigned char * *s*, void * *aVal*, myctype *type*) [protected]

4.18.2.72 bool StDbBuffer::WriteMem (char * *s*, void * *aVal*, myctype *type*) [protected]

4.18.2.73 bool StDbBuffer::WriteScalar (const char * *c*, const char *) [virtual]

Implements StDbBufferI.

Definition at line 419 of file StDbBuffer.cc.

```
{
  if(!s) return false;
//  cout<< "Createin pointerstring "<<endl;
  //**  char** tVal=new char*[1];
  char* aVal=new char[strlen(s)+1];
  strcpy(aVal,s);
  char** tVal=&aVal;
//  cout<< "Creatin storag string "<<endl;
  //**  tVal[0]=new char[strlen(s)+1];
//  cout<< "Copy storag string "<<endl;
  //**  strcpy(tVal[0],s);
  /* if (Find_Col(aName))
   //  { ChangeField(_string,(void*)&tVal,1);}
  { ChangeField(_string,(void*)tVal,1); delete [] tVal[0]; delete [] tVal;}
 else
   { AddField(aName,_string,(void*)tVal,1); delete [] tVal[0]; delete [] tVal;}
   */
 if (Find_Col(aName))
  { ChangeField(_string,(void*)tVal,1); delete [] aVal;}
 else
   { AddField(aName,_string,(void*)tVal,1); delete [] aVal;};
return true;
}
```

4.18.2.74 virtual bool StDbBuffer::WriteScalar (const double *d*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.75 virtual bool StDbBuffer::WriteScalar (const float *f*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.76 virtual bool StDbBuffer::WriteScalar (const long long *l*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.77 virtual bool StDbBuffer::WriteScalar (const unsigned long *l*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.78 virtual bool StDbBuffer::WriteScalar (const long *l*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.79 virtual bool StDbBuffer::WriteScalar (const unsigned int *i*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.80 virtual bool StDbBuffer::WriteScalar (const int *i*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.81 virtual bool StDbBuffer::WriteScalar (const unsigned short *h*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.82 virtual bool StDbBuffer::WriteScalar (const short *h*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.83 virtual bool StDbBuffer::WriteScalar (const unsigned char *c*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.84 virtual bool StDbBuffer::WriteScalar (const char *c*, const char *) [virtual]

Implements StDbBufferI.

4.18.2.85 void StDbBuffer::zeroColumn (int *istart*, int *iend*) [private]

Definition at line 142 of file StDbBuffer.cc.

```
142                                                 {
143   for(int i=istart;i<iend+1;i++){
144      mCol[i].name = 0;
145      mCol[i].val  = 0;
146   }
147 }
```

4.18.3 Member Data Documentation

4.18.3.1 column* StDbBuffer::mCol [private]

Definition at line 80 of file StDbBuffer.h.

4.18.3.2 int StDbBuffer::mCur [private]

Definition at line 78 of file StDbBuffer.h.

4.18.3.3 int StDbBuffer::mLast [private]

Definition at line 79 of file StDbBuffer.h.

4.18.3.4 int StDbBuffer::mMax [private]

Definition at line 77 of file StDbBuffer.h.

4.18.3.5 BuffMode StDbBuffer::mMode [private]

Definition at line 81 of file StDbBuffer.h.

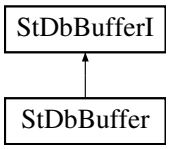
The documentation for this class was generated from the following files:

- StDbBuffer.h
- StDbBuffer.cc

4.19 StDbBufferI Class Reference

`#include <StDbBufferI.h>`

Inheritance diagram for StDbBufferI::



Public Member Functions

- virtual ~StDbBufferI ()
- virtual void SetClientMode ()=0
- virtual void SetStorageMode ()=0
- virtual bool IsClientMode ()=0
- virtual bool IsStorageMode ()=0
- virtual bool ReadScalar (char &c, const char *aName)=0
- virtual bool ReadScalar (unsigned char &c, const char *)=0
- virtual bool ReadScalar (short &h, const char *)=0
- virtual bool ReadScalar (unsigned short &h, const char *)=0
- virtual bool ReadScalar (int &i, const char *)=0
- virtual bool ReadScalar (unsigned int &i, const char *)=0
- virtual bool ReadScalar (long &l, const char *)=0
- virtual bool ReadScalar (unsigned long &l, const char *)=0
- virtual bool ReadScalar (long long &l, const char *)=0
- virtual bool ReadScalar (float &f, const char *)=0
- virtual bool ReadScalar (double &d, const char *)=0
- virtual bool ReadScalar (char *&c, const char *)=0
- virtual bool WriteScalar (const char c, const char *)=0
- virtual bool WriteScalar (const unsigned char c, const char *)=0
- virtual bool WriteScalar (const short h, const char *)=0
- virtual bool WriteScalar (const unsigned short h, const char *)=0
- virtual bool WriteScalar (const int i, const char *)=0
- virtual bool WriteScalar (const unsigned int i, const char *)=0
- virtual bool WriteScalar (const long l, const char *)=0
- virtual bool WriteScalar (const unsigned long l, const char *)=0
- virtual bool WriteScalar (const long long l, const char *)=0
- virtual bool WriteScalar (const float f, const char *)=0
- virtual bool WriteScalar (const double d, const char *)=0
- virtual bool WriteScalar (const char *c, const char *)=0
- virtual bool ReadArray (char *&c, int &len, const char *)=0
- virtual bool ReadArray (unsigned char *&c, int &len, const char *)=0
- virtual bool ReadArray (short *&c, int &len, const char *)=0
- virtual bool ReadArray (unsigned short *&c, int &len, const char *)=0
- virtual bool ReadArray (int *&c, int &len, const char *)=0
- virtual bool ReadArray (unsigned int *&c, int &len, const char *)=0
- virtual bool ReadArray (long *&c, int &len, const char *)=0

- virtual bool ReadArray (unsigned long *&c, int &len, const char *)=0
- virtual bool ReadArray (long long *&c, int &len, const char *)=0
- virtual bool ReadArray (float *&c, int &len, const char *)=0
- virtual bool ReadArray (double *&c, int &len, const char *)=0
- virtual bool ReadArray (char **&c, int &len, const char *)=0
- virtual bool WriteArray (char *c, int len, const char *)=0
- virtual bool WriteArray (unsigned char *c, int len, const char *)=0
- virtual bool WriteArray (short *c, int len, const char *)=0
- virtual bool WriteArray (unsigned short *c, int len, const char *)=0
- virtual bool WriteArray (int *c, int len, const char *)=0
- virtual bool WriteArray (unsigned int *c, int len, const char *)=0
- virtual bool WriteArray (long *c, int len, const char *)=0
- virtual bool WriteArray (unsigned long *c, int len, const char *)=0
- virtual bool WriteArray (long long *c, int len, const char *)=0
- virtual bool WriteArray (float *c, int len, const char *)=0
- virtual bool WriteArray (double *c, int len, const char *)=0
- virtual bool WriteArray (char **c, int len, const char *)=0

4.19.1 Constructor & Destructor Documentation

4.19.1.1 virtual StDbBufferI::~StDbBufferI () [inline, virtual]

Definition at line 62 of file StDbBufferI.h.

```
62 {};
```

4.19.2 Member Function Documentation

4.19.2.1 virtual bool StDbBufferI::IsClientMode () [pure virtual]

Implemented in StDbBuffer.

4.19.2.2 virtual bool StDbBufferI::IsStorageMode () [pure virtual]

Implemented in StDbBuffer.

4.19.2.3 virtual bool StDbBufferI::ReadArray (char **& *c*, int & *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.4 virtual bool StDbBufferI::ReadArray (double *& *c*, int & *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.5 virtual bool StDbBufferI::ReadArray (float *& *c*, int & *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.6 virtual bool StDbBufferI::ReadArray (long long *& *c*, int & *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.7 virtual bool StDbBufferI::ReadArray (unsigned long *& *c*, int & *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.8 virtual bool StDbBufferI::ReadArray (long *& *c*, int & *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.9 virtual bool StDbBufferI::ReadArray (unsigned int *& *c*, int & *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.10 virtual bool StDbBufferI::ReadArray (int *& *c*, int & *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.11 virtual bool StDbBufferI::ReadArray (unsigned short *& *c*, int & *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.12 virtual bool StDbBufferI::ReadArray (short *& *c*, int & *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.13 virtual bool StDbBufferI::ReadArray (unsigned char *& *c*, int & *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.14 virtual bool StDbBufferI::ReadArray (char *& *c*, int & *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.15 virtual bool StDbBufferI::ReadScalar (char *& *c*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.16 virtual bool StDbBufferI::ReadScalar (double & *d*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.17 virtual bool StDbBufferI::ReadScalar (float & *f*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.18 virtual bool StDbBufferI::ReadScalar (long long & *l*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.19 virtual bool StDbBufferI::ReadScalar (unsigned long & *l*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.20 virtual bool StDbBufferI::ReadScalar (long & *l*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.21 virtual bool StDbBufferI::ReadScalar (unsigned int & *i*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.22 virtual bool StDbBufferI::ReadScalar (int & *i*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.23 virtual bool StDbBufferI::ReadScalar (unsigned short & *h*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.24 virtual bool StDbBufferI::ReadScalar (short & *h*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.25 virtual bool StDbBufferI::ReadScalar (unsigned char & *c*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.26 virtual bool StDbBufferI::ReadScalar (char & *c*, const char * *aName*) [pure virtual]

Implemented in StDbBuffer.

4.19.2.27 virtual void StDbBufferI::SetClientMode () [pure virtual]

Implemented in StDbBuffer.

4.19.2.28 virtual void StDbBufferI::SetStorageMode () [pure virtual]

Implemented in StDbBuffer.

4.19.2.29 virtual bool StDbBufferI::WriteArray (char ** *c*, int *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.30 virtual bool StDbBufferI::WriteArray (double * *c*, int *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.31 virtual bool StDbBufferI::WriteArray (float * *c*, int *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.32 virtual bool StDbBufferI::WriteArray (long long * *c*, int *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.33 virtual bool StDbBufferI::WriteArray (unsigned long * *c*, int *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.34 virtual bool StDbBufferI::WriteArray (long * *c*, int *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.35 virtual bool StDbBufferI::WriteArray (unsigned int * *c*, int *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.36 virtual bool StDbBufferI::WriteArray (int * *c*, int *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.37 virtual bool StDbBufferI::WriteArray (unsigned short * *c*, int *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.38 virtual bool StDbBufferI::WriteArray (short * *c*, int *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.39 virtual bool StDbBufferI::WriteArray (unsigned char * *c*, int *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.40 virtual bool StDbBufferI::WriteArray (char * *c*, int *len*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.41 virtual bool StDbBufferI::WriteScalar (const char * *c*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.42 virtual bool StDbBufferI::WriteScalar (const double *d*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.43 virtual bool StDbBufferI::WriteScalar (const float *f*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.44 virtual bool StDbBufferI::WriteScalar (const long long *l*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.45 virtual bool StDbBufferI::WriteScalar (const unsigned long *l*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.46 virtual bool StDbBufferI::WriteScalar (const long *l*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.47 virtual bool StDbBufferI::WriteScalar (const unsigned int *i*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.48 virtual bool StDbBufferI::WriteScalar (const int *i*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.49 virtual bool StDbBufferI::WriteScalar (const unsigned short *h*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.50 virtual bool StDbBufferI::WriteScalar (const short *h*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.51 virtual bool StDbBufferI::WriteScalar (const unsigned char *c*, const char *) [pure virtual]

Implemented in StDbBuffer.

4.19.2.52 virtual bool StDbBufferI::WriteScalar (const char *c*, const char *) [pure virtual]

Implemented in StDbBuffer.

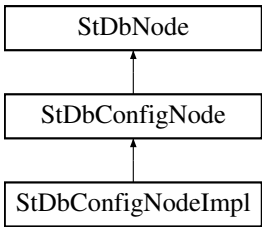
The documentation for this class was generated from the following file:

- StDbBufferI.h

4.20 StDbConfigNode Class Reference

`#include <StDbConfigNode.hh>`

Inheritance diagram for StDbConfigNode::



Public Member Functions

- StDbConfigNode (StDbConfigNode *parent, const char *nodeName, const char *configName)
- StDbConfigNode (StDbConfigNode *parent, StDbNode &node)
- StDbConfigNode (StDbType type, StDbDomain domain, const char *nodeName, const char *configName="none")
- virtual ~StDbConfigNode ()
- void setNextNode (StDbConfigNode *node)
- void setParentNode (StDbConfigNode *node)
- void setFirstChildNode (StDbConfigNode *node)
- virtual void setChildNode (StDbConfigNode *node)
- void appendNode (StDbConfigNode *node)
- StDbConfigNode * getNextNode ()
- StDbConfigNode * getParentNode ()
- StDbConfigNode * getFirstChildNode ()
- void deleteTree ()
- virtual void addChildren (dbEnvList *elist)=0
- void deleteChildren ()
- bool hasChildren ()
- bool hasData ()
- int getBranchID ()
- void setBranchID (int branchID)
- void setIsDbNode (bool isDbNode)
- bool isDbNode ()
- void printTree (int depth)
- virtual void resetConfig (const char *config, int opt=0)=0
- virtual int buildTree (int opt=0)=0
- virtual StDbElementIndex * getElementIndex ()=0
- virtual void setElementIndexInfo (const char *indexName, int indexID)=0
- virtual void getElementIndexInfo (char *&indexname, int &indexID)=0
- virtual int getNumIndeces () const =0
- virtual StDbTable * addDbTable (const char *tableName, const char *version="default")=0
- virtual StDbTable * addTable (const char *tableName, const char *version="default")=0
- virtual StDbTable * addTable (StDbNode *node)=0
- virtual StDbTable * findTable (const char *name, const char *sPath="/")=0
- virtual StDbTable * findLocalTable (const char *name)=0

- virtual void removeTable (StDbTable ∗table)=0
- virtual StDbTableIter ∗ getStDbTableIter ()=0
- virtual bool compareTables (StDbTable ∗tab1, StDbTable ∗tab2)=0
- virtual void printTables (int depth)=0
- virtual void printNumberStats ()=0
- virtual void getNumberStats (unsigned int &nNodes, unsigned int &ntables, unsigned int &numBytes)=0
- virtual void setTablesFlavor (const char ∗flavor)=0
- virtual void setTablesProdTime (unsigned int ptime)=0
- void setFlavor (const char ∗flavor)
- void setProdTime (unsigned int ptime)
- StDbConfigNode ∗ findConfigNode (StDbType t, StDbDomain d, const char ∗sPath)
- StDbConfigNode ∗ findConfigNode (StDbType t, StDbDomain d)
- StDbConfigNode ∗ findConfigNode (const char ∗sPath)
- StDbConfigNode ∗ findChildConfigNode (const char ∗nodeName)

Protected Member Functions

- void zeroNodes ()

Protected Attributes

- StDbConfigNode ∗ mfirstChildNode
- StDbConfigNode ∗ mnextNode
- StDbConfigNode ∗ mparentNode
- bool mhasData
- int mbranchID
- bool misDbNode

4.20.1 Constructor & Destructor Documentation

4.20.1.1 StDbConfigNode::StDbConfigNode (StDbConfigNode ∗ *parent*, const char ∗ *nodeName*, const char ∗ *configName*)

Definition at line 133 of file StDbConfigNode.cc.

```
133                                                                              : St
134
135   zeroNodes();
136   setParentNode(parent);
137   setDbName(parent->printDbName());
138   setDbType(parent->getDbType());
139   setDbDomain(parent->getDbDomain());
140 }
```

4.20.1.2 StDbConfigNode::StDbConfigNode (StDbConfigNode ∗ *parent*, StDbNode & *node*)

Definition at line 143 of file StDbConfigNode.cc.

```
143                                                       : StDbNode(node) {
144   zeroNodes();
145   setParentNode(parent);
146 }
```

4.20.1.3 StDbConfigNode::StDbConfigNode (StDbType *type*, StDbDomain *domain*, const char * *nodeName*, const char * *configName* = "none")

Definition at line 149 of file StDbConfigNode.cc.

```
149
150
151   zeroNodes();
152   mdbType = type;
153   mdbDomain = domain;
154 }
```

4.20.1.4 StDbConfigNode::~StDbConfigNode () [virtual]

Definition at line 157 of file StDbConfigNode.cc.

```
157 { deleteChildren();};
```

4.20.2 Member Function Documentation

4.20.2.1 virtual void StDbConfigNode::addChildren (dbEnvList * *elist*) [pure virtual]

Implemented in StDbConfigNodeImpl.

4.20.2.2 virtual StDbTable* StDbConfigNode::addDbTable (const char * *tableName*, const char * *version* = "default") [pure virtual]

Implemented in StDbConfigNodeImpl.

4.20.2.3 virtual StDbTable* StDbConfigNode::addTable (StDbNode * *node*) [pure virtual]

Implemented in StDbConfigNodeImpl.

4.20.2.4 virtual StDbTable* StDbConfigNode::addTable (const char * *tableName*, const char * *version* = "default") [pure virtual]

Implemented in StDbConfigNodeImpl.

4.20.2.5 void StDbConfigNode::appendNode (StDbConfigNode * *node*)

Definition at line 314 of file StDbConfigNode.cc.

```
314                                                    {
315   if(mnextNode){
316      mnextNode->appendNode(node);
317   } else {
318      mnextNode = node;
319   }
320 }
```

4.20.2.6 virtual int StDbConfigNode::buildTree (int *opt* = 0) [pure virtual]

Implemented in StDbConfigNodeImpl.

4.20.2.7 virtual bool StDbConfigNode::compareTables (StDbTable * *tab1*, StDbTable * *tab2*) [pure virtual]

Implemented in StDbConfigNodeImpl.

4.20.2.8 void StDbConfigNode::deleteChildren ()

Definition at line 203 of file StDbConfigNode.cc.

```
203                                {

204

205   StDbConfigNode* nextChild = mfirstChildNode;
206   while(nextChild){
207     StDbConfigNode* node=nextChild->getNextNode();
208     delete nextChild;
209     nextChild=node;
210   }
211   mfirstChildNode=0;
212 }
```

4.20.2.9 void StDbConfigNode::deleteTree ()

Definition at line 192 of file StDbConfigNode.cc.

```
192                            {

193

194  if(mfirstChildNode)delete mfirstChildNode;
195  if(mnextNode){
196    mnextNode->deleteTree();
197    delete mnextNode;
198  }
199 }
```

4.20.2.10 StDbConfigNode * StDbConfigNode::findChildConfigNode (const char * *nodeName*)

Definition at line 295 of file StDbConfigNode.cc.

```
295                                                    {

296

297   StDbConfigNode* retVal=0;
298   if(!name) return retVal;
299   retVal=getFirstChildNode();
300   while(retVal){
301     if(retVal->checkName(name)) break;
302     retVal=retVal->getNextNode();
303   }
304   return retVal;
305 }
```

4.20.2.11 StDbConfigNode * StDbConfigNode::findConfigNode (const char * *sPath*)

Definition at line 252 of file StDbConfigNode.cc.

```
252                                                 {
253
254   // tries to find a node below (in the child sense)
255   // where nodeName/nodeName/nodeName/... = subPath
256   //
257   // returns null pointer if not found
258
259   StDbConfigNode* node=0;
260   if(!subPath) return node;
261   char* path = mstrDup(subPath);
262
263 if(path[0]=='/')path++;
264 char* id=strstr(path,"/");
265 if(id){  *id='\0';   id++; }
266 char* nextNodeName=mstrDup(path);
267 node=getFirstChildNode();
268
269 if(node){
270  bool found = false;
271  while(!found){
272    if(!node->checkName(nextNodeName)){
273        node=node->getNextNode();
274        if(!node) found=true;
275    } else {
276        found=true;
277    }
278  }
279
280 }
281
282 if(node && id) {
283   node=node->findConfigNode(id);
284   *id='/'; // make path whole again
285 }
286
287 delete [] path;
288 delete [] nextNodeName;
289
290 return node;
291 }
```

4.20.2.12 StDbConfigNode * StDbConfigNode::findConfigNode (StDbType *t*, StDbDomain *d*)

Definition at line 225 of file StDbConfigNode.cc.

```
225                                                                 {
226
227   // Searches for the "highest" node of type & domain
228   // e.g. if request Calibrations , tpc & node is of that
229   // type & domain it checks parent 1st and if it is also
230   // that type & domain returns this call to the parent node
231   // .. else continue down the tree ...
232
233 StDbConfigNode* node = 0;
234
235 // search parent & this
236  if(isNode(type,domain)){
237     if(mparentNode && mparentNode->isNode(type,domain))
238       return mparentNode->findConfigNode(type,domain);
```

```
239     return this;
240  }
241
242 // search children & sibs
243    if(hasChildren())node=mfirstChildNode->findConfigNode(type,domain);
244    if(!node && mnextNode)node = mnextNode->findConfigNode(type,domain);
245
246 return node;
247 }
```

4.20.2.13 StDbConfigNode * StDbConfigNode::findConfigNode (StDbType *t*, StDbDomain *d*, const char * *sPath*)

Definition at line 217 of file StDbConfigNode.cc.

```
217                                                                                    {
218 StDbConfigNode* node=findConfigNode(type,domain);
219 return node->findConfigNode(subPath);
220 }
```

4.20.2.14 virtual StDbTable* StDbConfigNode::findLocalTable (const char * *name*) [pure virtual]

Implemented in StDbConfigNodeImpl.

4.20.2.15 virtual StDbTable* StDbConfigNode::findTable (const char * *name*, const char * *sPath* = "/") [pure virtual]

Implemented in StDbConfigNodeImpl.

4.20.2.16 int StDbConfigNode::getBranchID () [inline]

Definition at line 190 of file StDbConfigNode.hh.

```
190 { return mbranchID; };
```

4.20.2.17 virtual StDbElementIndex* StDbConfigNode::getElementIndex () [pure virtual]

Implemented in StDbConfigNodeImpl.

4.20.2.18 virtual void StDbConfigNode::getElementIndexInfo (char *& *indexname*, int & *indexID*) [pure virtual]

Implemented in StDbConfigNodeImpl.

4.20.2.19 StDbConfigNode * StDbConfigNode::getFirstChildNode () [inline]

Definition at line 188 of file StDbConfigNode.hh.

```
188 { return mfirstChildNode; };
```

4.20.2.20 StDbConfigNode * StDbConfigNode::getNextNode () [inline]

Definition at line 186 of file StDbConfigNode.hh.

```
186 { return mnextNode; };
```

4.20.2.21 virtual void StDbConfigNode::getNumberStats (unsigned int & *nNodes*, unsigned int & *ntables*, unsigned int & *numBytes*) [pure virtual]

Implemented in StDbConfigNodeImpl.

4.20.2.22 virtual int StDbConfigNode::getNumIndeces () const [pure virtual]

Implemented in StDbConfigNodeImpl.

4.20.2.23 StDbConfigNode * StDbConfigNode::getParentNode () [inline]

Definition at line 187 of file StDbConfigNode.hh.

```
187 { return mparentNode; };
```

4.20.2.24 virtual StDbTableIter* StDbConfigNode::getStDbTableIter () [pure virtual]

Implemented in StDbConfigNodeImpl.

4.20.2.25 bool StDbConfigNode::hasChildren () [inline]

Definition at line 194 of file StDbConfigNode.hh.

```
194 {return (mfirstChildNode) ? true : false; };
```

4.20.2.26 bool StDbConfigNode::hasData () [inline]

Definition at line 189 of file StDbConfigNode.hh.

```
189 { return mhasData;};
```

4.20.2.27 bool StDbConfigNode::isDbNode () [inline]

Definition at line 193 of file StDbConfigNode.hh.

```
193 { return misDbNode; };
```

4.20.2.28 virtual void StDbConfigNode::printNumberStats () [pure virtual]

Implemented in StDbConfigNodeImpl.

4.20.2.29 virtual void StDbConfigNode::printTables (int *depth*) [pure virtual]

Implemented in StDbConfigNodeImpl.

4.20.2.30 void StDbConfigNode::printTree (int *depth*)

Definition at line 344 of file StDbConfigNode.cc.

```
344                                 {
345
346 if(!depth)cout<<endl<<"************* Node Structure ****************" <<endl<<endl;
347
348   if(StDbManager::Instance()->IsVerbose()){
349     for(int k=0;k<depth;k++)cout<<" ";
350     cout<<"Node=" << mname <<" VersionKey = " << mversion <<endl;
351    if(mhasData) {
352      int depth2 = depth+4;
353      printTables(depth2);
354    }
355   }
356  int depth3=depth+4;
357  if(mfirstChildNode)mfirstChildNode->printTree(depth3);
358  if(mnextNode)mnextNode->printTree(depth);
359
360  if(!depth)cout<<endl<<"*******************************************" << endl;
361
362 }
```

4.20.2.31 virtual void StDbConfigNode::removeTable (StDbTable * *table*) [pure virtual]

Implemented in StDbConfigNodeImpl.

4.20.2.32 virtual void StDbConfigNode::resetConfig (const char * *config*, int *opt* = 0) [pure virtual]

Implemented in StDbConfigNodeImpl.

4.20.2.33 void StDbConfigNode::setBranchID (int *branchID*) [inline]

Definition at line 191 of file StDbConfigNode.hh.

```
191 { mbranchID=branchID; };
```

4.20.2.34 void StDbConfigNode::setChildNode (StDbConfigNode * *node*) [virtual]

Definition at line 324 of file StDbConfigNode.cc.

```
324                                                 {
325   if(mfirstChildNode) {
326     mfirstChildNode->appendNode(node);
327   } else {
328     mfirstChildNode = node;
329   }
330 }
```

4.20.2.35 virtual void StDbConfigNode::setElementIndexInfo (const char * *indexName*, int *indexID*) [pure virtual]

Implemented in StDbConfigNodeImpl.

4.20.2.36 void StDbConfigNode::setFirstChildNode (StDbConfigNode * *node*)

Definition at line 334 of file StDbConfigNode.cc.

```
334                                                     {
335   if(mfirstChildNode)node->appendNode(mfirstChildNode);
336   mfirstChildNode = 0;
337   setChildNode(node);
338 }
```

4.20.2.37 void StDbConfigNode::setFlavor (const char * *flavor*)

Definition at line 172 of file StDbConfigNode.cc.

```
172                                             {
173
174   setTablesFlavor(flavor);
175   if(mfirstChildNode)mfirstChildNode->setFlavor(flavor);
176   if(mnextNode)mnextNode->setFlavor(flavor);
177
178 }
```

4.20.2.38 void StDbConfigNode::setIsDbNode (bool *isDbNode*) [inline]

Definition at line 192 of file StDbConfigNode.hh.

```
192 {misDbNode=isDbNode; };
```

4.20.2.39 void StDbConfigNode::setNextNode (StDbConfigNode * *node*) [inline]

Definition at line 185 of file StDbConfigNode.hh.

```
185 { mnextNode=node;}
```

4.20.2.40 void StDbConfigNode::setParentNode (StDbConfigNode * *node*)

Definition at line 308 of file StDbConfigNode.cc.

```
308                                                     {
309 mparentNode = parent;
310 parent->setChildNode(this);
311 }
```

4.20.2.41 void StDbConfigNode::setProdTime (unsigned int *ptime*)

Definition at line 182 of file StDbConfigNode.cc.

```
182                                             {
183   setTablesProdTime(ptime);
184   if(mfirstChildNode)mfirstChildNode->setProdTime(ptime);
185   if(mnextNode)mnextNode->setProdTime(ptime);
186 }
```

4.20.2.42 virtual void StDbConfigNode::setTablesFlavor (const char * *flavor*) [pure virtual]

Implemented in StDbConfigNodeImpl.

4.20.2.43 virtual void StDbConfigNode::setTablesProdTime (unsigned int *ptime*) [pure virtual]

Implemented in StDbConfigNodeImpl.

4.20.2.44 void StDbConfigNode::zeroNodes () [protected]

Definition at line 160 of file StDbConfigNode.cc.

```
160                                 {
161   mfirstChildNode=0;
162   mnextNode=0;
163   mparentNode=0;
164   mhasData=false;
165   mbranchID=0;
166   misDbNode=false;
167 };
```

4.20.3 Member Data Documentation

4.20.3.1 int StDbConfigNode::mbranchID [protected]

Definition at line 108 of file StDbConfigNode.hh.

4.20.3.2 StDbConfigNode* StDbConfigNode::mfirstChildNode [protected]

Definition at line 103 of file StDbConfigNode.hh.

4.20.3.3 bool StDbConfigNode::mhasData [protected]

Definition at line 107 of file StDbConfigNode.hh.

4.20.3.4 bool StDbConfigNode::misDbNode [protected]

Definition at line 109 of file StDbConfigNode.hh.

4.20.3.5 StDbConfigNode* StDbConfigNode::mnextNode [protected]

Definition at line 104 of file StDbConfigNode.hh.

4.20.3.6 StDbConfigNode* StDbConfigNode::mparentNode [protected]

Definition at line 105 of file StDbConfigNode.hh.

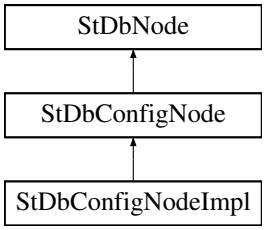
The documentation for this class was generated from the following files:

- StDbConfigNode.hh
- StDbConfigNode.cc

4.21 StDbConfigNodeImpl Class Reference

`#include <StDbConfigNodeImpl.hh>`

Inheritance diagram for StDbConfigNodeImpl::



Public Member Functions

- StDbConfigNodeImpl (StDbConfigNode *parent, const char *nodeName, const char *configName)
- StDbConfigNodeImpl (StDbConfigNode *parent, StDbNode &node)
- StDbConfigNodeImpl (StDbType type, StDbDomain domain, const char *nodeName, const char *configName="none")
- virtual ~StDbConfigNodeImpl ()
- virtual void addChildren (dbEnvList *elist)
- virtual void resetConfig (const char *config, int opt=0)
- virtual int buildTree (int opt=0)
- virtual StDbElementIndex * getElementIndex ()
- virtual void setElementIndexInfo (const char *indexName, int indexID)
- virtual void getElementIndexInfo (char *&indexname, int &indexID)
- virtual int getNumIndeces () const
- virtual StDbTable * addDbTable (const char *tableName, const char *version="default")
- virtual StDbTable * addTable (const char *tableName, const char *version="default")
- virtual StDbTable * addTable (StDbNode *node)
- virtual StDbTable * findTable (const char *name, const char *subPath="/")
- virtual StDbTable * findLocalTable (const char *name)
- virtual void removeTable (StDbTable *table)
- virtual StDbTableIter * getStDbTableIter ()
- virtual bool compareTables (StDbTable *tab1, StDbTable *tab2)
- virtual void printTables (int depth)
- virtual void printNumberStats ()
- virtual void getNumberStats (unsigned int &nNodes, unsigned int &ntables, unsigned int &numBytes)
- virtual void setTablesFlavor (const char *flavor)
- virtual void setTablesProdTime (unsigned int ptime)

Protected Member Functions

- void deleteTables ()
- void updateDbInfo ()
- void updateDbTables (int opt=0)
- void updateDbTable (StDbTable *table, int opt=0)

Protected Attributes

- StDbElementIndex * melementIndex
- int mindexRef
- TableList mTables

4.21.1 Constructor & Destructor Documentation

4.21.1.1 StDbConfigNodeImpl::StDbConfigNodeImpl (StDbConfigNode * *parent*, const char * *nodeName*, const char * *configName*)

Definition at line 65 of file StDbConfigNodeImpl.cc.

```

  melementIndex = new StDbElementIndex();
  setConfigured(false);
  mcanRollBack=false;
  updateDbInfo();
}
```

4.21.1.2 StDbConfigNodeImpl::StDbConfigNodeImpl (StDbConfigNode * *parent*, StDbNode & *node*)

Definition at line 74 of file StDbConfigNodeImpl.cc.

```
                                                                    : StDbConfigNode(parent,noc
  melementIndex = new StDbElementIndex();
  updateDbInfo();
}
```

4.21.1.3 StDbConfigNodeImpl::StDbConfigNodeImpl (StDbType *type*, StDbDomain *domain*, const char * *nodeName*, const char * *configName* = "none")

Definition at line 81 of file StDbConfigNodeImpl.cc.

```

  melementIndex = new StDbElementIndex();
  updateDbInfo();
}
```

4.21.1.4 StDbConfigNodeImpl::~StDbConfigNodeImpl () [virtual]

Definition at line 88 of file StDbConfigNodeImpl.cc.

```
                                        {
deleteTables();
delete melementIndex;
};
```

4.21.2 Member Function Documentation

4.21.2.1 void StDbConfigNodeImpl::addChildren (dbEnvList * *elist*) [virtual]

Implements StDbConfigNode.

Definition at line 385 of file StDbConfigNodeImpl.cc.

```
385                                                 {
386 #define __METHOD__ "addChildren(dbEnvList*)"
387
388   for(int i=0; i<elist->num; i++){
389     char* id=strstr(elist->envVar[i],"_");
390     if(id){
391       id++;
392       if(strcmp(id,mname) && !findChildConfigNode(id)){
393         StString nm;
394         nm<<" Adding DataBase="<<elist->envVar[i]<<" with KEY=";
395         nm<<elist->envDef[i]<<" from Environment variable definition";
396         StDbManager::Instance()->printInfo((nm.str()).c_str(),dbMWarn,__LINE__,__CLASS__,__METHOD__);
397 //VP        delete [] nMes;
398             new StDbConfigNodeImpl(this,id,elist->envDef[i]);
399       }
400     }
401   }
402 #undef __METHOD__
403 }
```

4.21.2.2 StDbTable * StDbConfigNodeImpl::addDbTable (const char * *tableName*, const char * *version* = "default") [virtual]

Implements StDbConfigNode.

Definition at line 204 of file StDbConfigNodeImpl.cc.

```
204                                                                         {
205
206   // just like addTable but also loads the descriptor from the database
207   StDbTable*   table = addTable(tableName,version);
208   updateDbTable(table);
209   return table;
210 }
```

4.21.2.3 StDbTable * StDbConfigNodeImpl::addTable (StDbNode * *node*) [virtual]

Implements StDbConfigNode.

Definition at line 229 of file StDbConfigNodeImpl.cc.

```
229                                                 {
230
231   StDbTable* table = StDbManager::Instance()->newDbTable(node);
232   if(table){
233      mTables.push_back(table);
234      mhasData=true;
235   }
236
237 return table;
238 }
```

4.21.2.4 StDbTable * StDbConfigNodeImpl::addTable (const char * *tableName*, const char * *version* = "default") [virtual]

Implements StDbConfigNode.

Definition at line 215 of file StDbConfigNodeImpl.cc.

```
215                                                                    {
216
217   StDbTable* table = StDbManager::Instance()->newDbTable(mdbName,tableName);
218   if(table) mTables.push_back(table);
219   table->setVersion((char*)version);
220   table->setNodeType("table");
221
222   StDataBaseI* db = StDbManager::Instance()->findDb(mdbType, mdbDomain);
223   if(db && table && db->QueryDb((StDbNode*)table))mhasData=true;
224 return table;
225 }
```

4.21.2.5 int StDbConfigNodeImpl::buildTree (int *opt* = 0) [virtual]

Implements StDbConfigNode.

Definition at line 138 of file StDbConfigNodeImpl.cc.

```
138                                    {
139 #define __METHOD__ "buildTree(int opt)"
140
141  StDataBaseI *db = StDbManager::Instance()->findDb(mdbType, mdbDomain);
142
143  if(!db)
144   return StDbManager::Instance()->printInfo(" No DB found for Node=",mname,dbMErr,__LINE__,__CLASS__,_
145
146  if(db->QueryDb(this)){
147   // check if we should connect via environment variables
148   if(isDbNode() && mdbDomain==dbStar){
149     dbEnvList* elist = StDbManager::Instance()->getEnvList(mname);
150     if(elist){
151       addChildren(elist);
152       delete elist;
153     }
154   }
155   // load descriptors & element structure as needed
156   updateDbTables(opt);
157   if(mfirstChildNode)mfirstChildNode->buildTree(opt);
158  }
159
160  if(mnextNode)mnextNode->buildTree(opt);
161
162  return 1;
163 #undef __METHOD__
164 }
```

4.21.2.6 bool StDbConfigNodeImpl::compareTables (StDbTable * *tab1*, StDbTable * *tab2*) [virtual]

Implements StDbConfigNode.

Definition at line 309 of file StDbConfigNodeImpl.cc.

```
309                                                                 {
310
311 bool retVal=false;
312 // compare name & version
313  if((strcmp(tab1->printName(),tab2->printName())!=0)) return retVal;
314  if((strcmp(tab1->printVersion(),tab2->printVersion())!=0)) return retVal;
315 // compare each row identifier
316     int nRows1; int nRows2;
317     int* elements1 = tab1->getElementID(nRows1);
318     int* elements2 = tab2->getElementID(nRows2);
319     if(nRows1 != nRows2) return retVal;
320     int i;
321     for(i=0;i<nRows1;i++)if(elements1[i] != elements2[i])break;
322     if(i==nRows1)retVal=true;
323 return retVal;
324 }
```

4.21.2.7 void StDbConfigNodeImpl::deleteTables () [protected]

Definition at line 369 of file StDbConfigNodeImpl.cc.

```
369                                 {
370
371   TableList::iterator itr;
372   StDbTable* table;
373   do {  for(itr = mTables.begin(); itr!=mTables.end(); ++itr){
374             table=*itr;
375             mTables.erase(itr);
376             delete table;
377             break;
378          }
379   } while (mTables.begin() != mTables.end());
380   mhasData=false;
381 }
```

4.21.2.8 StDbTable * StDbConfigNodeImpl::findLocalTable (const char * *name*) [virtual]

Implements StDbConfigNode.

Definition at line 257 of file StDbConfigNodeImpl.cc.

```
257                                                 {
258
259   TableList::iterator itr;
260   StDbTable* table=0;
261     for(itr = mTables.begin(); itr!=mTables.end(); ++itr){
262       if((*itr)->checkName(name)){
263         table=*itr;
264         break;
265       }
266     }
267 return table;
268 }
```

4.21.2.9 StDbTable * StDbConfigNodeImpl::findTable (const char * *name*, const char * *subPath* = "/") [virtual]

Implements StDbConfigNode.

Definition at line 242 of file StDbConfigNodeImpl.cc.

```
242                                                                 {
243
244 if(subPath && (strcmp(subPath,"/")==0))return findLocalTable(name);
245
246 StDbConfigNode* node = findConfigNode(subPath);
247 if(node)return node->findLocalTable(name);
248
249 // if we got here then we can't find it
250  StDbTable* table=0;
251 return table;
252 }
```

4.21.2.10 StDbElementIndex * StDbConfigNodeImpl::getElementIndex () [virtual]

Implements StDbConfigNode.

Definition at line 327 of file StDbConfigNodeImpl.cc.

```
327 {return melementIndex; }
```

4.21.2.11 void StDbConfigNodeImpl::getElementIndexInfo (char *& *indexname*, int & *indexID*) [virtual]

Implements StDbConfigNode.

Definition at line 337 of file StDbConfigNodeImpl.cc.

```
337                                                                 {
338
339   if(mindexRef < 0){ indexName=0; return; }
340     indexName=melementIndex->getIndexName(mindexRef);
341     indexID  =melementIndex->getIndexVal(mindexRef);
342 }
```

4.21.2.12 void StDbConfigNodeImpl::getNumberStats (unsigned int & *nNodes*, unsigned int & *ntables*, unsigned int & *numBytes*) [virtual]

Implements StDbConfigNode.

Definition at line 450 of file StDbConfigNodeImpl.cc.

```
450
451
452  numNodes++;
453  StDbTableIter* itr = getStDbTableIter();
454  StDbTable* table;
455  while((itr) && !(itr->done())){
456     table=itr->next();
457     numTables++;
458     numBytes+=(table->getTableSize()*table->GetNRows());
459    }
460  if(itr)delete itr;
461  if(mfirstChildNode)mfirstChildNode->getNumberStats(numNodes,numTables,numBytes);
462  if(mnextNode)mnextNode->getNumberStats(numNodes,numTables,numBytes);
463 }
```

4.21.2.13 int StDbConfigNodeImpl::getNumIndeces () const [virtual]

Implements StDbConfigNode.

Definition at line 346 of file StDbConfigNodeImpl.cc.

```
346 { return melementIndex->getNumIndeces();}
```

4.21.2.14 StDbTableIter * StDbConfigNodeImpl::getStDbTableIter () [virtual]

Implements StDbConfigNode.

Definition at line 350 of file StDbConfigNodeImpl.cc.

```
350                                     {
351   StDbTableIter* itr = new StDbTableIterImpl(this);
352   return itr;
353 }
```

4.21.2.15 void StDbConfigNodeImpl::printNumberStats () [virtual]

Implements StDbConfigNode.

Definition at line 428 of file StDbConfigNodeImpl.cc.

```
428                                      {
429 #define __METHOD__ "printNumberStats()"
430
431 unsigned int numNodes, numTables, numBytes;
432  numNodes=numTables=numBytes=0;
433  getNumberStats(numNodes,numTables, numBytes);
434  double kbs = ((double)numBytes)/1000.0;
435  StString cos;
436  cos<<"******************** Number Stats ******************** "<<stendl;
437  cos<<"Total Number of Nodes        = "<<numNodes              <<stendl;
438  cos<<"Total Number of Tables       = "<<numTables             <<stendl;
439  cos<<"Total Size of Data in Tables = "<<kbs<<" kBytes"        <<stendl;
440  cos<<"*****************************************************" <<stendl;
441
442  StDbManager::Instance()->printInfo((cos.str()).c_str(),dbMConnect,__LINE__,__CLASS__,__METHOD__);
443 //VP delete [] nstats;
444
445 #undef __METHOD__
446 }
```

4.21.2.16 void StDbConfigNodeImpl::printTables (int *depth*) [virtual]

Implements StDbConfigNode.

Definition at line 407 of file StDbConfigNodeImpl.cc.

```
407                                         {
408
409   if(StDbManager::Instance()->IsVerbose()){
410     StString os;
411     for(int k=0;k<depth;k++)os<<" ";
```

```

    string pdepth=os.str();
   TableList::iterator itr;
   for(itr = mTables.begin(); itr!=mTables.end(); ++itr){
     cout<<pdepth<<"Table="<<(*itr)->printName()<<", Version="<<(*itr)->printVersion();
      int nRows;
      int* eIDs=(*itr)->getElementID(nRows);
      cout <<", numRows="<<nRows;
      if(nRows==1)cout<<", elementID="<<eIDs[0];
      cout<<endl;
    }
//VP    delete [] pdepth;
  }
}
```

4.21.2.17 void StDbConfigNodeImpl::removeTable (StDbTable * *table*) [virtual]

Implements StDbConfigNode.

Definition at line 272 of file StDbConfigNodeImpl.cc.

```
                                                {

  if(!table)return;
  TableList::iterator itr;
  StDbTable* myTable=0;

    for(itr = mTables.begin(); itr!=mTables.end(); ++itr){
        myTable=*itr;
        if(myTable && compareTables(myTable,table)){
                mTables.erase(itr);   break;
        }
        myTable=0;
    }
}
```

4.21.2.18 void StDbConfigNodeImpl::resetConfig (const char * *config*, int *opt* = 0) [virtual]

Implements StDbConfigNode.

Definition at line 358 of file StDbConfigNodeImpl.cc.

```
                                                                {

if(mfirstChildNode)mfirstChildNode->deleteTree();
if(mhasData)deleteTables();
setVersion(configName);
buildTree(opt);
}
```

4.21.2.19 void StDbConfigNodeImpl::setElementIndexInfo (const char * *indexName*, int *indexID*) [virtual]

Implements StDbConfigNode.

Definition at line 331 of file StDbConfigNodeImpl.cc.

```
331                                                                    {
332   mindexRef=melementIndex->addNameValuePair(indexName,indexID);
333 }
```

4.21.2.20 void StDbConfigNodeImpl::setTablesFlavor (const char *flavor) [virtual]

Implements StDbConfigNode.

Definition at line 289 of file StDbConfigNodeImpl.cc.

```
289                                                        {
290
291   TableList::iterator itr;
292   for(itr = mTables.begin(); itr!=mTables.end(); ++itr){
293       if((*itr))(*itr)->setFlavor(flavor);
294    }
295 }
```

4.21.2.21 void StDbConfigNodeImpl::setTablesProdTime (unsigned int ptime) [virtual]

Implements StDbConfigNode.

Definition at line 299 of file StDbConfigNodeImpl.cc.

```
299                                                          {
300
301   TableList::iterator itr;
302   for(itr = mTables.begin(); itr!=mTables.end(); ++itr){
303       if((*itr))(*itr)->setProdTime(ptime);
304    }
305 }
```

4.21.2.22 void StDbConfigNodeImpl::updateDbInfo () [protected]

Definition at line 95 of file StDbConfigNodeImpl.cc.

```
95                                  {
96 #define __METHOD__ "updateDbInfo()"
97
98  if(mparentNode)melementIndex->addElementIndex(mparentNode->getElementIndex());
99
100   // If StarDb Type, then name holds map to real Db type
101   // while the domain may still be StarDb
102   // Else if StarDb domain, then name holds map to real domain type
103   // Else both type & domain are that of parent (which maybe User defined)
104
105   misConfigured = false;
106   if(mparentNode){ //top level node will have type&domain set by the manager.
107    if(mdbType==dbStDb){
108     mdbType=StDbManager::Instance()->getDbType(mname);
109    } else if(mdbDomain==dbStar) {
110     mdbDomain=StDbManager::Instance()->getDbDomain(mname);
111    }
112   }
113
114   if(!mnodeType) mnodeType=mstrDup("DB");
115   // replace nodetype of DB to Config moving from 1 db to another
```

```
116   // & allow overwriting of versionkey with environment variable
117   if(strcmp(mnodeType,"DB")==0){
118      setIsDbNode(true);
119      delete [] mnodeType;
120      mnodeType=mstrDup("Config");
121      setBranchID(0);
122      char* version=StDbManager::Instance()->getExternalVersion(mdbType,mdbDomain);
123      if(version && strcmp(version,mversion)){
124        StString nm;
125        nm<<" Overriding Key="<<mversion<<" with Environment Var Key="<<version;
126        nm<<" for DataBase=";
127        nm<<StDbManager::Instance()->printDbName(mdbType,mdbDomain);
128        StDbManager::Instance()->printInfo((nm.str()).c_str(),dbMWarn,__LINE__,__CLASS__,__METHOD__);
129        setVersion(version);
130      }
131   }
132
133 #undef __METHOD__
134 }
```

4.21.2.23 void StDbConfigNodeImpl::updateDbTable (StDbTable * *table*, int *opt* = 0) [protected]

Definition at line 183 of file StDbConfigNodeImpl.cc.

```
183                                                                    {
184
185  if(!table) return;
186  StDataBaseI* db=StDbManager::Instance()->findDb(mdbType, mdbDomain);
187  if(!db)return;
188
189  int  nRows;
190  char* elements=table->printElementName();
191  if(elements){
192     int* eList=db->selectElements(elements,melementIndex,nRows);
193     if(eList){
194        table->setElementID(eList,nRows);
195        delete [] eList;
196     }
197   }
198  if(!opt) db->QueryDescriptor(table);
199 }
```

4.21.2.24 void StDbConfigNodeImpl::updateDbTables (int *opt* = 0) [protected]

Definition at line 167 of file StDbConfigNodeImpl.cc.

```
167                                                 {
168
169  if(!mhasData)return;
170  StDataBaseI* db=StDbManager::Instance()->findDb(mdbType, mdbDomain);
171  if(!db) return;
172
173  StDbTableIter* itr = getStDbTableIter();
174  StDbTable* table;
175  while((itr) && !(itr->done())){
176     table=itr->next();
177     updateDbTable(table,opt);
178    }
179  if(itr)delete itr;
180
181 }
```

4.21.3 Member Data Documentation

4.21.3.1 StDbElementIndex* StDbConfigNodeImpl::melementIndex [protected]

Definition at line 52 of file StDbConfigNodeImpl.hh.

4.21.3.2 int StDbConfigNodeImpl::mindexRef [protected]

Definition at line 53 of file StDbConfigNodeImpl.hh.

4.21.3.3 TableList StDbConfigNodeImpl::mTables [protected]

Definition at line 54 of file StDbConfigNodeImpl.hh.

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

- StDbConfigNodeImpl.hh
- StDbConfigNodeImpl.cc

4.22 StDbDefaults Class Reference

```
#include <StDbDefaults.hh>
```

Public Member Functions

- ~StDbDefaults ()
- bool IsDefaultVersion (const char *version)
- bool IsDefaultFlavor (const char *flavor)
- char * printVersion ()
- char * printFlavor ()
- unsigned int getProdTime () const
- unsigned int getEndTime () const
- char * printDbServerVar ()
- char * printDbServerFile ()
- char * getServerFileName (const dbFindServerMode mode)
- char * getFileName (const char *fileName)

Static Public Member Functions

- StDbDefaults * Instance ()

Private Member Functions

- StDbDefaults ()

Private Attributes

- char mversion [64]
- char mflavor [16]
- unsigned int mprodTime
- unsigned int mendTime
- char mdbServerVar [16]
- char mdbServerFile [16]

Static Private Attributes

- StDbDefaults * mInstance = 0

4.22.1 Constructor & Destructor Documentation

4.22.1.1 StDbDefaults::StDbDefaults () [private]

Definition at line 8 of file StDbDefaults.cc.

```
8                           {
9
10  strncpy(mversion,"default",sizeof(mversion));
11  strncpy(mflavor,"ofl",sizeof(mflavor));
12  mprodTime = 0;
13  mendTime =2145873600; // Dec 31, 2037
14  strncpy(mdbServerVar,"STDB_SERVERS",sizeof(mdbServerVar));
15  strncpy(mdbServerFile,"dbServers.xml",sizeof(mdbServerFile));
16
17 }
```

4.22.1.2 StDbDefaults::~StDbDefaults () [inline]

Definition at line 62 of file StDbDefaults.hh.

```
62 {};
```

4.22.2 Member Function Documentation

4.22.2.1 unsigned int StDbDefaults::getEndTime () const [inline]

Definition at line 100 of file StDbDefaults.hh.

```
100 { return mendTime; }
```

4.22.2.2 char * StDbDefaults::getFileName (const char *fileName*)

Definition at line 31 of file StDbDefaults.cc.

```
31                                                  {
32
33   char* nullReturn=0;
34   if(!fileName) return nullReturn;
35
36   StString fn;
37
38   if(strcmp(fileName,"HOME")==0){
39     fn<<getenv("HOME")<<"/"<<mdbServerFile;
40   } else if(strcmp(fileName,"STAR")==0){
41     fn<<getenv("STAR")<<"/"<<"StDb/servers/"<<mdbServerFile;
42   } else {
43     char* fname=getenv(fileName);
44     if(!fname)return nullReturn;
45     fn<<fname;
46     if(opendir(fname))fn<<"/"<<mdbServerFile;
47   }
48
49   string fns=fn.str();
50   char* retVal= new char[fns.length()+1];
51   strcpy(retVal,fns.c_str());
52
53   return retVal;
54 }
```

4.22.2.3 unsigned int StDbDefaults::getProdTime () const [inline]

Definition at line 97 of file StDbDefaults.hh.

```
97 { return mprodTime; }
```

4.22.2.4 char * StDbDefaults::getServerFileName (const dbFindServerMode *mode*)

Definition at line 20 of file StDbDefaults.cc.

```
20                                                                  {
21
22   if(mode==userHome) return getFileName("HOME");
23   if(mode==serverEnvVar) return getFileName(mdbServerVar);
24   if(mode==starDefault) return getFileName("STAR");
25
26   return NULL;
27 };
```

4.22.2.5 StDbDefaults* StDbDefaults::Instance () [inline, static]

Definition at line 57 of file StDbDefaults.hh.

```
57                                  {
58          if(!mInstance)mInstance = new StDbDefaults;
59          return mInstance;
60   }
```

4.22.2.6 bool StDbDefaults::IsDefaultFlavor (const char * *flavor*) [inline]

Definition at line 85 of file StDbDefaults.hh.

```
85                                                      {
86 if(strcmp(flavor,mflavor)==0)return true;
87 return false;
88 }
```

4.22.2.7 bool StDbDefaults::IsDefaultVersion (const char * *version*) [inline]

Definition at line 79 of file StDbDefaults.hh.

```
79                                                        {
80 if(strcmp(version,mversion)==0)return true;
81 return false;
82 }
```

4.22.2.8 char * StDbDefaults::printDbServerFile () [inline]

Definition at line 106 of file StDbDefaults.hh.

```
106 { return mdbServerFile; };
```

4.22.2.9 char * StDbDefaults::printDbServerVar () [inline]

Definition at line 103 of file StDbDefaults.hh.

```
103 { return mdbServerVar; };
```

4.22.2.10 char * StDbDefaults::printFlavor () [inline]

Definition at line 94 of file StDbDefaults.hh.

```
94 { return mflavor; }
```

4.22.2.11 char * StDbDefaults::printVersion () [inline]

Definition at line 91 of file StDbDefaults.hh.

```
91 { return mversion; }
```

4.22.3 Member Data Documentation

4.22.3.1 char StDbDefaults::mdbServerFile[16] [private]

Definition at line 49 of file StDbDefaults.hh.

4.22.3.2 char StDbDefaults::mdbServerVar[16] [private]

Definition at line 48 of file StDbDefaults.hh.

4.22.3.3 unsigned int StDbDefaults::mendTime [private]

Definition at line 47 of file StDbDefaults.hh.

4.22.3.4 char StDbDefaults::mflavor[16] [private]

Definition at line 45 of file StDbDefaults.hh.

4.22.3.5 StDbDefaults * StDbDefaults::mInstance = 0 [static, private]

Definition at line 6 of file StDbDefaults.cc.

4.22.3.6 unsigned int StDbDefaults::mprodTime [private]

Definition at line 46 of file StDbDefaults.hh.

4.22.3.7 char StDbDefaults::mversion[64] [private]

Definition at line 44 of file StDbDefaults.hh.

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

- StDbDefaults.hh
- StDbDefaults.cc

4.23 StDbElementIndex Class Reference

```
#include <StDbElementIndex.hh>
```

Public Member Functions

- StDbElementIndex ()
- StDbElementIndex (StDbElementIndex &inval)
- virtual ~StDbElementIndex ()
- virtual void clearIndex ()
- virtual void addElementIndex (StDbElementIndex *inval)
- virtual int addNameValuePair (const char *name, int ival)
- virtual int getNumIndeces () const
- virtual int getIndexVal (int indexNumber)
- virtual char * getIndexName (int indexNumber)
- virtual char * printIndexName (int indexNumber)
- virtual void resetCounter ()
- virtual char * getNextIndex (int &indexVal)
- virtual char * printNextIndex (int &indexVal)

Private Attributes

- int mnumIndeces
- indexNameVals mnvals [N_MAX_INDEXVALS]
- int mcurrent

4.23.1 Constructor & Destructor Documentation

4.23.1.1 StDbElementIndex::StDbElementIndex ()

Definition at line 35 of file StDbElementIndex.cc.

```
35 {  clearIndex(); };
```

4.23.1.2 StDbElementIndex::StDbElementIndex (StDbElementIndex & *inval*)

Definition at line 37 of file StDbElementIndex.cc.

```
37                                                          {
38
39   clearIndex();
40   mnumIndeces = inval.getNumIndeces();
41   inval.resetCounter();
42   char* ctest;
43   int ival;
44   while ( (ctest=inval.printNextIndex(ival)) ){
45     strcpy(mnvals[mcurrent].iname,ctest);
46     mnvals[mcurrent].ival = ival;
47     mcurrent++;
48   }
49   resetCounter();
50   inval.resetCounter();
51 }
```

4.23.1.3 virtual StDbElementIndex::~StDbElementIndex () [inline, virtual]

Definition at line 55 of file StDbElementIndex.hh.

```
55 {};
```

4.23.2 Member Function Documentation

4.23.2.1 void StDbElementIndex::addElementIndex (StDbElementIndex * *inval*) [virtual]

Definition at line 55 of file StDbElementIndex.cc.

```
55                                                          {
56
57   if(!inval) return;
58   clearIndex();
59   mnumIndeces = inval->getNumIndeces();
60   inval->resetCounter();
61   char* ctest;
62   int ival;
63   while ( (ctest=inval->printNextIndex(ival))){
64     strcpy(mnvals[mcurrent].iname,ctest);
65     mnvals[mcurrent].ival = ival;
66     mcurrent++;
67   }
68
69   resetCounter();
70   inval->resetCounter();
71
72 }
```

4.23.2.2 int StDbElementIndex::addNameValuePair (const char * *name*, int *ival*) [virtual]

Definition at line 76 of file StDbElementIndex.cc.

```
76                                                          {
77
78   strcpy(mnvals[mnumIndeces].iname,name);
79   mnvals[mnumIndeces].ival = ival;
80   int retVal=mnumIndeces;
81   mnumIndeces++;
82   return retVal;
83 };
```

4.23.2.3 void StDbElementIndex::clearIndex () [inline, virtual]

Definition at line 72 of file StDbElementIndex.hh.

```
72                                          {
73  mcurrent=0; mnumIndeces=0;
74  memset(&mnvals,0,N_MAX_INDEXVALS*sizeof(indexNameVals));
75 }
```

4.23.2.4 char * StDbElementIndex::getIndexName (int *indexNumber*) [virtual]

Definition at line 88 of file StDbElementIndex.cc.

```
88                                                {
89   char* retVal=0;
90   if(indexNumber<mnumIndeces){
91     retVal = new char[strlen((char*)mnvals[indexNumber].iname)+1];
92     strcpy(retVal,(char*)mnvals[indexNumber].iname);
93   }
94   return retVal;
95 }
```

4.23.2.5 int StDbElementIndex::getIndexVal (int *indexNumber*) [inline, virtual]

Definition at line 78 of file StDbElementIndex.hh.

```
78                                                        {
79   if(indexNum>=N_MAX_INDEXVALS)return -1;
80   return mnvals[indexNum].ival;
81 }
```

4.23.2.6 char * StDbElementIndex::getNextIndex (int & *indexVal*) [virtual]

Definition at line 100 of file StDbElementIndex.cc.

```
100                                               {
101   char* retVal=0;
102   if(mcurrent<mnumIndeces){
103     retVal = new char[strlen((char*)mnvals[mcurrent].iname)+1];
104     strcpy(retVal,(char*)mnvals[mcurrent].iname);
105     indexVal=mnvals[mcurrent].ival;
106     mcurrent++;
107   }
108   return retVal;
109 }
```

4.23.2.7 int StDbElementIndex::getNumIndeces () const [inline, virtual]

Definition at line 76 of file StDbElementIndex.hh.

```
76 { return mnumIndeces; }
```

4.23.2.8 char * StDbElementIndex::printIndexName (int *indexNumber*) [inline, virtual]

Definition at line 82 of file StDbElementIndex.hh.

```
82                                                        {
83   if(indexNum>=N_MAX_INDEXVALS)return (char*)0;
84   return (char*)mnvals[indexNum].iname;
85 }
```

4.23.2.9 char * StDbElementIndex::printNextIndex (int & *indexVal*) [virtual]

Definition at line 113 of file StDbElementIndex.cc.

```
113                                                 {
114   char* retVal=0;
115   if(mcurrent<mnumIndeces){
116     retVal = (char*)mnvals[mcurrent].iname;
117     indexVal=mnvals[mcurrent].ival;
118     mcurrent++;
119   }
120   return retVal;
121 }
```

4.23.2.10 void StDbElementIndex::resetCounter () [inline, virtual]

Definition at line 86 of file StDbElementIndex.hh.

```
86 { mcurrent=0; }
```

4.23.3 Member Data Documentation

4.23.3.1 int StDbElementIndex::mcurrent [private]

Definition at line 49 of file StDbElementIndex.hh.

4.23.3.2 int StDbElementIndex::mnumIndeces [private]

Definition at line 47 of file StDbElementIndex.hh.

4.23.3.3 indexNameVals StDbElementIndex::mnvals[N_MAX_INDEXVALS] [private]

Definition at line 48 of file StDbElementIndex.hh.

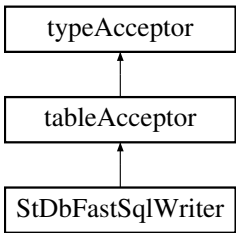
The documentation for this class was generated from the following files:

- StDbElementIndex.hh
- StDbElementIndex.cc

4.24 StDbFastSqlWriter Class Reference

`#include <StDbFastSqlWriter.h>`

Inheritance diagram for StDbFastSqlWriter::



Public Member Functions

- StDbFastSqlWriter ()
- StDbFastSqlWriter (StString &ofs)
- virtual ~StDbFastSqlWriter ()
- virtual void streamHeader (const char *name)
- virtual void streamTableName (const char *name)
- virtual void streamEndTableName ()
- virtual void streamAccessor ()
- virtual void endAccessor ()
- virtual void streamRow (int row)
- virtual void streamEndRow ()
- virtual void streamTail ()
- virtual void ioTable (StDbTable *table)
- virtual void pass (char *name, short &i, int &len)
- virtual void pass (char *name, int &i, int &len)
- virtual void pass (char *name, long &i, int &len)
- virtual void pass (char *name, unsigned short &i, int &len)
- virtual void pass (char *name, unsigned int &i, int &len)
- virtual void pass (char *name, unsigned long &i, int &len)
- virtual void pass (char *name, long long &i, int &len)
- virtual void pass (char *name, float &i, int &len)
- virtual void pass (char *name, double &i, int &len)
- virtual void pass (char *name, char *&i, int &len)
- virtual void pass (char *name, unsigned char &i, int &len)
- virtual void pass (char *name, unsigned char *&i, int &len)
- virtual void pass (char *name, short *&i, int &len)
- virtual void pass (char *name, int *&i, int &len)
- virtual void pass (char *name, long *&i, int &len)
- virtual void pass (char *name, unsigned short *&i, int &len)
- virtual void pass (char *name, unsigned int *&i, int &len)
- virtual void pass (char *name, unsigned long *&i, int &len)
- virtual void pass (char *name, long long *&i, int &len)
- virtual void pass (char *name, float *&i, int &len)
- virtual void pass (char *name, double *&i, int &len)

Protected Attributes

- StString * os

4.24.1 Constructor & Destructor Documentation

4.24.1.1 StDbFastSqlWriter::StDbFastSqlWriter () [inline]

Definition at line 70 of file StDbFastSqlWriter.h.

```
70 : os(0) {};
```

4.24.1.2 StDbFastSqlWriter::StDbFastSqlWriter (StString & *ofs*) [inline]

Definition at line 71 of file StDbFastSqlWriter.h.

```
71 { os=&ofs;};
```

4.24.1.3 virtual StDbFastSqlWriter::~StDbFastSqlWriter () [inline, virtual]

Definition at line 72 of file StDbFastSqlWriter.h.

```
72 {};
```

4.24.2 Member Function Documentation

4.24.2.1 virtual void StDbFastSqlWriter::endAccessor () [inline, virtual]

Implements tableAcceptor.

Definition at line 78 of file StDbFastSqlWriter.h.

```
78 {/* no-op */ };
```

4.24.2.2 void StDbFastSqlWriter::ioTable (StDbTable * *table*) [virtual]

Implements tableAcceptor.

Definition at line 5 of file StDbFastSqlWriter.cc.

```
5                                          {
6
7   table->setRowNumber();
8   int nrows=table->GetNRows();
9   for(int k=0;k<nrows;k++){
10     if(k>0)*os<<"),(";
11     *os<<table->getBeginTime();
12     table->dbStreamer((typeAcceptor*)this,false);
13   }
14
15 }
```

4.24.2.3 void StDbFastSqlWriter::pass (char * *name*, double *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 82 of file StDbFastSqlWriter.cc.

```
82                                                          {
83   *os<<",'"<<i[0];
84   for(int j=1;j<len;j++)*os<<","<<i[j];
85   *os<<"'";
86 }
```

4.24.2.4 void StDbFastSqlWriter::pass (char * *name*, float *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 75 of file StDbFastSqlWriter.cc.

```
75                                                          {
76   *os<<",'"<<i[0];
77   for(int j=1;j<len;j++)*os<<","<<i[j];
78   *os<<"'";
79 }
```

4.24.2.5 void StDbFastSqlWriter::pass (char * *name*, long long *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 68 of file StDbFastSqlWriter.cc.

```
68                                                          {
69   *os<<",'"<<i[0];
70   for(int j=1;j<len;j++)*os<<","<<i[j];
71   *os<<"'";
72 }
```

4.24.2.6 void StDbFastSqlWriter::pass (char * *name*, unsigned long *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 61 of file StDbFastSqlWriter.cc.

```
61                                                          {
62   *os<<",'"<<i[0];
63   for(int j=1;j<len;j++)*os<<","<<i[j];
64   *os<<"'";
65 }
```

4.24.2.7 void StDbFastSqlWriter::pass (char * *name*, unsigned int *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 54 of file StDbFastSqlWriter.cc.

```
54                                                                  {
55   *os<<",'"<<i[0];
56   for(int j=1;j<len;j++)*os<<","<<i[j];
57   *os<<"'";
58 }
```

4.24.2.8 void StDbFastSqlWriter::pass (char * *name*, unsigned short *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 47 of file StDbFastSqlWriter.cc.

```
47                                                                  {
48   *os<<",'"<<i[0];
49   for(int j=1;j<len;j++)*os<<","<<i[j];
50   *os<<"'";
51 }
```

4.24.2.9 void StDbFastSqlWriter::pass (char * *name*, long *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 39 of file StDbFastSqlWriter.cc.

```
39                                                                  {
40   *os<<",'"<<i[0];
41   for(int j=1;j<len;j++)*os<<","<<i[j];
42   *os<<"'";
43 }
```

4.24.2.10 void StDbFastSqlWriter::pass (char * *name*, int *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 32 of file StDbFastSqlWriter.cc.

```
32                                                                  {
33   *os<<",'"<<i[0];
34   for(int j=1;j<len;j++)*os<<","<<i[j];
35   *os<<"'";
36 }
```

4.24.2.11 void StDbFastSqlWriter::pass (char * *name*, short *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 25 of file StDbFastSqlWriter.cc.

```
25                                                                  {
26   *os<<",'"<<i[0];
27   for(int j=1;j<len;j++)*os<<","<<i[j];
28   *os<<"'";
29 }
```

4.24.2.12 void StDbFastSqlWriter::pass (char * *name*, unsigned char *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 18 of file StDbFastSqlWriter.cc.

```
18                                                                  {
19   *os<<",'"<<(int)i[0];
20   for(int j=1;j<len;j++)*os<<","<<(int)i[j];
21   *os<<"'";
22 }
```

4.24.2.13 void StDbFastSqlWriter::pass (char * *name*, unsigned char & *i*, int & *len*) [inline, virtual]

Implements typeAcceptor.

Definition at line 116 of file StDbFastSqlWriter.h.

```
116 { *os<<(int)i; };
```

4.24.2.14 void StDbFastSqlWriter::pass (char * *name*, char *& *i*, int & *len*) [inline, virtual]

Implements typeAcceptor.

Definition at line 122 of file StDbFastSqlWriter.h.

```
122                                                                 {
123 *os<<",'"<<i<<"'"; };
```

4.24.2.15 void StDbFastSqlWriter::pass (char * *name*, double & *i*, int & *len*) [inline, virtual]

Implements typeAcceptor.

Definition at line 121 of file StDbFastSqlWriter.h.

```
121 { *os<<","<<i; };
```

4.24.2.16 void StDbFastSqlWriter::pass (char * *name*, float & *i*, int & *len*) [inline, virtual]

Implements typeAcceptor.

Definition at line 120 of file StDbFastSqlWriter.h.

```
120 { *os<<","<<i; };
```

4.24.2.17 void StDbFastSqlWriter::pass (char * *name*, long long & *i*, int & *len*) [inline, virtual]

Implements typeAcceptor.

Definition at line 114 of file StDbFastSqlWriter.h.

```
114 { *os<<","<<i; };
```

4.24.2.18 void StDbFastSqlWriter::pass (char * *name*, unsigned long & *i*, int & *len*) [inline, virtual]

Implements typeAcceptor.

Definition at line 118 of file StDbFastSqlWriter.h.

```
118 { *os<<","<<i; };
```

4.24.2.19 void StDbFastSqlWriter::pass (char * *name*, unsigned int & *i*, int & *len*) [inline, virtual]

Implements typeAcceptor.

Definition at line 117 of file StDbFastSqlWriter.h.

```
117 { *os<<","<<i; };
```

4.24.2.20 void StDbFastSqlWriter::pass (char * *name*, unsigned short & *i*, int & *len*) [inline, virtual]

Implements typeAcceptor.

Definition at line 115 of file StDbFastSqlWriter.h.

```
115 { *os<<","<<i; };
```

4.24.2.21 void StDbFastSqlWriter::pass (char * *name*, long & *i*, int & *len*) [inline, virtual]

Implements typeAcceptor.

Definition at line 113 of file StDbFastSqlWriter.h.

```
113 { *os<<","<<i; };
```

4.24.2.22 void StDbFastSqlWriter::pass (char * *name*, int & *i*, int & *len*) [inline, virtual]

Implements typeAcceptor.

Definition at line 112 of file StDbFastSqlWriter.h.

```
112 { *os<<","<<i; };
```

4.24.2.23 void StDbFastSqlWriter::pass (char * *name*, short & *i*, int & *len*) [inline, virtual]

Implements typeAcceptor.

Definition at line 111 of file StDbFastSqlWriter.h.

```
111 { *os<<","<<i; };
```

4.24.2.24 virtual void StDbFastSqlWriter::streamAccessor () [inline, virtual]

Implements tableAcceptor.

Definition at line 77 of file StDbFastSqlWriter.h.

```
77 {/* no-op */ };
```

4.24.2.25 virtual void StDbFastSqlWriter::streamEndRow () [inline, virtual]

Implements tableAcceptor.

Definition at line 80 of file StDbFastSqlWriter.h.

```
80 {/* no-op */ };
```

4.24.2.26 virtual void StDbFastSqlWriter::streamEndTableName () [inline, virtual]

Implements tableAcceptor.

Definition at line 76 of file StDbFastSqlWriter.h.

```
76 {/* no-op */ };
```

4.24.2.27 virtual void StDbFastSqlWriter::streamHeader (const char * *name*) [inline, virtual]

Implements tableAcceptor.

Definition at line 74 of file StDbFastSqlWriter.h.

```
74 {/* no-op */ };
```

4.24.2.28 virtual void StDbFastSqlWriter::streamRow (int *row*) [inline, virtual]

Implements tableAcceptor.

Definition at line 79 of file StDbFastSqlWriter.h.

```
79 {/* no-op */ };
```

4.24.2.29 virtual void StDbFastSqlWriter::streamTableName (const char * *name*) [inline, virtual]

Implements tableAcceptor.

Definition at line 75 of file StDbFastSqlWriter.h.

```
75 {/* no-op */ };
```

4.24.2.30 virtual void StDbFastSqlWriter::streamTail () [inline, virtual]

Implements tableAcceptor.

Definition at line 81 of file StDbFastSqlWriter.h.

```
81 {/* no-op */ };
```

4.24.3 Member Data Documentation

4.24.3.1 StString* StDbFastSqlWriter::os [protected]

Definition at line 66 of file StDbFastSqlWriter.h.

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

- StDbFastSqlWriter.h
- StDbFastSqlWriter.cc

4.25 StDbLogger Class Reference

```
#include <StDbLogger.hh>
```

Public Member Functions

- StDbLogger ()

  calls to clock

- ~StDbLogger ()
- void start ()
- double end ()
- double getTotalTimes ()
- int getNumCalls ()
- double wallTime ()

Protected Attributes

- double mt0
- double mtotalTimes
- int mnTotal

4.25.1 Constructor & Destructor Documentation

4.25.1.1 StDbLogger::StDbLogger () [inline]

calls to clock

Definition at line 45 of file StDbLogger.hh.

```
45 : mtotalTimes(0), mnTotal(0) {};
```

4.25.1.2 StDbLogger::~StDbLogger () [inline]

Definition at line 46 of file StDbLogger.hh.

```
46 {};
```

4.25.2 Member Function Documentation

4.25.2.1 double StDbLogger::end () [inline]

Definition at line 58 of file StDbLogger.hh.

```
58                                {
59   double elapsedTime=wallTime()-mt0;
60   mtotalTimes+=elapsedTime;
61   mnTotal++;
62   return elapsedTime;
63 }
```

4.25.2.2 int StDbLogger::getNumCalls () [inline]

Definition at line 65 of file StDbLogger.hh.

```
65 { return mnTotal; }
```

4.25.2.3 double StDbLogger::getTotalTimes () [inline]

Definition at line 64 of file StDbLogger.hh.

```
64 { return mtotalTimes; };
```

4.25.2.4 void StDbLogger::start () [inline]

Definition at line 57 of file StDbLogger.hh.

```
57 {mt0=wallTime();}
```

4.25.2.5 double StDbLogger::wallTime () [inline]

Definition at line 66 of file StDbLogger.hh.

```
66                                  {
67     struct timeval Tp;
68     gettimeofday( &Tp, (struct timezone *) 0);
69 // seconds + microseconds/1000000:
70     return Tp.tv_sec + Tp.tv_usec/1000000.0;
71 };
```

4.25.3 Member Data Documentation

4.25.3.1 int StDbLogger::mnTotal [protected]

Definition at line 41 of file StDbLogger.hh.

4.25.3.2 double StDbLogger::mt0 [protected]

Definition at line 39 of file StDbLogger.hh.

4.25.3.3 double StDbLogger::mtotalTimes [protected]

Definition at line 40 of file StDbLogger.hh.

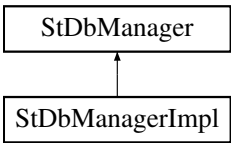
The documentation for this class was generated from the following file:

- StDbLogger.hh

4.26 StDbManager Class Reference

`#include <StDbManager.hh>`

Inheritance diagram for StDbManager::



Public Member Functions

- virtual ~StDbManager ()
- virtual StDbConfigNode * initConfig (const char *dbName)=0
- virtual StDbConfigNode * initConfig (const char *dbName, const char *configName, int opt=0)=0
- virtual StDbConfigNode * initConfig (StDbType type, StDbDomain domain)=0
- virtual StDbConfigNode * initConfig (StDbType type, StDbDomain domain, const char *configName, int opt=0)=0
- virtual StDbConfigNode * initConfig (StDbType type, StDbDomain domain, unsigned int requestTime, int opt=0)=0
- virtual StDbConfigNode * initConfig (const char *dbName, unsigned int requestTime, int opt=0)=0
- virtual char * getConfigNodeName (StDbType t, StDbDomain d)=0
- virtual char * getExternalVersion (StDbType t, StDbDomain d)=0
- virtual dbEnvList * getEnvList (const char *name)=0
- virtual StDbTable * newDbTable (const char *dbName, const char *tName)=0
- virtual StDbTable * newDbTable (StDbNode *node)=0
- virtual StDataBaseI * findDb (StDbType type, StDbDomain domain)=0
- virtual StDataBaseI * findDb (const char *dbType, const char *dbDomain)=0
- virtual StDataBaseI * findDb (const char *databaseName)=0
- virtual StDbServer * findServer (StDbType type, StDbDomain domain)=0
- virtual StDbServer * findServer (StDbNode *node)=0
- virtual StDbServer * findServer (const char *type, const char *domain)=0
- virtual StDbServer * findServer (const char *databaseName)=0
- virtual StDbServer * findDefaultServer ()=0
- virtual char * getDbTypeName (StDbType type)=0
- virtual char * getDbDomainName (StDbDomain domain)=0
- virtual StDbType getDbType (const char *typeName)=0
- virtual StDbDomain getDbDomain (const char *domainName)=0
- virtual char * getDbName (const char *typeName, const char *domName)=0
- virtual char * printDbName (StDbType type, StDbDomain domain)=0
- virtual void setRequestTime (unsigned int time)=0
- virtual void setRequestTime (const char *time)=0
- virtual unsigned int getUnixRequestTime ()=0
- virtual char * getDateRequestTime ()=0
- virtual unsigned int getUnixCheckTime ()=0
- virtual char * getDateCheckTime ()=0
- virtual void setStoreTime (unsigned int time)=0
- virtual void setStoreTime (const char *time)=0

- virtual unsigned int getUnixStoreTime ()=0
- virtual char * getDateStoreTime ()=0
- virtual bool getDataBaseInfo (const char *dbname, char *&type, char *&domain)=0
- virtual bool getDataBaseInfo (const char *dbname, StDbType &type, StDbDomain &domain)=0
- virtual bool IsValid (StDbTable *table)=0
- virtual bool fetchDbTable (StDbTable *table)=0
- virtual bool fetchDbTable (StDbTable *table, char *whereClause)=0
- virtual bool fetchAllTables (StDbConfigNode *node)=0
- virtual bool storeDbTable (StDbTable *table, bool commitWhenDone=true)=0
- virtual bool storeAllTables (StDbConfigNode *node, bool commitWhenDone=true)=0
- virtual int storeConfig (StDbConfigNode *node, int currentID, int &configID, bool commitWhenDone=true)=0
- virtual bool rollBackAllTables (StDbConfigNode *node)=0
- virtual bool rollBackAllNodes (StDbConfigNode *node)=0
- virtual bool rollBack (StDbNode *node)=0
- virtual bool rollBack (StDbTable *table)=0
- virtual bool commitAllTables (StDbConfigNode *node)=0
- virtual bool commitAllNodes (StDbConfigNode *node)=0
- virtual void closeAllConnections ()=0
- virtual void closeAllConnections (StDbConfigNode *node)=0
- virtual void closeConnection (StDbNode *node)=0
- virtual void printTimeStats ()=0
- virtual bool IsVerbose () const
- virtual void setVerbose (bool isVerbose)
- virtual bool IsQuiet () const
- virtual void setQuiet (bool isQuiet)
- virtual void turnOffTimeLogging ()
- virtual void updateMessLevel ()
- virtual void setMessenger (StDbMessService *service)
- virtual StDbMessService * getMessenger ()
- virtual int printInfo (const char *m1, StDbMessLevel ml, int lineNumber=0, const char *class-Name=" ", const char *methodName=" ")
- virtual int printInfo (const char *m1, const char *m2, StDbMessLevel ml, int lineNumber=0, const char *className=" ", const char *methodName=" ")
- virtual char * userName ()
- virtual char * pWord ()
- virtual void setUser (const char *userName, const char *pWord)

Static Public Member Functions

- StDbManager * Instance ()

  strdup(..) is not ANSI

Protected Member Functions

- StDbManager ()

  flag for logging timing

- char * mstringDup (const char *str)

Protected Attributes

- bool misVerbose
- bool misQuiet

  3 levels of verbosity: verbose, normal, quiet.

- StDbMessService * Messenger

  where normal=!misVerbose && !misQuiet

- char * muserName
- char * mpWord
- bool misTimeLogged

Static Protected Attributes

- StDbManager * mInstance = 0

4.26.1 Constructor & Destructor Documentation

4.26.1.1 StDbManager::StDbManager () [protected]

flag for logging timing

Definition at line 157 of file StDbManager.cc.

```
157                         : misVerbose(false), misQuiet(false), muserName(0), mpWord(0), misTimeLogged(t
158    Messenger=new StDbMessenger();
159    updateMessLevel();
160 };
```

4.26.1.2 StDbManager::~StDbManager () [virtual]

Definition at line 161 of file StDbManager.cc.

```
161                          {
162   if(muserName) delete [] muserName;
163   if(mpWord) delete [] mpWord;
164   mInstance=0;
165 }
```

4.26.2 Member Function Documentation

4.26.2.1 virtual void StDbManager::closeAllConnections (StDbConfigNode * *node*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.2 virtual void StDbManager::closeAllConnections () [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.3 virtual void StDbManager::closeConnection (StDbNode * *node*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.4 virtual bool StDbManager::commitAllNodes (StDbConfigNode * *node*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.5 virtual bool StDbManager::commitAllTables (StDbConfigNode * *node*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.6 virtual bool StDbManager::fetchAllTables (StDbConfigNode * *node*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.7 virtual bool StDbManager::fetchDbTable (StDbTable * *table*, char * *whereClause*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.8 virtual bool StDbManager::fetchDbTable (StDbTable * *table*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.9 virtual StDataBaseI* StDbManager::findDb (const char * *databaseName*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.10 virtual StDataBaseI* StDbManager::findDb (const char * *dbType*, const char * *dbDomain*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.11 virtual StDataBaseI* StDbManager::findDb (StDbType *type*, StDbDomain *domain*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.12 virtual StDbServer* StDbManager::findDefaultServer () [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.13 virtual StDbServer* StDbManager::findServer (const char * *databaseName*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.14 virtual StDbServer* StDbManager::findServer (const char * *type*, const char * *domain*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.15 virtual StDbServer* StDbManager::findServer (StDbNode * *node*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.16 virtual StDbServer* StDbManager::findServer (StDbType *type*, StDbDomain *domain*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.17 virtual char* StDbManager::getConfigNodeName (StDbType *t*, StDbDomain *d*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.18 virtual bool StDbManager::getDataBaseInfo (const char * *dbname*, StDbType & *type*, StDbDomain & *domain*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.19 virtual bool StDbManager::getDataBaseInfo (const char * *dbname*, char *& *type*, char *& *domain*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.20 virtual char* StDbManager::getDateCheckTime () [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.21 virtual char* StDbManager::getDateRequestTime () [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.22 virtual char* StDbManager::getDateStoreTime () [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.23 virtual StDbDomain StDbManager::getDbDomain (const char * *domainName*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.24 virtual char* StDbManager::getDbDomainName (StDbDomain *domain*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.25 virtual char* StDbManager::getDbName (const char * *typeName*, const char * *domName*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.26 virtual StDbType StDbManager::getDbType (const char * *typeName*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.27 virtual char* StDbManager::getDbTypeName (StDbType *type*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.28 virtual dbEnvList* StDbManager::getEnvList (const char * *name*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.29 virtual char* StDbManager::getExternalVersion (StDbType *t*, StDbDomain *d*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.30 StDbMessService * StDbManager::getMessenger () [inline, virtual]

Definition at line 301 of file StDbManager.hh.

```
301 { return Messenger; }
```

4.26.2.31 virtual unsigned int StDbManager::getUnixCheckTime () [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.32 virtual unsigned int StDbManager::getUnixRequestTime () [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.33 virtual unsigned int StDbManager::getUnixStoreTime () [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.34 virtual StDbConfigNode* StDbManager::initConfig (const char * *dbName*, unsigned int *requestTime*, int *opt* = 0) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.35 virtual StDbConfigNode* StDbManager::initConfig (StDbType *type*, StDbDomain *domain*, unsigned int *requestTime*, int *opt* = 0) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.36 virtual StDbConfigNode* StDbManager::initConfig (StDbType *type*, StDbDomain *domain*, const char * *configName*, int *opt* = 0) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.37 virtual StDbConfigNode* StDbManager::initConfig (StDbType *type*, StDbDomain *domain*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.38 virtual StDbConfigNode* StDbManager::initConfig (const char * *dbName*, const char * *configName*, int *opt* = 0) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.39 virtual StDbConfigNode* StDbManager::initConfig (const char * *dbName*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.40 StDbManager * StDbManager::Instance () [static]

strdup(..) is not ANSI

Definition at line 152 of file StDbManager.cc.

```
152                                 {
153   if(!mInstance)mInstance=new StDbManagerImpl();
154   return mInstance;
155 }
```

4.26.2.41 bool StDbManager::IsQuiet () const [inline, virtual]

Definition at line 285 of file StDbManager.hh.

```
285 { return misQuiet;};
```

4.26.2.42 virtual bool StDbManager::IsValid (StDbTable * *table*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.43 bool StDbManager::IsVerbose () const [inline, virtual]

Definition at line 279 of file StDbManager.hh.

```
279 { return misVerbose;};
```

4.26.2.44 char * StDbManager::mstringDup (const char * *str*) [protected]

Definition at line 173 of file StDbManager.cc.

```
173                                     {
174
175 char* retString=0;
176 if(!str)return retString;
177 retString = new char[strlen(str)+1];
178 strcpy(retString,str);
179
180 return retString;
181 }
```

4.26.2.45 virtual StDbTable* StDbManager::newDbTable (StDbNode * *node*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.46 virtual StDbTable* StDbManager::newDbTable (const char * *dbName*, const char * *tName*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.47 virtual char* StDbManager::printDbName (StDbType *type*, StDbDomain *domain*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.48 int StDbManager::printInfo (const char * *m1*, const char * *m2*, StDbMessLevel *ml*, int *lineNumber* = 0, const char * *className* = " ", const char * *methodName* = " ") [virtual]

Definition at line 193 of file StDbManager.cc.

```
193
194
195     StString ms;
196     ms<<m1<<" "<<m2;
197     printInfo((ms.str()).c_str(),ml,lineNumber,className,methodName);
198     return 0;
199 }
```

4.26.2.49 int StDbManager::printInfo (const char * *m1*, StDbMessLevel *ml*, int *lineNumber* = 0, const char * *className* = " ", const char * *methodName* = " ") [virtual]

Definition at line 185 of file StDbManager.cc.

```
185
186
187     Messenger->printMessage(m1,ml,lineNumber,className,methodName);
188     return 0;
189 }
```

4.26.2.50 virtual void StDbManager::printTimeStats () [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.51 char * StDbManager::pWord () [inline, virtual]

Definition at line 303 of file StDbManager.hh.

```
303 { return mpWord; }
```

4.26.2.52 virtual bool StDbManager::rollBack (StDbTable * *table*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.53 virtual bool StDbManager::rollBack (StDbNode * *node*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.54 virtual bool StDbManager::rollBackAllNodes (StDbConfigNode * *node*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.55 virtual bool StDbManager::rollBackAllTables (StDbConfigNode * *node*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.56 void StDbManager::setMessenger (StDbMessService * *service*) [inline, virtual]

Definition at line 296 of file StDbManager.hh.

```
296                                                                 {
297   delete Messenger;
298   Messenger=service;
299   updateMessLevel();
300 }
```

4.26.2.57 void StDbManager::setQuiet (bool *isQuiet*) [inline, virtual]

Definition at line 286 of file StDbManager.hh.

```
286                                                 {
287    misQuiet=isQuiet;
288    if(isQuiet) misVerbose=false;
289    updateMessLevel();
290 }
```

4.26.2.58 virtual void StDbManager::setRequestTime (const char * *time*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.59 virtual void StDbManager::setRequestTime (unsigned int *time*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.60 virtual void StDbManager::setStoreTime (const char * *time*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.61 virtual void StDbManager::setStoreTime (unsigned int *time*) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.62 void StDbManager::setUser (const char * *userName*, const char * *pWord*) [virtual]

Definition at line 205 of file StDbManager.cc.

```
205                                                                 {
206
207   if(userName){
208     if(muserName) delete [] muserName;
```

```
    muserName=mstringDup(userName);
  }
  if(pWord){
   if(mpWord) delete [] mpWord;
   mpWord = mstringDup(pWord);
  }
}
```

4.26.2.63 void StDbManager::setVerbose (bool *isVerbose*) [inline, virtual]

Definition at line 280 of file StDbManager.hh.

```
                                                {
 misVerbose=isVerbose;
 if(isVerbose)misQuiet=false;
 updateMessLevel();
};
```

4.26.2.64 virtual bool StDbManager::storeAllTables (StDbConfigNode * *node*, bool *commitWhenDone* = true) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.65 virtual int StDbManager::storeConfig (StDbConfigNode * *node*, int *currentID*, int & *configID*, bool *commitWhenDone* = true) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.66 virtual bool StDbManager::storeDbTable (StDbTable * *table*, bool *commitWhenDone* = true) [pure virtual]

Implemented in StDbManagerImpl.

4.26.2.67 void StDbManager::turnOffTimeLogging () [virtual]

Reimplemented in StDbManagerImpl.

Definition at line 169 of file StDbManager.cc.

```
{ misTimeLogged=false; }
```

4.26.2.68 void StDbManager::updateMessLevel () [inline, virtual]

Definition at line 291 of file StDbManager.hh.

```
                                        {
 Messenger->setMessLevel(dbMWarn);
 if(misQuiet)Messenger->setMessLevel(dbMErr);
 if(misVerbose)Messenger->setMessLevel(dbMDebug);
}
```

4.26.2.69 char * StDbManager::userName () [inline, virtual]

Definition at line 302 of file StDbManager.hh.

```
302 { return muserName; }
```

4.26.3 Member Data Documentation

4.26.3.1 StDbMessService* StDbManager::Messenger [protected]

where normal=!misVerbose && !misQuiet

Definition at line 151 of file StDbManager.hh.

4.26.3.2 StDbManager * StDbManager::mInstance = 0 [static, protected]

Definition at line 150 of file StDbManager.cc.

4.26.3.3 bool StDbManager::misQuiet [protected]

3 levels of verbosity: verbose, normal, quiet.

Definition at line 150 of file StDbManager.hh.

4.26.3.4 bool StDbManager::misTimeLogged [protected]

Definition at line 155 of file StDbManager.hh.

4.26.3.5 bool StDbManager::misVerbose [protected]

Definition at line 149 of file StDbManager.hh.

4.26.3.6 char* StDbManager::mpWord [protected]

Definition at line 153 of file StDbManager.hh.

4.26.3.7 char* StDbManager::muserName [protected]

Definition at line 152 of file StDbManager.hh.

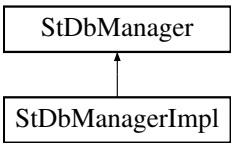
The documentation for this class was generated from the following files:

- StDbManager.hh
- StDbManager.cc

4.27 StDbManagerImpl Class Reference

`#include <StDbManagerImpl.hh>`

Inheritance diagram for StDbManagerImpl::



Public Member Functions

- virtual ~StDbManagerImpl ()
- virtual void turnOffTimeLogging ()
- virtual StDbConfigNode * initConfig (const char *databaseName)
- virtual StDbConfigNode * initConfig (const char *databaseName, const char *configName, int opt=0)
- virtual StDbConfigNode * initConfig (StDbType type, StDbDomain domain)
- virtual StDbConfigNode * initConfig (StDbType type, StDbDomain domain, const char *configName, int opt=0)
- virtual StDbConfigNode * initConfig (StDbType type, StDbDomain domain, unsigned int requestTime, int opt=0)
- virtual StDbConfigNode * initConfig (const char *databaseName, unsigned int requestTime, int opt=0)
- virtual char * getConfigNodeName (StDbType type, StDbDomain domain)
- virtual char * getExternalVersion (StDbType type, StDbDomain domain)
- virtual dbEnvList * getEnvList (const char *name)
- virtual StDbTable * newDbTable (const char *dbName, const char *tabName)
- virtual StDbTable * newDbTable (StDbNode *node)
- virtual StDataBaseI * findDb (StDbType type, StDbDomain domain)
- virtual StDataBaseI * findDb (const char *dbType, const char *dbDomain)
- virtual StDataBaseI * findDb (const char *databaseName)
- virtual StDbServer * findServer (StDbType type, StDbDomain domain)
- virtual StDbServer * findServer (StDbNode *node)
- virtual StDbServer * findServer (const char *dbType, const char *dbDomain)
- virtual StDbServer * findServer (const char *databaseName)
- virtual StDbServer * findDefaultServer ()
- virtual char * getDbTypeName (StDbType type)
- virtual char * getDbDomainName (StDbDomain domain)
- virtual StDbType getDbType (const char *typeName)
- virtual StDbDomain getDbDomain (const char *domainName)
- virtual char * getDbName (const char *typeName, const char *domainName)
- virtual char * printDbName (StDbType type, StDbDomain domain)
- virtual void setRequestTime (unsigned int time)
- virtual void setRequestTime (const char *time)
- virtual unsigned int getUnixRequestTime ()
- virtual char * getDateRequestTime ()
- virtual unsigned int getUnixCheckTime ()

- virtual char * getDateCheckTime ()
- virtual void setStoreTime (unsigned int time)
- virtual void setStoreTime (const char *time)
- virtual unsigned int getUnixStoreTime ()
- virtual char * getDateStoreTime ()
- virtual bool getDataBaseInfo (const char *dbname, char *&type, char *&domain)
- virtual bool getDataBaseInfo (const char *dbname, StDbType &type, StDbDomain &domain)
- virtual bool IsValid (StDbTable *table)
- virtual bool fetchDbTable (StDbTable *table)
- virtual bool fetchDbTable (StDbTable *table, char *whereClause)
- virtual bool fetchAllTables (StDbConfigNode *node)
- virtual bool storeDbTable (StDbTable *table, bool commitWhenDone=true)
- virtual bool storeAllTables (StDbConfigNode *node, bool commitWhenDone=true)
- virtual int storeConfig (StDbConfigNode *node, int currentID, int &configID, bool commitWhenDone=true)
- virtual bool rollBackAllTables (StDbConfigNode *node)
- virtual bool rollBackAllNodes (StDbConfigNode *node)

  for table data

- virtual bool rollBack (StDbNode *node)

  for node data

- virtual bool rollBack (StDbTable *table)

  for node data

- virtual bool commitAllTables (StDbConfigNode *node)

  for table data

- virtual bool commitAllNodes (StDbConfigNode *node)

  table commits

- virtual void closeAllConnections ()

  node commits

- virtual void closeAllConnections (StDbConfigNode *node)
- virtual void closeConnection (StDbNode *node)
- virtual void printTimeStats ()

Public Attributes

- std::vector< std::string > xmlServerList
- short xmlInputSource

Protected Member Functions

- StDbManagerImpl ()
- virtual void initTypes ()
- virtual void initDomains ()
- virtual void addDbType (StDbType type, const char *typeName)

- virtual void addDbDomain (StDbDomain domain, const char *domainName)
- virtual StDbType newDbType (const char *typeName)
- virtual StDbDomain newDbDomain (const char *domainName)
- virtual void deleteServers ()
- virtual void deleteTypes ()
- virtual void deleteDomains ()
- virtual void lookUpServers ()
- virtual void findServersXml (ifstream &is)
- virtual char * findServerString (ifstream &is)
- virtual char * getNextName (char *&name)
- void updateDateTime (StDbTime &t)
- void updateUnixTime (StDbTime &t)

Private Attributes

- dbTypes mTypes

  in order to create "singleton" with inheritance

- dbDomains mDomains

  enum list->mapping shortcut

- StDbType dbTypeFree

  enum list->mapping shortcut

- StDbDomain dbDomainFree

  enum of last unspecified type

- ServerList mservers

  enum of last unspecified domain

- parseXmlString mparser

  servers to handle the Query

- bool mhasServerList

  parses strings in XML

- bool mhasDefaultServer
- StDbTime mcheckTime
- StDbTime mstoreTime
- StDbTableFactory * mfactory
- StDbLogger mnodeLog
- StDbLogger mdataLog

4.27.1 Constructor & Destructor Documentation

4.27.1.1 StDbManagerImpl::StDbManagerImpl () [protected]

Definition at line 220 of file StDbManagerImpl.cc.

```
220                                 : StDbManager(), dbTypeFree(dbTUser1), dbDomainFree(dbDUser1), mhasSer
221
222   initTypes();
223   initDomains();
224   mfactory = new StDbTableFactory();
225 };
```

4.27.1.2 StDbManagerImpl::~StDbManagerImpl () [virtual]

Definition at line 229 of file StDbManagerImpl.cc.

```
229                                 {
230
231   deleteServers();
232   deleteDomains();
233   deleteTypes();
234   delete mfactory;
235   delete Messenger;
236   mInstance=0;
237
238 }
```

4.27.2 Member Function Documentation

4.27.2.1 void StDbManagerImpl::addDbDomain (StDbDomain *domain*, const char * *domainName*) [protected, virtual]

Definition at line 307 of file StDbManagerImpl.cc.

```
307                                                                 {
308   mDomains.push_back(new dbDomain(domain,domainName));
309 }
```

4.27.2.2 void StDbManagerImpl::addDbType (StDbType *type*, const char * *typeName*) [protected, virtual]

Definition at line 300 of file StDbManagerImpl.cc.

```
300                                                             {
301   mTypes.push_back(new dbType(type,typeName));
302 }
```

4.27.2.3 void StDbManagerImpl::closeAllConnections (StDbConfigNode * *node*) [virtual]

Implements StDbManager.

Definition at line 1347 of file StDbManagerImpl.cc.

```
1347                                                    {
1348
1349 if(!node) return;
1350
1351  closeConnection(node);
```

```
1352  if(node->hasChildren())closeAllConnections(node->getFirstChildNode());
1353  StDbConfigNode* next = node->getNextNode();
1354  if(next)closeAllConnections(next);
1355
1356 }
```

4.27.2.4 void StDbManagerImpl::closeAllConnections () [virtual]

node commits

Implements StDbManager.

Definition at line 1338 of file StDbManagerImpl.cc.

```
1338                                    {
1339
1340       for(ServerList::iterator itr = mservers.begin();
1341           itr != mservers.end(); ++itr)
1342           (*itr)->closeConnection();
1343 }
```

4.27.2.5 void StDbManagerImpl::closeConnection (StDbNode * *node*) [virtual]

Implements StDbManager.

Definition at line 1360 of file StDbManagerImpl.cc.

```
1360                                                {
1361
1362 if(!node) return;
1363 StDbServer* server=findServer(node->getDbType(),node->getDbDomain());
1364 if(server && (server->isConnected())) server->closeConnection();
1365
1366 }
```

4.27.2.6 bool StDbManagerImpl::commitAllNodes (StDbConfigNode * *node*) [virtual]

table commits

Implements StDbManager.

Definition at line 1318 of file StDbManagerImpl.cc.

```
1318                                                      {
1319
1320   if(node->hasData()){
1321     StDbTableIter* itr = node->getStDbTableIter();
1322     while(!itr->done()){
1323       StDbNode* ntab= (StDbNode*)itr->next();
1324       ntab->commit();
1325     }
1326     delete itr;
1327   }
1328 if(node->hasChildren())commitAllNodes(node->getFirstChildNode());
1329   StDbConfigNode* nextNode;
1330 if((nextNode=node->getNextNode()))commitAllNodes(nextNode);
1331 node->commit();
1332
1333 return true;
1334 }
```

4.27.2.7 bool StDbManagerImpl::commitAllTables (StDbConfigNode * *node*) [virtual]

for table data

Implements StDbManager.

Definition at line 1293 of file StDbManagerImpl.cc.

```
1293                                                    {
1294
1295 bool retVal=true;
1296
1297   if(node->hasData()){
1298     StDbTableIter* itr = node->getStDbTableIter();
1299     while(!itr->done()){
1300       StDbTable* tab = itr->next();
1301       tab->commitData();
1302     }
1303     delete itr;
1304   }
1305
1306 bool children = true;
1307 bool siblings = true;
1308
1309   if(node->hasChildren())children = commitAllTables(node->getFirstChildNode());
1310   StDbConfigNode* nextNode;
1311   if((nextNode=node->getNextNode()))siblings = commitAllTables(nextNode);
1312
1313 return (retVal && children && siblings);
1314 }
```

4.27.2.8 void StDbManagerImpl::deleteDomains () [protected, virtual]

Definition at line 403 of file StDbManagerImpl.cc.

```
403                                {
404
405   dbDomains::iterator itr;
406   dbDomain* domain;
407
408   do {
409     for(itr = mDomains.begin(); itr!=mDomains.end(); ++itr){
410         domain=*itr;
411         mDomains.erase(itr);
412         delete domain;
413         break;
414     }
415   } while (mDomains.begin() != mDomains.end());
416
417 }
```

4.27.2.9 void StDbManagerImpl::deleteServers () [protected, virtual]

Definition at line 382 of file StDbManagerImpl.cc.

```
382                                {
383
384   if(!mhasServerList)return;
385   ServerList::iterator itr;
386   StDbServer* server;
```

```
387
388   do {
389       for(itr = mservers.begin(); itr != mservers.end(); ++itr){
390          server = *itr;
391          mservers.erase(itr);
392          delete server;
393          break;
394         }
395      } while( mservers.begin() != mservers.end() );
396
397 }
```

4.27.2.10 void StDbManagerImpl::deleteTypes () [protected, virtual]

Definition at line 422 of file StDbManagerImpl.cc.

```
422                              {
423
424   dbTypes::iterator itr;
425   dbType* type;
426
427   do {
428     for(itr = mTypes.begin(); itr!=mTypes.end(); ++itr){
429         type=*itr;
430         mTypes.erase(itr);
431         delete type;
432         break;
433     }
434   } while (mTypes.begin() != mTypes.end());
435
436 }
```

4.27.2.11 bool StDbManagerImpl::fetchAllTables (StDbConfigNode * *node*) [virtual]

Implements StDbManager.

Definition at line 1095 of file StDbManagerImpl.cc.

```
1095                                                        {
1096
1097 bool tables=true;
1098 bool children = true;
1099 bool siblings = true;
1100
1101   if(node->hasData()){
1102     StDbTableIter* itr = node->getStDbTableIter();
1103     while(!itr->done()) {
1104       StDbTable* tab=itr->next();
1105       tables = (tables && fetchDbTable(tab));
1106     }
1107     delete itr;
1108   }
1109
1110   if(node->hasChildren())children = fetchAllTables(node->getFirstChildNode());
1111   StDbConfigNode* nextNode;
1112   if((nextNode=node->getNextNode()))siblings = fetchAllTables(nextNode);
1113
1114 return (tables && children && siblings);
1115 }
```

4.27.2.12 bool StDbManagerImpl::fetchDbTable (StDbTable * *table*, char * *whereClause*) [virtual]

Implements StDbManager.

Definition at line 1083 of file StDbManagerImpl.cc.

```
1083                                                                    {
1084
1085  if(!table) return false;
1086  StDataBaseI* db=findDb(table->getDbType(),table->getDbDomain());
1087  if(!db) return false;
1088  if(misTimeLogged)mdataLog.start();
1089  bool retVal=(bool) db->QueryDb(table,whereClause);
1090  if(misTimeLogged)mdataLog.end();
1091  return retVal;
1092 }
```

4.27.2.13 bool StDbManagerImpl::fetchDbTable (StDbTable * *table*) [virtual]

Implements StDbManager.

Definition at line 1069 of file StDbManagerImpl.cc.

```
1069                                                    {
1070
1071  bool retVal = false;
1072  if(!table) return false;
1073  StDataBaseI* db=findDb(table->getDbType(),table->getDbDomain());
1074  if(!db) return false;
1075
1076  if(misTimeLogged)mdataLog.start();
1077  if(db->QueryDb(table,mcheckTime.munixTime))retVal=true;
1078  if(misTimeLogged)mdataLog.end();
1079
1080  return retVal;
1081 }
```

4.27.2.14 StDataBaseI * StDbManagerImpl::findDb (const char * *databaseName*) [virtual]

Implements StDbManager.

Definition at line 697 of file StDbManagerImpl.cc.

```
697                                                         {
698   return findServer(databaseName)->useDb();
699 }
```

4.27.2.15 StDataBaseI * StDbManagerImpl::findDb (const char * *dbType*, const char * *dbDomain*) [virtual]

Implements StDbManager.

Definition at line 693 of file StDbManagerImpl.cc.

```
693                                                                     {
694   return findServer(dbType,dbDomain)->useDb();
695 }
```

4.27.2.16 StDataBaseI * StDbManagerImpl::findDb (StDbType *type*, StDbDomain *domain*) [virtual]

Implements StDbManager.

Definition at line 689 of file StDbManagerImpl.cc.

```
689                                                                    {
690   return findServer(type,domain)->useDb();
691 }
```

4.27.2.17 StDbServer * StDbManagerImpl::findDefaultServer () [virtual]

Implements StDbManager.

Definition at line 769 of file StDbManagerImpl.cc.

```
769                                                {

770

771  if(!mhasServerList)lookUpServers();
772  StDbServer* server = 0;
773

774 for(ServerList::iterator itr = mservers.begin(); itr != mservers.end(); ++itr){
775    if((*itr)->isDefault()){
776      server = *itr;
777      break;
778    }
779  }
780

781 return server;
782 }
```

4.27.2.18 StDbServer * StDbManagerImpl::findServer (const char * *databaseName*) [virtual]

Implements StDbManager.

Definition at line 739 of file StDbManagerImpl.cc.

```
739                                                                    {

740

741   // if databaseName contains "_" then = 'dbTypeName_dbDomainName'
742   // else = 'dbTypeName' and dbDomainName="Star"
743

744 char* typeName = new char[strlen(databaseName)+1];
745 strcpy(typeName,databaseName);
746 char* domainName;
747

748 char* id=strstr(typeName,"_");
749 if(id){
750    *id='\0';
751     id++;
752     domainName=new char[strlen(id)+1];
753     strcpy(domainName,id);
754 } else {
755     domainName=new char[strlen("Star")+1];
756     strcpy(domainName,"Star");
757 }
758
```

```
759 StDbServer* server=findServer(typeName,(const char*) domainName);
760 delete [] domainName;
761 delete [] typeName;
762
763 return server;
764 }
```

4.27.2.19 StDbServer * StDbManagerImpl::findServer (const char * *dbType*, const char * *dbDomain*) [virtual]

Implements StDbManager.

Definition at line 733 of file StDbManagerImpl.cc.

```
733                                                                                    {
734 return findServer(getDbType(typeName),getDbDomain(domainName));
735 }
```

4.27.2.20 StDbServer * StDbManagerImpl::findServer (StDbNode * *node*) [virtual]

Implements StDbManager.

Definition at line 727 of file StDbManagerImpl.cc.

```
727                                                    {
728 return findServer(node->getDbType(),node->getDbDomain());
729 }
```

4.27.2.21 StDbServer * StDbManagerImpl::findServer (StDbType *type*, StDbDomain *domain*) [virtual]

Implements StDbManager.

Definition at line 701 of file StDbManagerImpl.cc.

```
701                                                                        {
702 #define __METHOD__
703
704  if(!mhasServerList)lookUpServers();
705  StDbServer* server = 0;
706
707  // first check if it exists in list
708  for(ServerList::iterator itr = mservers.begin();
709      itr != mservers.end(); ++itr){
710    if((*itr)->useDb(type,domain)){
711      server = *itr;
712      break;
713    }
714  }
715
716  // if not build from default server
717  if(!server){
718    server=findDefaultServer();
719    server->addDataBase(type,domain);
720  }
721 return server;
722 #undef __METHOD__
723 }
```

4.27.2.22 char * StDbManagerImpl::findServerString (ifstream & *is*) [protected, virtual]

Definition at line 610 of file StDbManagerImpl.cc.

```
610                                                    {
611
612 char tmpline[256];
613 bool done = false;
614 bool started = false;
615 char* id;
616
617  char* line=NULL;
618  StString os;
619
620 while(!done){
621
622    if(is.eof()){
623      done = true;
624      return line;
625    } else {
626      is.getline(tmpline,255);
627
628      if((id=strstr(tmpline,"//")))continue;
629
630      if(!started){
631         id= strstr(tmpline,"<StDbServer>");
632         if(id){
633           os<<tmpline;
634           started = true;
635         }
636      } else {
637         os<<tmpline;
638         id=strstr(tmpline,"</StDbServer>");
639         if(id)done=true;
640
641      } // started check
642    } // eof check
643  } // while loop
644
645  string osstr=os.str();
646  line= new char[osstr.length()+1];
647  strcpy(line,osstr.c_str());
648
649 return line;
650 }
```

4.27.2.23 void StDbManagerImpl::findServersXml (ifstream & *is*) [protected, virtual]

Definition at line 480 of file StDbManagerImpl.cc.

```
480                                                    {
481
482   char* stardatabase=NULL;
483
484   /* Take only the relevant server names from the XML file.
485 What determines the relevance:
486 1) whether the user belongs to the group authorized to use either production or analysis class of serv
487 2) whether the time of the day allows a particular server to be used
488   */
489
490   struct tm *tp;
491   time_t timeNow;
```

```
  timeNow = time(NULL);
  tp = localtime(&timeNow);
  /*
  char* whoami = getenv("USER");
  if (!whoami) whoami = getenv("LOGNAME");
  */
  while(!is.eof()){

  stardatabase = findServerString(is);

  if(!stardatabase) continue;

  // local DTD ...
  char bserver[32]="<server>"; char eserver[32]="</server>";
  char bhost[32]="<host>"; char ehost[32]="</host>";
  char bsock[32]="<socket>"; char esock[32]="</socket>";
  char bport[32]="<port>"; char eport[32]="</port>";
  char bdb[32]="<databases>"; char edb[32]="</databases>";
  //  int portNum = 3306;
  int portNum = 3316;

  char* servName = mparser.getString(stardatabase,(char*)bserver,(char*)eserver);
  char* hostName = mparser.getString(stardatabase,(char*)bhost,(char*)ehost);
  char* uSocket = mparser.getString(stardatabase,(char*)bsock,(char*)esock);
  char* portNumber = mparser.getString(stardatabase,(char*)bport,(char*)eport);

 char* bcos = "<cos>"; char* ecos = "</cos>";
 char* bfrom = "<from_time>"; char* efrom = "</from_time>";
 char* bto = "<to_time>"; char* eto = "</to_time>";
 char* buser = "<user>"; char* euser = "</user>";
 char* production_cos = "production";
 char* analysis_cos = "analysis";

 /*
 char* user = mparser.getString(stardatabase,buser,euser);
 if (user)
   {
     if (strcmp(user,whoami)!=0) continue;
   }
 */

 char* cos = mparser.getString(stardatabase,bcos,ecos);
 char* from_time = mparser.getString(cos,bfrom,efrom);
 char* to_time = mparser.getString(cos,bto,eto);

 if (from_time==NULL) from_time = "0";
 if (to_time==NULL) to_time = "0";

 int from_time_i = atoi(from_time);
 int to_time_i = atoi(to_time);

 bool unknown_cos = true;


 if (cos) // support pre-"cos" XML format
   {
     if (xmlInputSource == userHome)
       {
         if (!strstr(cos,analysis_cos)) continue;
       }
     else
       {
         if (!strstr(cos,production_cos)) continue;
       }
   }


```

```
559  if (from_time_i!=0 || to_time_i!=0)
560    {
561      if (tp->tm_hour < from_time_i && tp->tm_hour > to_time_i)
562        {
563          continue;
564        }
565    }
566
567
568  xmlServerList.push_back(hostName);
569
570   if(portNumber)portNum = atoi(portNumber);
571
572   StDbServerImpl* myserver = new StDbServerImpl(servName,hostName,uSocket,portNum);
573   myserver->PointMysqlDb(this);
574   StDbServer* server = myserver;
575
576   if(muserName) server->setUser(muserName,mpWord);
577   delete [] servName;
578   delete [] hostName;
579   delete [] uSocket;
580   if(portNumber)delete [] portNumber;
581   mservers.push_back(server);
582   server->setTimeLogging(misTimeLogged);
583
584   char* dbNames = mparser.getString(stardatabase,(char*)bdb,(char*)edb);
585   if( !dbNames && !mhasDefaultServer ){
586
587     server->setIsDefaultServer();
588     mhasDefaultServer = true;
589
590   } else {
591
592     //    char* p1 = &dbNames[0];
593     char* p1 = dbNames;
594     char* aname;
595     StDbType type; StDbDomain domain;
596     while(p1 && (aname=getNextName(p1))){
597       if(getDataBaseInfo(aname,type,domain))server->addDataBase(type,domain);
598       delete [] aname;
599     }
600    if(dbNames)delete [] dbNames;
601   }
602     if(stardatabase){
603       delete [] stardatabase;
604       stardatabase=0;
605     }
606   }
607 }
```

4.27.2.24 char * StDbManagerImpl::getConfigNodeName (StDbType *type*, StDbDomain *domain*) [virtual]

Implements StDbManager.

Definition at line 949 of file StDbManagerImpl.cc.

```
949                                                                    {
950   // config node name is based on type IF domain is "dbStar",
951   // else it is based on the domain
952  if(domain == dbStar) return getDbTypeName(type);
953  return getDbDomainName(domain);
954 }
```

4.27.2.25 bool StDbManagerImpl::getDataBaseInfo (const char * *dbname*, StDbType & *type*, StDbDomain & *domain*) [virtual]

Implements StDbManager.

Definition at line 1452 of file StDbManagerImpl.cc.

```
1452                                                                                    {
1453   char* typeName=0; char* domainName=0;
1454   if(!dbName || !(getDataBaseInfo(dbName,typeName,domainName)))return false;
1455   type=getDbType(typeName);
1456   domain=getDbDomain(domainName);
1457   if(typeName)delete [] typeName;
1458   if(domainName) delete [] domainName;
1459 return true;
1460 }
```

4.27.2.26 bool StDbManagerImpl::getDataBaseInfo (const char * *dbname*, char *& *type*, char *& *domain*) [virtual]

Implements StDbManager.

Definition at line 1426 of file StDbManagerImpl.cc.

```
1426                                                                                    {
1427 #define __METHOD__ "getDataBaseInfo(dbName,type,domain)"
1428 char* tmpName=mstringDup(dbName);
1429 if(!tmpName) return false;
1430 char* id;
1431  if( !(id=strstr(tmpName,"_"))){
1432     type=tmpName;
1433     domain = mstringDup("Star");
1434  } else {
1435     *id='\0'; id++;
1436     type=mstringDup(tmpName);
1437     domain=mstringDup(id);
1438     id--; *id='_';
1439     delete [] tmpName;
1440  }
1441  StString ni;
1442  ni<<" Found dbType="<<type<<" & dbDomain="<<domain;
1443  ni<<" from DataBase Name="<<dbName;
1444
1445  printInfo((ni.str()).c_str(),dbMDebug,__LINE__,__CLASS__,__METHOD__);
1446
1447 return true;
1448 #undef __METHOD__
1449 }
```

4.27.2.27 char * StDbManagerImpl::getDateCheckTime () [virtual]

Implements StDbManager.

Definition at line 1051 of file StDbManagerImpl.cc.

```
1051 { return mcheckTime.mdateTime; }
```

4.27.2.28 char * StDbManagerImpl::getDateRequestTime () [virtual]

Implements StDbManager.

Definition at line 1052 of file StDbManagerImpl.cc.

```
1052 { return mcheckTime.mdateTime; }
```

4.27.2.29 char * StDbManagerImpl::getDateStoreTime () [virtual]

Implements StDbManager.

Definition at line 1056 of file StDbManagerImpl.cc.

```
1056 { return mstoreTime.mdateTime; }
```

4.27.2.30 StDbDomain StDbManagerImpl::getDbDomain (const char * *domainName*) [virtual]

Implements StDbManager.

Definition at line 853 of file StDbManagerImpl.cc.

```
853                                                                 {
854 #define __METHOD__ "getDbDomain(domainName)"
855
856   StDbDomain retType=dbDomainUnknown;
857   bool found=false;
858   for(dbDomains::iterator itr=mDomains.begin();
859       itr != mDomains.end(); ++itr){
860     if(strcmp((*itr)->name,domainName) ==0){
861       retType = (*itr)->domain;
862       found=true;
863       break;
864     }
865   }
866
867   if(found)return retType;
868   printInfo("Adding New dbDomain=",domainName,dbMWarn,__LINE__,__CLASS__,__METHOD__);
869 return newDbDomain(domainName);
870 #undef __METHOD__
871 }
```

4.27.2.31 char * StDbManagerImpl::getDbDomainName (StDbDomain *domain*) [virtual]

Implements StDbManager.

Definition at line 807 of file StDbManagerImpl.cc.

```
807                                                         {
808 #define __METHOD__ "getDbDomainName(StDbDomain)"
809
810 if(domain==dbDomainUnknown)return mstringDup("Star");
811
812 char* name=0;
813   for(dbDomains::iterator itr=mDomains.begin();
```

```
      itr != mDomains.end(); ++itr){
    if((*itr)->domain == domain){
      name = (*itr)->name;
      break;
    }
  }

  if(name) return mstringDup(name);
  printInfo(" Domain Name not found",dbMErr,__LINE__,__CLASS__,__METHOD__);
  return name;
#undef __METHOD__
}
```

4.27.2.32 char * StDbManagerImpl::getDbName (const char * *typeName*, const char * *domainName*) [virtual]

Implements StDbManager.

Definition at line 1464 of file StDbManagerImpl.cc.

```
1464                                                                    {
1465
1466   StString dbname;
1467   dbname<<typeName;
1468   if(strcmp(domainName,"Star")!=0)dbname<<"_"<<domainName;
1469   char* retName = mstringDup((dbname.str()).c_str());
1470
1471   return retName;
1472 }
```

4.27.2.33 StDbType StDbManagerImpl::getDbType (const char * *typeName*) [virtual]

Implements StDbManager.

Definition at line 830 of file StDbManagerImpl.cc.

```
830                                                     {
831 #define __METHOD__ "getDbType(typeName)"
832   StDbType retType=dbStDb;
833   bool found=false;
834   for(dbTypes::iterator itr=mTypes.begin();
835       itr != mTypes.end(); ++itr){
836     if(strcmp((*itr)->name,typeName)==0){
837       retType = (*itr)->type;
838       found = true;
839       break;
840     }
841   }
842
843   if(found)return retType;
844   printInfo("Adding New User dbType=",typeName,dbMDebug,__LINE__,__CLASS__,__METHOD__);
845
846  return newDbType(typeName); //This'll overwrite User db definition
847 #undef __METHOD__
848 }
```

4.27.2.34 char * StDbManagerImpl::getDbTypeName (StDbType *type*) [virtual]

Implements StDbManager.

Definition at line 786 of file StDbManagerImpl.cc.

```
786                                                 {
787 #define __METHOD__ "getDbTypeName(StDbType)"
788
789 char* name=0;
790   for(dbTypes::iterator itr=mTypes.begin();
791       itr != mTypes.end(); ++itr){
792     if((*itr)->type == type){
793       name = (*itr)->name;
794       break;
795     }
796   }
797
798   if(name) return mstringDup(name);
799   printInfo(" No dbType name found ",dbMErr,__LINE__,__CLASS__,__METHOD__);
800   return name;
801 #undef __METHOD__
802 }
```

4.27.2.35 dbEnvList * StDbManagerImpl::getEnvList (const char * *name*) [virtual]

Implements StDbManager.

Definition at line 962 of file StDbManagerImpl.cc.

```
962                                                         {
963
964   dbEnvList* retVal=new dbEnvList;
965
966   for(int i=0;;i++){
967     if(!environ[i])break;
968     if(strstr(environ[i],name)){
969       char* tmpName=mstringDup(environ[i]);
970       char* id=strstr(tmpName,"=");
971       if(id){
972         *id='\0';id++;
973         if(strstr(tmpName,name)){
974            retVal->envVar[retVal->num]=mstringDup(tmpName);
975            retVal->envDef[retVal->num]=mstringDup(id);
976            retVal->num++;
977         } // check of name in env-var side
978         id--;*id='=';
979       } // check of "=" in env-var
980       delete [] tmpName;
981     } // check of name in env-var
982   } // loop over all env-vars
983
984   if(retVal->num==0){
985     delete retVal;
986     retVal=0;
987   }
988
989   return retVal;
990 }
```

4.27.2.36 char * StDbManagerImpl::getExternalVersion (StDbType *type*, StDbDomain *domain*) [virtual]

Implements StDbManager.

Definition at line 957 of file StDbManagerImpl.cc.

```
957                                                                    {
958   return getenv(printDbName(type,domain));
959 }
```

4.27.2.37 char * StDbManagerImpl::getNextName (char *& *name*) [protected, virtual]

Definition at line 653 of file StDbManagerImpl.cc.

```
653                                                {

654

655 char* nextName = 0;
656 if(!names)return nextName;
657

658 char* id = strstr(names,",");
659

660 if(!id) {
661  nextName = mparser.removeBlankEnds(names);
662  names = 0;
663 } else {
664   int iloc = id-names;
665   char* saveName = new char[iloc+1];
666   strncpy(saveName,names,iloc);
667   saveName[iloc]='\0';
668   nextName=mparser.removeBlankEnds(saveName);
669   delete [] saveName;
670   names = id; names++;
671 }

672

673 return nextName;
674 }
```

4.27.2.38 unsigned int StDbManagerImpl::getUnixCheckTime () [virtual]

Implements StDbManager.

Definition at line 1049 of file StDbManagerImpl.cc.

```
1049 { return mcheckTime.munixTime;}
```

4.27.2.39 unsigned int StDbManagerImpl::getUnixRequestTime () [virtual]

Implements StDbManager.

Definition at line 1050 of file StDbManagerImpl.cc.

```
1050 {return mcheckTime.munixTime;}
```

4.27.2.40 unsigned int StDbManagerImpl::getUnixStoreTime () [virtual]

Implements StDbManager.

Definition at line 1055 of file StDbManagerImpl.cc.

```
1055 { return mstoreTime.munixTime; }
```

4.27.2.41 StDbConfigNode * StDbManagerImpl::initConfig (const char * *databaseName*, unsigned int *requestTime*, int *opt* = 0) [virtual]

Implements StDbManager.

Definition at line 942 of file StDbManagerImpl.cc.

```
942                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                {
943  StDbType type;  StDbDomain domain;
944  getDataBaseInfo(dbName,type,domain);  // fill type & domain strings via dbName
945 return initConfig(type,domain,requestTime,opt);
946 }
```

4.27.2.42 StDbConfigNode * StDbManagerImpl::initConfig (StDbType *type*, StDbDomain *domain*, unsigned int *requestTime*, int *opt* = 0) [virtual]

Implements StDbManager.

Definition at line 919 of file StDbManagerImpl.cc.

```
919
920 #define __METHOD__ "initConfig(type,domain,time)"
921
922  StDbConfigNode* configNode=0;
923  StDbTable* table = new StDbTable("dbCollection");
924  StDataBaseI* db = findServer(type,domain)->useDb();
925  if(!db){
926    printInfo("DataBase Not Found",dbMErr,__LINE__,__CLASS__,__METHOD__);
927    return configNode;
928  }
929
930  if(!db->QueryDb(table,requestTime)){
931    printInfo("Timestamped collection Not found",dbMErr,__LINE__,__CLASS__,__METHOD__);
932    return configNode;
933  }
934  dbCollection* collection = (dbCollection*)table->GetTable();
935  configNode=initConfig(type,domain,collection->name,opt);
936  delete table;
937  return configNode;
938 #undef __METHOD__
939 }
```

4.27.2.43 StDbConfigNode * StDbManagerImpl::initConfig (StDbType *type*, StDbDomain *domain*, const char * *configName*, int *opt* = 0) [virtual]

Implements StDbManager.

Definition at line 907 of file StDbManagerImpl.cc.

```
907
908
909   if(misTimeLogged)mnodeLog.start();
910   StDbConfigNode* configNode=initConfig(type,domain);
911   configNode->setVersion(configName);
912   configNode->buildTree(opt);
913   if(misTimeLogged)mnodeLog.end();
914
915  return configNode;
916 }
```

4.27.2.44 StDbConfigNode * StDbManagerImpl::initConfig (StDbType *type*, StDbDomain *domain*) [virtual]

Implements StDbManager.

Definition at line 897 of file StDbManagerImpl.cc.

```
897                                                                    {
898   // create Config node with appropriate name for db type & domain
899   char* name = getConfigNodeName(type,domain);
900   StDbConfigNode* configNode = new StDbConfigNodeImpl(type,domain,name);
901   configNode->setIsDbNode(true);
902   delete [] name;
903  return configNode;
904 }
```

4.27.2.45 StDbConfigNode * StDbManagerImpl::initConfig (const char * *databaseName*, const char * *configName*, int *opt* = 0) [virtual]

Implements StDbManager.

Definition at line 888 of file StDbManagerImpl.cc.

```
888                                                                                                    {
889 #define __METHOD__ "initConfig(dbName,configName,opt)"
890  StDbType type; StDbDomain domain;
891  getDataBaseInfo(dbName,type,domain);
892 return initConfig(type,domain,configName,opt);
893 #undef __METHOD__
894 }
```

4.27.2.46 StDbConfigNode * StDbManagerImpl::initConfig (const char * *databaseName*) [virtual]

Implements StDbManager.

Definition at line 881 of file StDbManagerImpl.cc.

```
881                                                        {
882  StDbType type; StDbDomain domain;
883  getDataBaseInfo(dbName,type,domain);
884  return initConfig(type,domain);
885 }
```

4.27.2.47 void StDbManagerImpl::initDomains () [protected, virtual]

Definition at line 268 of file StDbManagerImpl.cc.

```
268                             {
269
270 addDbDomain(dbStar,"Star");
271 addDbDomain(dbTpc,"tpc");
272 addDbDomain(dbFtpc,"ftpc");
273 addDbDomain(dbEmc,"emc");
274 addDbDomain(dbSvt,"svt");
```

```
addDbDomain(dbCtb,"ctb");
addDbDomain(dbTrg,"trg");
addDbDomain(dbDaq,"daq");
addDbDomain(dbScaler,"scaler");
addDbDomain(dbGlobal,"global");
addDbDomain(dbL3,"l3");
addDbDomain(dbOnl,"onl");
addDbDomain(dbRich,"rich");
addDbDomain(dbMwc,"mwc");
addDbDomain(dbRhic,"rhic");
addDbDomain(dbSsd,"ssd");
addDbDomain(dbRts,"rts");
addDbDomain(dbTof,"tof");
addDbDomain(dbFpd,"fpd");
addDbDomain(dbEemc,"eemc");
addDbDomain(dbPmd,"pmd");
addDbDomain(dbBbc,"bbc");
addDbDomain(dbTracker,"tracker");
addDbDomain(dbZdc,"zdc");

}
```

4.27.2.48 void StDbManagerImpl::initTypes () [protected, virtual]

Definition at line 249 of file StDbManagerImpl.cc.

```
249                          {

addDbType(dbStDb,"StarDb");
addDbType(dbRunLog,"RunLog");
addDbType(dbServer," ");
addDbType(dbConditions,"Conditions");
addDbType(dbCalibrations,"Calibrations");
addDbType(dbGeometry,"Geometry");
addDbType(dbRunCatalog,"RunCatalog");
addDbType(dbConfigurations,"Configurations");
addDbType(dbRunParams,"RunParams");
addDbType(dbScalers,"Scalers");
addDbType(dbTestScheme,"TestScheme");

}
```

4.27.2.49 bool StDbManagerImpl::IsValid (StDbTable * *table*) [virtual]

Implements StDbManager.

Definition at line 1059 of file StDbManagerImpl.cc.

```
1059                                                {
1060  bool retVal = false;
1061  if(table) {
1062   unsigned int time = mcheckTime.munixTime;
1063   if(time >= table->getBeginTime() && time < table->getEndTime())retVal=true;
1064  }
1065  return retVal;
1066 }
```

4.27.2.50 void StDbManagerImpl::lookUpServers () [protected, virtual]

Definition at line 439 of file StDbManagerImpl.cc.

```
439                                   {
440 #define __METHOD__ "lookUpServer()"
441
442  char* xmlFile[3]={NULL,NULL,NULL};
443 // dbFindServerMode mode[3]={userHome,serverEnvVar,starDefault};
444  dbFindServerMode mode[3]={serverEnvVar,userHome,starDefault};
445
446
447  StString cos;
448  cos<<stendl<<"******** Order of Files searched for dbServers ********* "<<stendl;
449
450  for(int i=0;i<3; i++){
451    xmlFile[i]=StDbDefaults::Instance()->getServerFileName(mode[i]);
452
453    if(xmlFile[i]){
454      ifstream is(xmlFile[i]);
455      if(is){
456
457        cout << " looking at "<<xmlFile[i]<<"\n";
458        cos<<"  "<<i+1<<". "<< xmlFile[i] <<stendl;
459        findServersXml(is);
460        is.close();
461
462        xmlInputSource = mode[i];
463        break;
464      }
465      delete [] xmlFile[i];
466      xmlFile[i]=NULL;
467    }
468  }
469
470  cos <<"*******************************************************" << stendl;
471
472  printInfo((cos.str()).c_str(),dbMConnect,__LINE__,__CLASS__,__METHOD__);
473
474  mhasServerList = true;
475
476 #undef __METHOD__
477 }
```

4.27.2.51 StDbDomain StDbManagerImpl::newDbDomain (const char * *domainName*) [protected, virtual]

Definition at line 348 of file StDbManagerImpl.cc.

```
348                                                   {
349
350   StDbDomain retVal=dbDomainFree;
351   switch(dbDomainFree) {
352     case dbDUser1:
353       {
354        addDbDomain(dbDUser1,domainName);
355        dbDomainFree = dbDUser2;
356        break;
357       }
358     case dbDUser2:
359       {
360        addDbDomain(dbDUser2,domainName);
361        dbDomainFree = dbDUser3;
```

```
        break;
      }
    case dbDUser3:
      {
        addDbDomain(dbDUser3,domainName);
        dbDomainFree = dbDEnd;
        break;
      }
    default:
      {
        break;
      }
  }

  return retVal;
}
```

4.27.2.52 StDbTable * StDbManagerImpl::newDbTable (StDbNode * *node*) [virtual]

Implements StDbManager.

Definition at line 998 of file StDbManagerImpl.cc.

```
998                                                    {
999   StDbTable* table=mfactory->newDbTable(node->printDbName(),node->printName());
1000   if(table)table->setNodeInfo(node);
1001   return table;
1002 }
```

4.27.2.53 StDbTable * StDbManagerImpl::newDbTable (const char * *dbName*, const char * *tabName*) [virtual]

Implements StDbManager.

Definition at line 993 of file StDbManagerImpl.cc.

```
993                                                                                {
994   return mfactory->newDbTable(dbName,tabName);
995 }
```

4.27.2.54 StDbType StDbManagerImpl::newDbType (const char * *typeName*) [protected, virtual]

Definition at line 314 of file StDbManagerImpl.cc.

```
314                                                {

316   StDbType retVal=dbTypeFree;
317   switch(dbTypeFree) {
318     case dbTUser1:
319       {
320        addDbType(dbTUser1,typeName);
321        dbTypeFree = dbTUser2;
322        break;
323       }
324     case dbTUser2:
325       {
```

```
326         addDbType(dbTUser2,typeName);
327         dbTypeFree = dbTUser3;
328         break;
329        }
330      case dbTUser3:
331        {
332         addDbType(dbTUser3,typeName);
333         dbTypeFree = dbTEnd;
334         break;
335        }
336      default:
337        {
338         break;
339        }
340    }
341
342    return retVal;
343  }
```

4.27.2.55 char * StDbManagerImpl::printDbName (StDbType *type*, StDbDomain *domain*) [virtual]

Implements StDbManager.

Definition at line 874 of file StDbManagerImpl.cc.

```
874                                                                    {
875    StDataBaseI* db=findServer(type,domain)->useDb();
876    if(db) return db->printDbName();
877    return (char*)"Unknown DataBase";
878  }
```

4.27.2.56 void StDbManagerImpl::printTimeStats () [virtual]

Implements StDbManager.

Definition at line 1370 of file StDbManagerImpl.cc.

```
1370                                {
1371  #define __METHOD__ "printTimeStats()"
1372
1373  if(!misTimeLogged){ cout<< "Timing Not Logged"<<stendl; return; }
1374  double queryTimes=0;
1375  double socketTimes=0;
1376   double connectTimes[10];
1377   char*  serverID[10];
1378   int j=0;
1379  for(ServerList::iterator itr = mservers.begin();
1380      itr != mservers.end(); ++itr){
1381        queryTimes+=(*itr)->getQueryTimes();
1382        socketTimes+=(*itr)->getSocketTimes();
1383        connectTimes[j]=(*itr)->getConnectTimes();
1384        serverID[j]=(*itr)->printServerName();
1385        j++;
1386  }
1387   double dbTotalConnect=0.;
1388   int i;
1389   for(i=0;i<j;i++)dbTotalConnect+=connectTimes[i];
1390
1391   double dbTotalTimes=mnodeLog.getTotalTimes() + mdataLog.getTotalTimes();
1392   double dbNodeTotal=mnodeLog.getTotalTimes();// - dbTotalConnect;
```

```
1393  double dbNF=100.0*(dbNodeTotal/dbTotalTimes);
1394  double dbDF=100.0*(mdataLog.getTotalTimes()/dbTotalTimes);
1395  double dbQF=100.0*(queryTimes/dbTotalTimes);
1396  double dbSF=100.0*(socketTimes/dbTotalTimes);
1397  // double dbTF=100.0*((socketTimes+queryTimes)/dbTotalTimes);
1398  double dbCF=100.0*(dbTotalConnect/dbTotalTimes);
1399
1400  StString cos;
1401  cos<<stendl<<"************************* DataBase ClockTime Stats ************************* "<<st
1402
1403  cos<<"Total Time in DB-API       = "<<dbTotalTimes<<"\t sec"<<stendl;
1404  cos<<" --------------------- In aggregate ------------------"<<stendl;
1405  cos<<" - Prepare Nodes & Servers = "<<dbNodeTotal<<"\t sec --> "<<dbNF<<"% of total"<<stendl;
1406  cos<<" - Select & Retrieve data  = "<<mdataLog.getTotalTimes()<<"\t sec --> "<<dbDF<<"% of total"<<s
1407  cos<<" --------------- In MySQL C-API (approximate) ---------"<<stendl;
1408  cos<<" - Connecting to Servers   = "<<dbTotalConnect<<"\t sec --> "<<dbCF<<"% of total"<<stendl;
1409  if(j>1){
1410   cos<<" [ Connections per Server; ";
1411   for(i=0;i<j;i++)cos<<serverID[i]<<"="<<connectTimes[i]<<" sec ";
1412   cos<<" ] "<<stendl;
1413   }
1414  cos<<" - SQL Query Times         = "<<queryTimes<<"\t sec --> "<<dbQF<<"% of total"<<stendl;
1415  cos<<" - socket transfer Times   = "<<socketTimes<<"\t sec --> "<<dbSF<<"% of total"<<stendl;
1416
1417  cos<<"******************************************************************************"<<stendl;
1418
1419  printInfo((cos.str()).c_str(),dbMConnect,__LINE__,__CLASS__,__METHOD__);
1420
1421 #undef __METHOD__
1422 }
```

4.27.2.57 bool StDbManagerImpl::rollBack (StDbTable * *table*) [virtual]

for node data

Implements StDbManager.

Definition at line 1281 of file StDbManagerImpl.cc.

```
1281                                          {
1282
1283   int numRows;
1284   int* dataRows=table->getWrittenRows(numRows);
1285   if(!dataRows)return true;
1286   StDataBaseI* db=findDb(table->getDbType(),table->getDbDomain());
1287   if(!db) return false;
1288   return db->rollBack(table);
1289 }
```

4.27.2.58 bool StDbManagerImpl::rollBack (StDbNode * *node*) [virtual]

for node data

Implements StDbManager.

Definition at line 1267 of file StDbManagerImpl.cc.

```
1267                                        {
1268 #define __METHOD__ "rollBack(node)"
1269
1270  if(!node->canRollBack())
```

```
    return (bool)printInfo(" Cannot rollback Store of Node ",dbMErr,__LINE__,__CLASS__,__METHOD__);

 StDataBaseI* db=findDb(node->getDbType(),node->getDbDomain());
 if(!db) return false;
 return db->rollBack(node);
#undef __METHOD__
}
```

4.27.2.59 bool StDbManagerImpl::rollBackAllNodes (StDbConfigNode * *node*) [virtual]

for table data

Implements StDbManager.

Definition at line 1240 of file StDbManagerImpl.cc.

```
1240                                                    {
1241
1242   bool tables, children, siblings, thisNode;
1243   tables=children=siblings=true;
1244
1245   if(node->hasData()){
1246     StDbTableIter* itr = node->getStDbTableIter();
1247     while(!itr->done()){
1248       StDbTable* tab = itr->next();
1249       tables=(tables && rollBack((StDbNode*)tab));
1250     }
1251     delete itr;
1252   }
1253
1254 if(node->hasChildren())
1255    children=rollBackAllNodes(node->getFirstChildNode());
1256   StDbConfigNode* nextNode;
1257 if((nextNode=node->getNextNode()))
1258    siblings=rollBackAllNodes(nextNode);
1259
1260  thisNode=rollBack(node);
1261
1262 return (tables && children && siblings && thisNode);
1263 }
```

4.27.2.60 bool StDbManagerImpl::rollBackAllTables (StDbConfigNode * *node*) [virtual]

Implements StDbManager.

Definition at line 1215 of file StDbManagerImpl.cc.

```
1215                                                    {
1216
1217   bool tables, children, siblings;
1218   tables=children=siblings=true;
1219
1220   if(node->hasData()){
1221     StDbTableIter* itr = node->getStDbTableIter();
1222     while(!itr->done()){
1223      StDbTable* table=itr->next();
1224      if(table && table->canRollBack())tables = (tables && rollBack(table));
1225     }
1226     delete itr;
1227   }
1228
```

```
1229   if(node->hasChildren())
1230       children = rollBackAllTables(node->getFirstChildNode());
1231   StDbConfigNode* nextNode;
1232   if((nextNode=node->getNextNode()))
1233       siblings = rollBackAllTables(nextNode);
1234
1235 return (tables && children && siblings);
1236 }
```

4.27.2.61 void StDbManagerImpl::setRequestTime (const char * *time*) [virtual]

Implements StDbManager.

Definition at line 1011 of file StDbManagerImpl.cc.

```
1011                                                    {
1012  mcheckTime.setDateTime(time);
1013  updateUnixTime(mcheckTime);
1014 }
```

4.27.2.62 void StDbManagerImpl::setRequestTime (unsigned int *time*) [virtual]

Implements StDbManager.

Definition at line 1005 of file StDbManagerImpl.cc.

```
1005                                                    {
1006  mcheckTime.munixTime = time;
1007  updateDateTime(mcheckTime);
1008 }
```

4.27.2.63 void StDbManagerImpl::setStoreTime (const char * *time*) [virtual]

Implements StDbManager.

Definition at line 1023 of file StDbManagerImpl.cc.

```
1023                                                    {
1024  mstoreTime.setDateTime(time);
1025  updateUnixTime(mstoreTime);
1026 }
```

4.27.2.64 void StDbManagerImpl::setStoreTime (unsigned int *time*) [virtual]

Implements StDbManager.

Definition at line 1017 of file StDbManagerImpl.cc.

```
1017                                                    {
1018  mstoreTime.munixTime = time;
1019  updateDateTime(mstoreTime);
1020 }
```

4.27.2.65 bool StDbManagerImpl::storeAllTables (StDbConfigNode * *node*, bool *commitWhenDone* = true) [virtual]

Implements StDbManager.

Definition at line 1134 of file StDbManagerImpl.cc.

```
1134                                                                                    {
1135 #define __METHOD__ "storeAllTable(StDbConfigNode*)"
1136
1137 if(!node)  return (bool) printInfo("Cannot store Null Node",dbMErr,__LINE__,__CLASS__,__METHOD__);
1138 if(mstoreTime.munixTime==0) return (bool) printInfo("Cannot store withno timestamp",dbMErr,__LINE__,_
1139
1140   bool tables=true;
1141   if(node->hasData()){
1142     StDbTableIter* itr = node->getStDbTableIter();
1143     while(!itr->done()){
1144       if(tables){
1145         tables = (tables && storeDbTable(itr->next(), false));
1146       } else {
1147         (itr->next())->commitData(); // prevent rollback of non-stored tables
1148       }
1149     }
1150   delete itr;
1151   }
1152
1153   // end it here if store fails
1154 if(!tables) {
1155   rollBackAllTables(node);
1156   return tables;
1157 }
1158
1159 // do children & siblings
1160 bool children = true;
1161 bool siblings = true;
1162
1163   if(node->hasChildren()) children = storeAllTables(node->getFirstChildNode(),false);
1164
1165   if(!children){
1166     rollBackAllTables(node);
1167     return false;
1168   }
1169
1170   StDbConfigNode* nextNode;
1171   if((nextNode=node->getNextNode())) siblings = storeAllTables(nextNode,false);
1172
1173   if(!siblings){
1174     rollBackAllTables(node);
1175     return false;
1176   }
1177   if(commitWhenDone)commitAllTables(node);
1178 return true;
1179 #undef __METHOD__
1180 }
```

4.27.2.66 int StDbManagerImpl::storeConfig (StDbConfigNode * *node*, int *currentID*, int & *configID*, bool *commitWhenDone* = true) [virtual]

Implements StDbManager.

Definition at line 1184 of file StDbManagerImpl.cc.

```
1184                                                                                    {
```

```
1185 #define __METHOD__ "storeConfig(node,ID,configID,commitFlag)"
1186
1187  StDataBaseI* db = findDb(node->getDbType(),node->getDbDomain());
1188  if(!db) return 0;
1189  int nodeID, childID, sibID;
1190
1191 if(!(nodeID=db->WriteDb(node,currentID,configID))) return printInfo(" Write Failed ",dbMErr,__LINE__,
1192
1193  node->addWrittenNode(nodeID);
1194
1195  if(node->hasChildren())
1196    if(!(childID=storeConfig(node->getFirstChildNode(),nodeID,configID,false))){
1197         rollBack(node);
1198         return printInfo(" Write Failed in Child Node",dbMErr,__LINE__,__CLASS__,__METHOD__);
1199    }
1200
1201  if(node->getNextNode())
1202    if(!(sibID=storeConfig(node->getNextNode(),currentID,configID,false))) {
1203         rollBackAllNodes(node->getFirstChildNode());
1204         rollBack(node);
1205         return printInfo(" Write Failed in Sibling Node",dbMErr,__LINE__,__CLASS__,__METHOD__);
1206    }
1207
1208 if(commitWhenDone) commitAllNodes(node);
1209 return nodeID;
1210 #undef __METHOD__
1211 }
```

4.27.2.67 bool StDbManagerImpl::storeDbTable (StDbTable * *table*, bool *commitWhenDone* = true) [virtual]

Implements StDbManager.

Definition at line 1118 of file StDbManagerImpl.cc.

```
1118                                                                    {
1119 #define __METHOD__ "storeDbTable(StDbTable*)"
1120
1121 if(!table) return (bool) printInfo("Cannot store Null Table",dbMErr,__LINE__,__CLASS__,__METHOD__);
1122 if(mstoreTime.munixTime==0) return (bool) printInfo("Cannot store withou timestamp",dbMErr,__LINE__,_
1123
1124   StDataBaseI* db = findDb(table->getDbType(),table->getDbDomain());
1125   if(!db) return false;
1126   bool retVal = (bool)db->WriteDb(table,mstoreTime.munixTime);
1127   if(commitWhenDone)table->commitData();
1128
1129 return retVal;
1130 #undef __METHOD__
1131 }
```

4.27.2.68 void StDbManagerImpl::turnOffTimeLogging () [virtual]

Reimplemented from StDbManager.

Definition at line 680 of file StDbManagerImpl.cc.

```
680                                          {
681   StDbManager::turnOffTimeLogging();
682   if(!mhasServerList)lookUpServers();
683   for(ServerList::iterator itr = mservers.begin();
684       itr != mservers.end(); ++itr)
```

```
685      (*itr)->setTimeLogging(misTimeLogged);
686 }
```

4.27.2.69 void StDbManagerImpl::updateDateTime (StDbTime & *t*) [protected]

Definition at line 1029 of file StDbManagerImpl.cc.

```
1029                                                {
1030 #define __METHOD__ "updateDateTime(StDbTime)"
1031   StDbServer* server=findDefaultServer();
1032   StDataBaseI* db=server->useDb();
1033   char* dt;
1034   if(db && (dt=db->getDateTime(t.munixTime))){
1035      t.setDateTime(dt);
1036      delete [] dt;
1037   }
1038 #undef __METHOD__
1039 }
```

4.27.2.70 void StDbManagerImpl::updateUnixTime (StDbTime & *t*) [protected]

Definition at line 1042 of file StDbManagerImpl.cc.

```
1042                                                {
1043   StDbServer* server = findDefaultServer();
1044   StDataBaseI* db=server->useDb();
1045   if(db)t.setUnixTime(db->getUnixTime(t.mdateTime));
1046 }
```

4.27.3 Member Data Documentation

4.27.3.1 StDbDomain StDbManagerImpl::dbDomainFree [private]

enum of last unspecified type

Definition at line 110 of file StDbManagerImpl.hh.

4.27.3.2 StDbType StDbManagerImpl::dbTypeFree [private]

enum list->mapping shortcut

Definition at line 109 of file StDbManagerImpl.hh.

4.27.3.3 StDbTime StDbManagerImpl::mcheckTime [private]

Definition at line 116 of file StDbManagerImpl.hh.

4.27.3.4 StDbLogger StDbManagerImpl::mdataLog [private]

Definition at line 120 of file StDbManagerImpl.hh.

4.27.3.5 dbDomains StDbManagerImpl::mDomains [private]

enum list->mapping shortcut

Definition at line 108 of file StDbManagerImpl.hh.

4.27.3.6 StDbTableFactory* StDbManagerImpl::mfactory [private]

Definition at line 118 of file StDbManagerImpl.hh.

4.27.3.7 bool StDbManagerImpl::mhasDefaultServer [private]

Definition at line 115 of file StDbManagerImpl.hh.

4.27.3.8 bool StDbManagerImpl::mhasServerList [private]

parses strings in XML

Definition at line 114 of file StDbManagerImpl.hh.

4.27.3.9 StDbLogger StDbManagerImpl::mnodeLog [private]

Definition at line 119 of file StDbManagerImpl.hh.

4.27.3.10 parseXmlString StDbManagerImpl::mparser [private]

servers to handle the Query

Definition at line 112 of file StDbManagerImpl.hh.

4.27.3.11 ServerList StDbManagerImpl::mservers [private]

enum of last unspecified domain

Definition at line 111 of file StDbManagerImpl.hh.

4.27.3.12 StDbTime StDbManagerImpl::mstoreTime [private]

Definition at line 117 of file StDbManagerImpl.hh.

4.27.3.13 dbTypes StDbManagerImpl::mTypes [private]

in order to create "singleton" with inheritance

Definition at line 107 of file StDbManagerImpl.hh.

4.27.3.14 short StDbManagerImpl::xmlInputSource

Definition at line 219 of file StDbManagerImpl.hh.

4.27.3.15 std::vector<std::string> StDbManagerImpl::xmlServerList

Definition at line 217 of file StDbManagerImpl.hh.

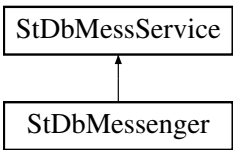
The documentation for this class was generated from the following files:

- StDbManagerImpl.hh
- StDbManagerImpl.cc

4.28 StDbMessenger Class Reference

`#include <StDbMessenger.hh>`

Inheritance diagram for StDbMessenger::



Public Member Functions

- StDbMessenger ()
- StDbMessenger (ostream &os)
- StDbMessenger (StDbMessenger &m)
- virtual ~StDbMessenger ()
- virtual void printMessage (const char *message, StDbMessLevel dbLevel, int lineNumber, const char *className, const char *methodName)
- virtual void printMessage (const char *message, const char *levelString, int lineNumber, const char *className, const char *methodName)

Public Attributes

- ostream & mos

4.28.1 Constructor & Destructor Documentation

4.28.1.1 StDbMessenger::StDbMessenger () [inline]

Definition at line 51 of file StDbMessenger.hh.

```
51 :mos(cout){};
```

4.28.1.2 StDbMessenger::StDbMessenger (ostream & *os*) [inline]

Definition at line 52 of file StDbMessenger.hh.

```
52 :mos(os){};
```

4.28.1.3 StDbMessenger::StDbMessenger (StDbMessenger & *m*) [inline]

Definition at line 53 of file StDbMessenger.hh.

```
53 : mos(m.mos) { mdbLevel=m.getMessLevel(); }
```

4.28.1.4 virtual StDbMessenger::~StDbMessenger () [inline, virtual]

Definition at line 55 of file StDbMessenger.hh.

```
55 {};
```

4.28.2 Member Function Documentation

4.28.2.1 void StDbMessenger::printMessage (const char * *message*, const char * *levelString*, int *lineNumber*, const char * *className*, const char * *methodName*) [virtual]

Implements StDbMessService.

Definition at line 87 of file StDbMessenger.cc.

```
87
88   mos<<" *** "<<levelString<<" line="<<lineNumber<<" *** ";
89   mos<<className<<"::"<<methodName<<" "<<endl<<message<<endl;
90 }
```

4.28.2.2 void StDbMessenger::printMessage (const char * *message*, StDbMessLevel *dbLevel*, int *lineNumber*, const char * *className*, const char * *methodName*) [virtual]

Implements StDbMessService.

Definition at line 49 of file StDbMessenger.cc.

```
49
50
51 if(dbLevel<mdbLevel)return;
52
53  char lString[64];
54  switch(dbLevel){
55  case dbMDebug:
56    {
57       strcpy(lString,"Debug");
58       break;
59    }
60  case dbMWarn:
61    {
62       strcpy(lString,"Warning");
63       break;
64    }
65  case dbMConnect:
66    {
67       strcpy(lString,"Info");
68       break;
69    }
70  case dbMErr:
71    {
72      strcpy(lString,"Error");
73      break;
74    }
75  default:
76    {
77      strcpy(lString," ");
78      break;
79    }
80  }
81 printMessage(message,(const char*)lString,lineNumber,className,methodName);
82 }
```

4.28.3 Member Data Documentation

4.28.3.1 ostream& StDbMessenger::mos

Definition at line 49 of file StDbMessenger.hh.

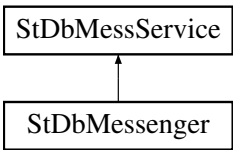
The documentation for this class was generated from the following files:

- StDbMessenger.hh
- StDbMessenger.cc

4.29 StDbMessService Class Reference

`#include <StDbMessService.hh>`

Inheritance diagram for StDbMessService::



Public Member Functions

- StDbMessService ()
- virtual ~StDbMessService ()
- void setMessLevel (StDbMessLevel dbLevel)
- StDbMessLevel getMessLevel ()
- virtual void printMessage (const char *message, StDbMessLevel dbLevel, int lineNumber, const char *className, const char *methodName)=0
- virtual void printMessage (const char *message, const char *levelString, int lineNumber, const char *className, const char *methodName)=0

Protected Attributes

- StDbMessLevel mdbLevel

4.29.1 Constructor & Destructor Documentation

4.29.1.1 StDbMessService::StDbMessService () [inline]

Definition at line 51 of file StDbMessService.hh.

```
51 : mdbLevel(dbMWarn) {};
```

4.29.1.2 virtual StDbMessService::~StDbMessService () [inline, virtual]

Definition at line 52 of file StDbMessService.hh.

```
52 {};
```

4.29.2 Member Function Documentation

4.29.2.1 StDbMessLevel StDbMessService::getMessLevel () [inline]

Definition at line 71 of file StDbMessService.hh.

```
71 {return mdbLevel;}
```

4.29.2.2 virtual void StDbMessService::printMessage (const char * *message*, const char * *levelString*, int *lineNumber*, const char * *className*, const char * *methodName*) [pure virtual]

Implemented in StDbMessenger.

4.29.2.3 virtual void StDbMessService::printMessage (const char * *message*, StDbMessLevel *dbLevel*, int *lineNumber*, const char * *className*, const char * *methodName*) [pure virtual]

Implemented in StDbMessenger.

4.29.2.4 void StDbMessService::setMessLevel (StDbMessLevel *dbLevel*) [inline]

Definition at line 70 of file StDbMessService.hh.

```
70 { mdbLevel=lev;  }
```

4.29.3 Member Data Documentation

4.29.3.1 StDbMessLevel StDbMessService::mdbLevel [protected]

Definition at line 47 of file StDbMessService.hh.

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

- StDbMessService.hh

4.30 StDbModifier Class Reference

```
#include <StDbModifier.h>
```

Public Member Functions

- StDbModifier ()
- virtual ~StDbModifier ()
- virtual Int_t ReadDataFromDB ()
- virtual Int_t WriteDataToDB ()
- virtual void SetTime (unsigned int timestamp)
- virtual void SetDateTime (const char *timestamp)
- virtual void SetDbName (const char *dbname)
- virtual void SetFlavor (const char *flavorname)
- virtual void SetInputFileName (const char *inputfilename)
- virtual void SetOutputFileName (const char *outputfilename)
- virtual void SetTableName (const char *tablename)
- virtual void SetVersionName (const char *versionname)
- virtual void SetDebug ()
- virtual void SetDebug (Int_t debug)

Private Attributes

- char * fDbName
- Int_t fDebug
- char * fTableName
- unsigned int funixTime
- char * fTimestamp
- char * fVersionName
- char * fOutputFileName
- char * fInputFileName
- char * fFlavorName

4.30.1 Constructor & Destructor Documentation

4.30.1.1 StDbModifier::StDbModifier ()

4.30.1.2 StDbModifier::~StDbModifier () [virtual]

Definition at line 83 of file StDbModifier.cxx.

```
{
  // destructor of StDbModifier
  if(fTimestamp) delete [] fTimestamp;

}
```

4.30.2 Member Function Documentation

4.30.2.1 Int_t StDbModifier::ReadDataFromDB () [virtual]

Definition at line 91 of file StDbModifier.cxx.

```
92 {
93   // This funciton will connect database server, get informaion of table and
94   // make a file that contains data of table in database.
95
96   if( funixTime==0 && !(fTimestamp)) {
97     cout<< " No timestamp specified " << endl;
98     return 0;
99   }
100
101   if ( fDbName == 0 ){
102     cout << "  StDbModifier; Set Database name by " << endl;
103     cout << "                StDbModifier::setDbName(TString dbname)";
104     cout << endl;
105     return 0;
106   }
107   if ( fTableName == 0 ){
108     cout << "  StDbModifier; Set table name by " << endl;
109     cout << "                StDbModifier::setTableName(TString tablename)";
110     cout << endl;
111     return 0;
112   }
113
114   StDbManager* mgr = StDbManager::Instance();               // Get the singleton manager
115   if ( fDebug == 1 ){
116     bool isVerbose = true;
117     mgr->setVerbose(isVerbose);                             // Set verbose mode for debuging for fDebu
118   }
119
120   StDbConfigNode* configNode = mgr -> initConfig(fDbName);  // Connect to the db & get an empty contai
121
122   StDbTable* dbtable;
123   if ( fVersionName == 0 ){
124     dbtable = configNode -> addDbTable(fTableName);
125     // Add a table to the container with descriptor given by Database with wsing version name as "defa
126   }else{
127     dbtable = configNode -> addDbTable(fTableName,fVersionName);
128     // Add a table to the container with descriptor given by Database with wsing version name specifie
129   }
130   if ( dbtable == 0 ){                              // If table asigned by fTableName does not exist i
131     cout << " No Table : " << fTableName << endl;   // program is stoped and exit from this function.
132     return 0;                                       //
133   }
134
135   if ( fFlavorName != 0 ){
136     dbtable -> setFlavor(fFlavorName);
137     cout << "Flavor is set as " << fFlavorName << " by StDbTable::setFlavor." << endl;
138   }else{
139     cout << "Flavor is NOT assigned. Default value is set as 'ofl'. " << endl;
140     dbtable -> setFlavor("ofl");
141   }
142
143
144   if(funixTime) {
145     mgr->setRequestTime(funixTime);
146   } else {
147     mgr->setRequestTime(fTimestamp);
148   }
149
150   //  mgr -> setRequestTime(fTimestamp);          // Set the request time
151   mgr -> fetchDbTable(dbtable);               // Fetch the data from Database
```

```

  void* cstruct = dbtable -> GetTableCpy();   // Get pointer of table and copy to c-structure
  Int_t   nrows = dbtable -> GetNRows();      // Get number of raws in the table

  TTable* table = TTable::New(fTableName,fTableName,cstruct,nrows);
  // Create new TTable object for c-structure

  ofstream ofs(fOutputFileName);  // Open a file
  table -> SavePrimitive(ofs,0);  // Write information of c-structure from object of TTable to the fil
  ofs.close();                    // Close the file

  return 1;
}
```

4.30.2.2 void StDbModifier::SetDateTime (const char * *timestamp*) [virtual]

Definition at line 283 of file StDbModifier.cxx.

```
{
  fTimestamp = new char[strlen(timestamp)+1];
  strcpy(fTimestamp,timestamp);
}
```

4.30.2.3 void StDbModifier::SetDbName (const char * *dbname*) [virtual]

Definition at line 292 of file StDbModifier.cxx.

```
{
  fDbName = new char[strlen(dbname)+1];
  strcpy(fDbName,dbname);
}
```

4.30.2.4 virtual void StDbModifier::SetDebug (Int_t *debug*) [inline, virtual]

Definition at line 61 of file StDbModifier.h.

```
{ fDebug = debug; }
```

4.30.2.5 virtual void StDbModifier::SetDebug () [inline, virtual]

Definition at line 60 of file StDbModifier.h.

```
{ fDebug = 1    ; }
```

4.30.2.6 void StDbModifier::SetFlavor (const char * *flavorname*) [virtual]

Definition at line 327 of file StDbModifier.cxx.

```
{
  fFlavorName = new char[strlen(flavorname)+1];
  strcpy(fFlavorName,flavorname);
  cout << "       Flavor is set " << flavorname  << endl;
  cout << "       Flavor is set " << fFlavorName << endl;
}
```

4.30.2.7 void StDbModifier::SetInputFileName (const char * *inputfilename*) [virtual]

Definition at line 299 of file StDbModifier.cxx.

```
300 {
301   fInputFileName = new char[strlen(inputfilename)+1];
302   strcpy(fInputFileName,inputfilename);
303 }
```

4.30.2.8 void StDbModifier::SetOutputFileName (const char * *outputfilename*) [virtual]

Definition at line 306 of file StDbModifier.cxx.

```
307 {
308   fOutputFileName = new char[strlen(outputfilename)+1];
309   strcpy(fOutputFileName,outputfilename);
310 }
```

4.30.2.9 void StDbModifier::SetTableName (const char * *tablename*) [virtual]

Definition at line 313 of file StDbModifier.cxx.

```
314 {
315   fTableName = new char[strlen(tablename)+1];
316   strcpy(fTableName,tablename);
317 }
```

4.30.2.10 void StDbModifier::SetTime (unsigned int *timestamp*) [virtual]

Definition at line 289 of file StDbModifier.cxx.

```
289 { funixTime=time;}
```

4.30.2.11 void StDbModifier::SetVersionName (const char * *versionname*) [virtual]

Definition at line 320 of file StDbModifier.cxx.

```
321 {
322   fVersionName = new char[strlen(versionname)+1];
323   strcpy(fVersionName,versionname);
324 }
```

4.30.2.12 Int_t StDbModifier::WriteDataToDB () [virtual]

Definition at line 168 of file StDbModifier.cxx.

```
{
  // This funciton will make a table as object of TTable.
  // Data of the table will be read from a file that is generated from
  // StDbModifier::ReadDataFromDB().
  // After data is read, connection to database server is made and data will
  // be stored in database.


  if( funixTime==0 && !(fTimestamp)) {
    cout<< " No timestamp specified " << endl;
    return 0;
  }

  if ( fDbName == 0 ){
    cout << "  StDbModifier; Set Database name by " << endl;
    cout << "                StDbModifier::setDbName(TString dbname)";
    cout << endl;
    return 0;
  }
  if ( fTableName == 0 ){
    cout << "  StDbModifier; Set table name by " << endl;
    cout << "                StDbModifier::setTableName(TString tablename)";
    cout << endl;
    return 0;
  }

  TString command;                  // Make a string to load Root C macro generated
  command = ".L ";                  // by ReadDataFromDB()
  command += fInputFileName;

  if ( fDebug == 1 ) printf("   LoadTable: %s\n",(const char*)command);

  gInterpreter->ProcessLine(command);                                  // Load the file in CINT
  TTable* table = (TTable*) gInterpreter->Calc("CreateTable()");      // execute comand in the file
  command.ReplaceAll(".L ",".U ");                                     // chage comand .L to .U
  gInterpreter->ProcessLine(command);                                  // unload file from CINT

  void* cstruct = table -> GetArray();      // Get pointer of table and copy to c-structure
  Int_t   nrows = table -> GetNRows();      // Get number of raws in the table
 //MPD - below is need TTable info to correct c-array 8 byte padding issue
 //could have broke it out - but I chose to keep all the TTable (root) stuff together

   Int_t  rowSize  = table -> GetRowSize();   // Get the size (in bytes) of each row - fixes gap/paddi
//   Int_t nCols  = table -> GetNumberOfColumns();

 StDbTableDescriptor* TD = new StDbTableDescriptor();
  TD->storeRowSize(rowSize);
   //table -> Print(0,2);  //uncomment for debugging



  StDbManager* mgr = StDbManager::Instance();               // Get the singleton manager
  if ( fDebug == 1 ){
    bool isVerbose = true;
    mgr->setVerbose(isVerbose);                             // Set verbose mode for debuging for fDebu
  }
  StDbConfigNode* configNode = mgr -> initConfig(fDbName);  // Connect to the db & get an empty contai

  StDbTable* dbtable;
  if ( fVersionName == 0 ){
    dbtable = configNode -> addDbTable(fTableName);
    // Add a table to the container with descriptor given by Database with wsing version name as "defa
  }else{
    dbtable = configNode -> addDbTable(fTableName,fVersionName);
    // Add a table to the container with descriptor given by Database with wsing version name specifie
  }

```

```


  if ( dbtable == 0 ){                            // If table asigned by fTableName does not exist in
    cout << " No Table : " << fTableName << endl; // program is stoped and exit from this function.
    return 0;                                     //
  }

  if ( fFlavorName != 0 ){
    cout << "set Flavor" << endl;
    dbtable -> setFlavor(fFlavorName);
    cout << "Flavor is set as " << fFlavorName << " by StDbTable::setFlavor." << endl;
  }else{
    cout << "Flavor is NOT assigned. Default value is set as 'ofl'. " << endl;
    dbtable -> setFlavor("ofl");
  }

  int dbnrows=dbtable->GetNRows();
  int* eidList=0;
  int tmp;
  int* dbeidList=dbtable->getElementID(tmp);
  if(nrows>dbnrows){
    eidList=new int[nrows];
    int i;
    for(i=0;i<dbnrows;i++)eidList[i]=dbeidList[i];
    for(i=dbnrows;i<nrows;i++)eidList[i]=eidList[dbnrows-1]+i;
  }


  dbtable -> SetTable((char*)cstruct,nrows,eidList);      // Put data in local table on memory
  //  dbtable -> SetNRows(nrows);                     // Set number of rows on table in database

  //  mgr -> setStoreTime(fTimestamp);          // Set the time stamp
  if(funixTime) {
    mgr->setStoreTime(funixTime);
  } else {
    mgr->setStoreTime(fTimestamp);
  }

  int retVal=0;
  if(mgr -> storeDbTable(dbtable)) retVal=1;             // Fetch the data
  if(eidList) delete [] eidList;


  return retVal;
}
```

4.30.3 Member Data Documentation

4.30.3.1 char* StDbModifier::fDbName [private]

Definition at line 32 of file StDbModifier.h.

4.30.3.2 Int_t StDbModifier::fDebug [private]

Definition at line 33 of file StDbModifier.h.

4.30.3.3 char* StDbModifier::fFlavorName [private]

Definition at line 40 of file StDbModifier.h.

4.30.3.4 char* StDbModifier::fInputFileName [private]

Definition at line 39 of file StDbModifier.h.

4.30.3.5 char* StDbModifier::fOutputFileName [private]

Definition at line 38 of file StDbModifier.h.

4.30.3.6 char* StDbModifier::fTableName [private]

Definition at line 34 of file StDbModifier.h.

4.30.3.7 char* StDbModifier::fTimestamp [private]

Definition at line 36 of file StDbModifier.h.

4.30.3.8 unsigned int StDbModifier::funixTime [private]

Definition at line 35 of file StDbModifier.h.

4.30.3.9 char* StDbModifier::fVersionName [private]

Definition at line 37 of file StDbModifier.h.

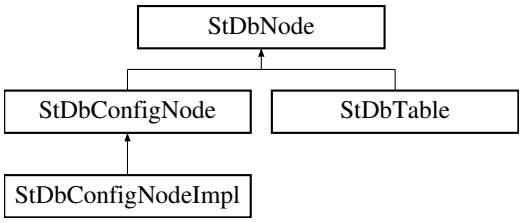
The documentation for this class was generated from the following files:

- StDbModifier.h
- StDbModifier.cxx

4.31 StDbNode Class Reference

`#include <StDbNode.hh>`

Inheritance diagram for StDbNode::



Public Member Functions

- StDbNode ()
- StDbNode (const char *name, const char *versionKey)
- StDbNode (const char *name)
- StDbNode (StDbNode &node)
- virtual ~StDbNode ()
- char * getName ()
- char * printName ()
- char * getMyName ()
- char * getVersion ()
- char * printVersion ()
- char * getDbName ()
- char * printDbName ()
- StDbType getDbType () const
- StDbDomain getDbDomain () const
- int getNodeID () const
- char * getNodeType ()
- char * printNodeType ()
- void setName (const char *nodeName)
- void setVersion (const char *nodeVersion)
- void setDbName (const char *nodeDbName)
- void setDbType (StDbType type)
- void setDbDomain (StDbDomain domain)
- void setNodeID (int id)
- void setNodeType (const char *nodeType)
- bool canRollBack () const
- void addWrittenNode (int dataID)
- void commit ()
- bool checkName (const char *nodeName) const
- bool checkVersion (const char *nodeVersion) const
- bool checkNode (const char *nodeName, const char *nodeVersion) const
- bool IsConfigured () const
- void setConfigured (bool isConfigured)
- bool isNode (StDbType type, StDbDomain domain)
- virtual bool IsTable () const

- char * mstrDup (const char *s2)
- int * decodeElementID (const char *elementID, int &numRows)
- char * getNextID (char *&currentElement) const

Protected Attributes

- char * mname
- char * mversion
- char * mdbName
- StDbType mdbType
- StDbDomain mdbDomain
- int mnodeID
- char * mnodeType
- bool misConfigured
- bool mcanRollBack

4.31.1 Constructor & Destructor Documentation

4.31.1.1 StDbNode::StDbNode () [inline]

Definition at line 90 of file StDbNode.hh.

```
90 : mname(0), mversion(0), mdbName(0), mnodeID(0), mnodeType(0), misConfigured(false), mcanRollBack(false
```

4.31.1.2 StDbNode::StDbNode (const char * *name*, const char * *versionKey*)

Definition at line 84 of file StDbNode.cc.

```
84                                                          :  mname(0), mversion(0), mdbName(0), mnodeID
85
86   setName(name);
87   setVersion(versionKey);
88   misConfigured = false;
89   mcanRollBack  = false;
90 }
```

4.31.1.3 StDbNode::StDbNode (const char * *name*)

Definition at line 93 of file StDbNode.cc.

```
93                                  : mname(0), mversion(0), mdbName(0), mnodeID(0) ,mnodeType(0){
94   setName(name);
95   setVersion(StDbDefaults::Instance()->printVersion());
96   misConfigured = false;
97   mcanRollBack  = false;
98 }
```

4.31.1.4 StDbNode::StDbNode (StDbNode & *node*)

Definition at line 101 of file StDbNode.cc.

```
101                                 {
102
103   mname         = node.getName();
104   mversion      = node.getVersion();
105   mdbName       = node.getDbName();
106   mdbType       = node.getDbType();
107   mdbDomain     = node.getDbDomain();
108   mnodeID       = node.getNodeID();
109   mnodeType     = node.getNodeType();
110   misConfigured = node.IsConfigured();
111   mcanRollBack  = node.canRollBack();
112 }
```

4.31.1.5 StDbNode::~StDbNode () [virtual]

Definition at line 115 of file StDbNode.cc.

```
115                     {
116   if(mname)     delete [] mname;
117   if(mversion)  delete [] mversion;
118   if(mdbName)   delete [] mdbName;
119   if(mnodeType) delete [] mnodeType;
120 }
```

4.31.2 Member Function Documentation

4.31.2.1 void StDbNode::addWrittenNode (int *dataID*) [inline]

Definition at line 158 of file StDbNode.hh.

```
158                                                 {
159  mcanRollBack=true;
160  mnodeID=dataID;
161 }
```

4.31.2.2 bool StDbNode::canRollBack () const [inline]

Definition at line 157 of file StDbNode.hh.

```
157 { return mcanRollBack; }
```

4.31.2.3 bool StDbNode::checkName (const char * *nodeName*) const [inline]

Definition at line 163 of file StDbNode.hh.

```
163                                                         {
164 return (mname && (strcmp(mname,nodeName)==0)) ? true : false;
165 }
```

4.31.2.4 bool StDbNode::checkNode (const char * *nodeName*, const char * *nodeVersion*) const [inline]

Definition at line 169 of file StDbNode.hh.

```
169                                                                    {
170 return (checkName(name) && checkVersion(version)) ? true : false;
171 }
```

4.31.2.5 bool StDbNode::checkVersion (const char * *nodeVersion*) const [inline]

Definition at line 166 of file StDbNode.hh.

```
166                                                          {
167 return (mversion && (strcmp(mversion,nodeVersion)==0)) ? true : false;
168 }
```

4.31.2.6 void StDbNode::commit () [inline]

Definition at line 162 of file StDbNode.hh.

```
162 { mcanRollBack = false; }
```

4.31.2.7 int * StDbNode::decodeElementID (const char * *elementID*, int & *numRows*)

Definition at line 147 of file StDbNode.cc.

```
147                                                              {
148
149 numRows=1;
150 int * retVal = 0;
151 char* id=strstr((char*)elemID,"None");
152
153 if(id){
154   numRows=1;
155   int* e = new int[1]; *e=0;
156   return e;
157 }
158
159 char* tmpName = new char[strlen(elemID)+1];
160 strcpy(tmpName,elemID);
161
162 int numElements = 1;
163 id = strstr(tmpName,",");
164 char* id1;
165 char* id2;
166
167 id2 = strstr(tmpName,"-");
168 StString sl;
169
170 if(id2 && ( (id && id2<id) || !id)){
171   id=id2;
172   id[0]=',';
173   sl<<"r";
174 } else {
175   sl<<"l";
```

```
}

int numEntries = 1;
if(id)id++;
while(id){
  //  cout << "id = " << id << endl;
   numEntries++;
   id1=strstr(id,",");
   id2=strstr(id,"-");
   id = id1;
   if(id && id2 && id2<id){
       id=id2;
       id[0]=',';
       sl<<"r";
   } else {
       sl<<"l";
   }
   if(id)id++;
}
 char* islist=new char[strlen((sl.str()).c_str())+1];
 strcpy(islist,(sl.str()).c_str());

 // cout << "My string list = " << islist << endl;

 int* tmpElements = new int[100000];
 char* p1=&tmpName[0];
 char* anID=0;
 anID = getNextID(p1);
 tmpElements[0] = atoi(anID);
 if(anID)delete [] anID;
 numElements = 1;
 int iEnd, iStart, k;
 for(int ient=1;ient<numEntries;ient++){
   anID = getNextID(p1);
   if(islist[ient-1]=='r'){
     iEnd = atoi(anID);
     iStart = tmpElements[numElements-1];
     int irange=iEnd-iStart;
     for(int ir=1;ir<=irange;ir++){
       numElements++;
       tmpElements[numElements-1]=iStart+ir;
     }
   } else {
     numElements++;
     tmpElements[numElements-1]=atoi(anID);
   }
   if(anID) delete [] anID;
 }

 retVal = new int[numElements];
 for(k=0;k<numElements;k++)retVal[k]=tmpElements[k];
 numRows = numElements;

 delete [] islist;
 delete [] tmpElements;
 delete [] tmpName;

return retVal;
}
```

4.31.2.8 StDbDomain StDbNode::getDbDomain () const [inline]

Definition at line 147 of file StDbNode.hh.

```
147 { return mdbDomain; }
```

4.31.2.9 char * StDbNode::getDbName ()

Definition at line 126 of file StDbNode.cc.

```
126 { return mstrDup((const char*)mdbName); }
```

4.31.2.10 StDbType StDbNode::getDbType () const [inline]

Definition at line 146 of file StDbNode.hh.

```
146 { return mdbType;}
```

4.31.2.11 char * StDbNode::getMyName () [inline]

Definition at line 145 of file StDbNode.hh.

```
145 { return printName(); }
```

4.31.2.12 char * StDbNode::getName ()

Definition at line 124 of file StDbNode.cc.

```
124 { return mstrDup((const char*)mname); };
```

4.31.2.13 char * StDbNode::getNextID (char *& *currentElement*) const

Definition at line 238 of file StDbNode.cc.

```
238                                                 {
239
240 char* nextID = 0;
241 if(!currentElement)return nextID;
242
243 char* id = strstr(currentElement,",");
244
245 if(!id) {
246   nextID = new char[strlen(currentElement)+1];
247   strcpy(nextID,currentElement);
248   currentElement = 0;
249 } else {
250   int iloc = id-currentElement;
251   nextID = new char[iloc+1];
252   strncpy(nextID,currentElement,iloc);
253   nextID[iloc]='\0';
254   currentElement = id; currentElement++;
255 }
256
257 return nextID;
258 }
```

4.31.2.14 int StDbNode::getNodeID () const [inline]

Definition at line 150 of file StDbNode.hh.

```
150 { return mnodeID; }
```

4.31.2.15 char * StDbNode::getNodeType ()

Definition at line 127 of file StDbNode.cc.

```
127 { return mstrDup(mnodeType); }
```

4.31.2.16 char * StDbNode::getVersion ()

Definition at line 125 of file StDbNode.cc.

```
125 {  return mstrDup((const char*)mversion); }
```

4.31.2.17 bool StDbNode::IsConfigured () const [inline]

Definition at line 153 of file StDbNode.hh.

```
153 { return misConfigured; }
```

4.31.2.18 bool StDbNode::isNode (StDbType *type*, StDbDomain *domain*) [inline]

Definition at line 154 of file StDbNode.hh.

```
154                                                         {
155   return ( (type==mdbType) && (domain==mdbDomain) ) ? true : false;
156 }
```

4.31.2.19 bool StDbNode::IsTable () const [inline, virtual]

Reimplemented in StDbTable.

Definition at line 172 of file StDbNode.hh.

```
172 { return false; }
```

4.31.2.20 char * StDbNode::mstrDup (const char * *s2*)

Definition at line 261 of file StDbNode.cc.

```
261                                     {
262
263   char* s1=0;
264   if(!s2) return s1;
265   s1 = new char[strlen(s2)+1];
266   strcpy(s1,s2);
267 return s1;
268 }
```

4.31.2.21 char * StDbNode::printDbName () [inline]

Definition at line 142 of file StDbNode.hh.

```
142 { return mdbName; };
```

4.31.2.22 char * StDbNode::printName () [inline]

Definition at line 141 of file StDbNode.hh.

```
141 { return mname; };
```

4.31.2.23 char * StDbNode::printNodeType () [inline]

Definition at line 143 of file StDbNode.hh.

```
143 { return mnodeType; };
```

4.31.2.24 char * StDbNode::printVersion () [inline]

Definition at line 144 of file StDbNode.hh.

```
144 { return mversion; };
```

4.31.2.25 void StDbNode::setConfigured (bool *isConfigured*) [inline]

Definition at line 152 of file StDbNode.hh.

```
152 { misConfigured=isC; }
```

4.31.2.26 void StDbNode::setDbDomain (StDbDomain *domain*) [inline]

Definition at line 149 of file StDbNode.hh.

```
149 { mdbDomain=domain; }
```

4.31.2.27 void StDbNode::setDbName (const char * *nodeDbName*)

Definition at line 137 of file StDbNode.cc.

```
137                                                 {
138   if(mdbName)delete [] mdbName;
139   mdbName=mstrDup(dbName);
140 }
```

4.31.2.28 void StDbNode::setDbType (StDbType *type*) [inline]

Definition at line 148 of file StDbNode.hh.

```
148 { mdbType=type;}
```

4.31.2.29 void StDbNode::setName (const char * *nodeName*)

Definition at line 129 of file StDbNode.cc.

```
129                                                 {
130   if(mname) delete [] mname;
131    mname=mstrDup(nodeName);
132 }
```

4.31.2.30 void StDbNode::setNodeID (int *id*) [inline]

Definition at line 151 of file StDbNode.hh.

```
151 {mnodeID = id; }
```

4.31.2.31 void StDbNode::setNodeType (const char * *nodeType*)

Definition at line 141 of file StDbNode.cc.

```
141                                                 {
142   if(mnodeType) delete [] mnodeType;
143   mnodeType=mstrDup(type);
144 }
```

4.31.2.32 void StDbNode::setVersion (const char * *nodeVersion*)

Definition at line 133 of file StDbNode.cc.

```
133                                                 {
134   if(mversion) delete [] mversion;
135   mversion=mstrDup(version);
136 }
```

4.31.3 Member Data Documentation

4.31.3.1 bool StDbNode::mcanRollBack [protected]

Definition at line 86 of file StDbNode.hh.

4.31.3.2 StDbDomain StDbNode::mdbDomain [protected]

Definition at line 79 of file StDbNode.hh.

4.31.3.3 char* StDbNode::mdbName [protected]

Definition at line 77 of file StDbNode.hh.

4.31.3.4 StDbType StDbNode::mdbType [protected]

Definition at line 78 of file StDbNode.hh.

4.31.3.5 bool StDbNode::misConfigured [protected]

Definition at line 85 of file StDbNode.hh.

4.31.3.6 char* StDbNode::mname [protected]

Definition at line 75 of file StDbNode.hh.

4.31.3.7 int StDbNode::mnodeID [protected]

Definition at line 82 of file StDbNode.hh.

4.31.3.8 char* StDbNode::mnodeType [protected]

Definition at line 83 of file StDbNode.hh.

4.31.3.9 char* StDbNode::mversion [protected]

Definition at line 76 of file StDbNode.hh.

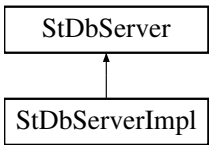
The documentation for this class was generated from the following files:

- StDbNode.hh
- StDbNode.cc

4.32 StDbServer Class Reference

`#include <StDbServer.hh>`

Inheritance diagram for StDbServer::



Public Member Functions

- StDbServer ()
- StDbServer (const char *name, const char *host, const char *sock, int port)
- StDbServer (StDbServer &server)
- virtual ~StDbServer ()
- virtual void setHostName (const char *name)
- virtual void setUnixSocket (const char *name)
- virtual void setPortNumber (int port)
- virtual void setServerName (const char *name)
- virtual void setUser (const char *name, const char *pword=0)
- virtual char * getServerName () const
- virtual char * getHostName () const
- virtual char * getUnixSocket () const
- virtual int getPortNumber () const
- virtual char * printServerName ()
- virtual char * printHostName ()
- virtual char * printUnixSocket ()
- virtual char * printUser ()
- virtual char * printPword ()
- virtual void setIsDefaultServer ()
- virtual bool isDefault () const
- virtual void addDataBase (StDbType type, StDbDomain domain)=0
- virtual void addDataBase (const char *typeName, const char *domName)=0
- virtual StDataBaseI * useDb (StDbType type, StDbDomain domain)=0
- virtual StDataBaseI * useDb (const char *typeName, const char *domName)=0
- virtual StDataBaseI * useDb ()=0
- virtual bool isConnected ()=0
- virtual void closeConnection ()=0
- virtual void setTimeLogging (bool isTimeLogged)=0
- virtual double getQueryTimes ()=0
- virtual double getSocketTimes ()=0
- virtual double getConnectTimes ()=0

Protected Member Functions

- char * mstringDup (const char *str) const

Protected Attributes

- char * mserverName
- char * mhostName
- char * munixSocket
- char * muserName
- char * mpword
- int mportNumber
- bool misDefault

4.32.1 Constructor & Destructor Documentation

4.32.1.1 StDbServer::StDbServer ()

Definition at line 90 of file StDbServer.cc.

```
90 : mserverName(0),mhostName(0), munixSocket(0), muserName(0), mpword(0), misDefault(false){};
```

4.32.1.2 StDbServer::StDbServer (const char * *name*, const char * *host*, const char * *sock*, int *port*)

Definition at line 92 of file StDbServer.cc.

```
92                                                                                                  : mserverName(0),m
93
94   setServerName(name);
95   setHostName(host);
96   setUnixSocket(sock);
97 };
```

4.32.1.3 StDbServer::StDbServer (StDbServer & *server*)

Definition at line 99 of file StDbServer.cc.

```
99                                          :  mserverName(0), mhostName(0), munixSocket(0) , muserName(0)
100
101   setServerName(server.printServerName());
102   setHostName(server.printHostName());
103   setUnixSocket(server.printUnixSocket());
104   setPortNumber(server.getPortNumber());
105 }
```

4.32.1.4 StDbServer::~StDbServer () [virtual]

Definition at line 108 of file StDbServer.cc.

```
108                        {
109    if(mserverName)delete [] mserverName;
110    if(mhostName)  delete [] mhostName;
111    if(munixSocket)delete [] munixSocket;
112    if(muserName)  delete [] muserName;
113    if(mpword)     delete [] mpword;
114 }
```

4.32.2 Member Function Documentation

4.32.2.1 virtual void StDbServer::addDataBase (const char * *typeName*, const char * *domName*) [pure virtual]

Implemented in StDbServerImpl.

4.32.2.2 virtual void StDbServer::addDataBase (StDbType *type*, StDbDomain *domain*) [pure virtual]

Implemented in StDbServerImpl.

4.32.2.3 virtual void StDbServer::closeConnection () [pure virtual]

Implemented in StDbServerImpl.

4.32.2.4 virtual double StDbServer::getConnectTimes () [pure virtual]

Implemented in StDbServerImpl.

4.32.2.5 char * StDbServer::getHostName () const [virtual]

Definition at line 131 of file StDbServer.cc.

```
131 { return mstringDup(mhostName); };
```

4.32.2.6 int StDbServer::getPortNumber () const [inline, virtual]

Definition at line 144 of file StDbServer.hh.

```
144 { return mportNumber; }
```

4.32.2.7 virtual double StDbServer::getQueryTimes () [pure virtual]

Implemented in StDbServerImpl.

4.32.2.8 char * StDbServer::getServerName () const [virtual]

Definition at line 133 of file StDbServer.cc.

```
133 { return mstringDup(mserverName); }
```

4.32.2.9 virtual double StDbServer::getSocketTimes () [pure virtual]

Implemented in StDbServerImpl.

4.32.2.10 char * StDbServer::getUnixSocket () const [virtual]

Definition at line 132 of file StDbServer.cc.

```
132 { return mstringDup(munixSocket); }
```

4.32.2.11 virtual bool StDbServer::isConnected () [pure virtual]

Implemented in StDbServerImpl.

4.32.2.12 bool StDbServer::isDefault () const [inline, virtual]

Definition at line 151 of file StDbServer.hh.

```
151 { return misDefault; }
```

4.32.2.13 char * StDbServer::mstringDup (const char * *str*) const [protected]

Definition at line 137 of file StDbServer.cc.

```
137                                                    {
138
139 char* retString=0;
140 if(!str)return retString;
141 retString = new char[strlen(str)+1];
142 strcpy(retString,str);
143 return retString;
144 }
```

4.32.2.14 char * StDbServer::printHostName () [inline, virtual]

Definition at line 146 of file StDbServer.hh.

```
146 { return mhostName; }
```

4.32.2.15 char * StDbServer::printPword () [inline, virtual]

Definition at line 149 of file StDbServer.hh.

```
149 { return mpword; }
```

4.32.2.16 char * StDbServer::printServerName () [inline, virtual]

Definition at line 145 of file StDbServer.hh.

```
145 { return mserverName; }
```

4.32.2.17 char * StDbServer::printUnixSocket () [inline, virtual]

Definition at line 147 of file StDbServer.hh.

```
147 { return munixSocket; }
```

4.32.2.18 char * StDbServer::printUser () [inline, virtual]

Definition at line 148 of file StDbServer.hh.

```
148 { return muserName; }
```

4.32.2.19 void StDbServer::setHostName (const char * *name*) [virtual]

Definition at line 117 of file StDbServer.cc.

```
117 {mhostName=mstringDup(name); }
```

4.32.2.20 void StDbServer::setIsDefaultServer () [inline, virtual]

Definition at line 150 of file StDbServer.hh.

```
150 { misDefault=true; }
```

4.32.2.21 void StDbServer::setPortNumber (int *port*) [inline, virtual]

Definition at line 143 of file StDbServer.hh.

```
143 { mportNumber = port;}
```

4.32.2.22 void StDbServer::setServerName (const char * *name*) [virtual]

Definition at line 119 of file StDbServer.cc.

```
119 {mserverName=mstringDup(name);}
```

4.32.2.23 virtual void StDbServer::setTimeLogging (bool *isTimeLogged*) [pure virtual]

Implemented in StDbServerImpl.

4.32.2.24 void StDbServer::setUnixSocket (const char * *name*) [virtual]

Definition at line 118 of file StDbServer.cc.

```
118 {munixSocket=mstringDup(name);}
```

4.32.2.25 void StDbServer::setUser (const char * *name*, const char * *pword* = 0) [virtual]

Definition at line 124 of file StDbServer.cc.

```
124                                                     {
125   muserName=mstringDup(name);
126   mpword=mstringDup(pword);
127 }
```

4.32.2.26 virtual StDataBaseI* StDbServer::useDb () [pure virtual]

Implemented in StDbServerImpl.

4.32.2.27 virtual StDataBaseI* StDbServer::useDb (const char * *typeName*, const char * *domName*) [pure virtual]

Implemented in StDbServerImpl.

4.32.2.28 virtual StDataBaseI* StDbServer::useDb (StDbType *type*, StDbDomain *domain*) [pure virtual]

Implemented in StDbServerImpl.

4.32.3 Member Data Documentation

4.32.3.1 char* StDbServer::mhostName [protected]

Definition at line 90 of file StDbServer.hh.

4.32.3.2 bool StDbServer::misDefault [protected]

Definition at line 95 of file StDbServer.hh.

4.32.3.3 int StDbServer::mportNumber [protected]

Definition at line 94 of file StDbServer.hh.

4.32.3.4 char* StDbServer::mpword [protected]

Definition at line 93 of file StDbServer.hh.

4.32.3.5 char* StDbServer::mserverName [protected]

Definition at line 89 of file StDbServer.hh.

4.32.3.6 char* StDbServer::munixSocket [protected]

Definition at line 91 of file StDbServer.hh.

4.32.3.7 char* StDbServer::muserName [protected]

Definition at line 92 of file StDbServer.hh.

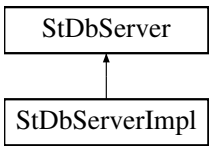
The documentation for this class was generated from the following files:

- StDbServer.hh
- StDbServer.cc

4.33 StDbServerImpl Class Reference

```
#include <StDbServerImpl.hh>
```

Inheritance diagram for StDbServerImpl::



Public Member Functions

- StDbServerImpl ()
- StDbServerImpl (const char *name, const char *host, const char *socket, int port)
- StDbServerImpl (StDbServerImpl &server)
- virtual ~StDbServerImpl ()
- virtual void addDataBase (StDbType type, StDbDomain domain)
- virtual void addDataBase (const char *typeName, const char *domName)
- virtual StDataBaseI * useDb (StDbType type, StDbDomain domain)
- virtual StDataBaseI * useDb (const char *typeName, const char *domName)
- virtual StDataBaseI * useDb ()
- virtual bool isConnected ()
- virtual void closeConnection ()
- virtual void setTimeLogging (bool isTimeLogged)
- virtual double getQueryTimes ()
- virtual double getSocketTimes ()
- virtual double getConnectTimes ()
- void PointMysqlDb (StDbManagerImpl *m)

Protected Member Functions

- void deleteDataBases ()

Protected Attributes

- MysqlDb Db
- StDbBuffer buff
- StDataBaseI * mcurrentDb
- DbList mdataBases

4.33.1 Constructor & Destructor Documentation

4.33.1.1 StDbServerImpl::StDbServerImpl ()

Definition at line 43 of file StDbServerImpl.cc.

```
43 : StDbServer(), mcurrentDb(0){}
```

4.33.1.2 StDbServerImpl::StDbServerImpl (const char * *name*, const char * *host*, const char * *socket*, int *port*)

Definition at line 45 of file StDbServerImpl.cc.

```
45 : StDbServer(name,host,sock,port), mcurrentDb(0) {};
```

4.33.1.3 StDbServerImpl::StDbServerImpl (StDbServerImpl & *server*)

Definition at line 47 of file StDbServerImpl.cc.

```
47 :  StDbServer(server), mcurrentDb(0){}
```

4.33.1.4 StDbServerImpl::~StDbServerImpl () [virtual]

Definition at line 49 of file StDbServerImpl.cc.

```
49 {   deleteDataBases(); }
```

4.33.2 Member Function Documentation

4.33.2.1 void StDbServerImpl::addDataBase (const char * *typeName*, const char * *domName*) [virtual]

Implements StDbServer.

Definition at line 67 of file StDbServerImpl.cc.

```
67                                                                  {
68
69   if(!useDb(type,domain)){
70     mcurrentDb = new StDbSql(Db,buff,type,domain);
71     if(!Db.IsConnected())Db.Connect(mhostName,muserName,mpword,mcurrentDb->printDbName(),mportNumber);
72     mcurrentDb->use();
73     mdataBases.push_back(mcurrentDb);
74   }
75 }
```

4.33.2.2 void StDbServerImpl::addDataBase (StDbType *type*, StDbDomain *domain*) [virtual]

Implements StDbServer.

Definition at line 54 of file StDbServerImpl.cc.

```
54                                                                {
55
56   StDataBaseI* db=useDb(type,domain);
57   if(!db){
58     mcurrentDb = new StDbSql(Db,buff,type,domain);
59     if(!Db.IsConnected())Db.Connect(mhostName,muserName,mpword,mcurrentDb->printDbName(),mportNumber);
60     mcurrentDb->use();
61     mdataBases.push_back(mcurrentDb);
62   }
63 }
```

4.33.2.3 void StDbServerImpl::closeConnection () [virtual]

Implements StDbServer.

Definition at line 154 of file StDbServerImpl.cc.

```
154 {  Db.Close();  }
```

4.33.2.4 void StDbServerImpl::deleteDataBases () [protected]

Definition at line 137 of file StDbServerImpl.cc.

```
137                                 {
138
139  DbList::iterator itr;
140  StDataBaseI* db;
141   do {
142       for(itr = mdataBases.begin(); itr != mdataBases.end(); ++itr){
143          db = *itr;
144          mdataBases.erase(itr);
145          delete db;
146          break;
147         }
148      } while( mdataBases.begin() != mdataBases.end() );
149
150 }
```

4.33.2.5 double StDbServerImpl::getConnectTimes () [virtual]

Implements StDbServer.

Definition at line 159 of file StDbServerImpl.cc.

```
159 {return Db.mconnectLog.getTotalTimes();};
```

4.33.2.6 double StDbServerImpl::getQueryTimes () [virtual]

Implements StDbServer.

Definition at line 157 of file StDbServerImpl.cc.

```
157 {return Db.mqueryLog.getTotalTimes();};
```

4.33.2.7 double StDbServerImpl::getSocketTimes () [virtual]

Implements StDbServer.

Definition at line 158 of file StDbServerImpl.cc.

```
158 {return Db.msocketLog.getTotalTimes();};
```

4.33.2.8 bool StDbServerImpl::isConnected () [virtual]

Implements StDbServer.

Definition at line 153 of file StDbServerImpl.cc.

```
153 {  return Db.IsConnected(); }
```

4.33.2.9 void StDbServerImpl::PointMysqlDb (StDbManagerImpl * *m*) [inline]

Definition at line 90 of file StDbServerImpl.hh.

```
90 {Db.my_manager = m;}
```

4.33.2.10 void StDbServerImpl::setTimeLogging (bool *isTimeLogged*) [virtual]

Implements StDbServer.

Definition at line 156 of file StDbServerImpl.cc.

```
156 { Db.mlogTime=isTLog; }
```

4.33.2.11 StDataBaseI * StDbServerImpl::useDb () [virtual]

Implements StDbServer.

Definition at line 121 of file StDbServerImpl.cc.

```
121                     {
122
123   if(!mcurrentDb){
124     mcurrentDb= new StDbSql(Db,buff,dbServer,dbStar);
125     mcurrentDb->use();
126     mdataBases.push_back(mcurrentDb);
127   }
128
129  if(Db.IsConnected() ||
130     Db.Connect(mhostName,muserName,mpword,mcurrentDb->printDbName(),mportNumber)) return mcurrentDb;
131
132  return (StDataBaseI*) NULL;
133 }
```

4.33.2.12 StDataBaseI * StDbServerImpl::useDb (const char * *typeName*, const char * *domName*) [virtual]

Implements StDbServer.

Definition at line 100 of file StDbServerImpl.cc.

```
100                                                         {
101
102 if( mcurrentDb &&
103     mcurrentDb->checkDbType(type) &&
```

```
104     mcurrentDb->checkDbDomain(domain)) return mcurrentDb;
105
106   StDataBaseI* db=0;
107   for(DbList::iterator itr = mdataBases.begin();
108       itr != mdataBases.end(); ++itr){
109     if((*itr)->checkDbType(type) && (*itr)->checkDbDomain(domain)){
110        mcurrentDb=*itr;
111        mcurrentDb->use();
112        db=mcurrentDb;
113        break;
114     }
115   }
116   return db;
117 }
```

4.33.2.13 StDataBaseI * StDbServerImpl::useDb (StDbType *type*, StDbDomain *domain*) [virtual]

Implements StDbServer.

Definition at line 79 of file StDbServerImpl.cc.

```
79                                                          {
80
81 if(mcurrentDb &&
82    mcurrentDb->getDbType()==type &&
83    mcurrentDb->getDbDomain()==domain) return mcurrentDb;
84
85   StDataBaseI* db=0; // return null if not found
86   for(DbList::iterator itr = mdataBases.begin();
87       itr != mdataBases.end(); ++itr){
88     if(type==(*itr)->getDbType() && domain==(*itr)->getDbDomain()){
89        mcurrentDb=*itr;
90        mcurrentDb->use();
91        db=mcurrentDb;
92        break;
93     }
94   }
95   return db;
96 }
```

4.33.3 Member Data Documentation

4.33.3.1 StDbBuffer StDbServerImpl::buff [protected]

Definition at line 62 of file StDbServerImpl.hh.

4.33.3.2 MysqlDb StDbServerImpl::Db [protected]

Definition at line 61 of file StDbServerImpl.hh.

4.33.3.3 StDataBaseI* StDbServerImpl::mcurrentDb [protected]

Definition at line 64 of file StDbServerImpl.hh.

4.33.3.4 DbList StDbServerImpl::mdataBases [protected]

Definition at line 65 of file StDbServerImpl.hh.

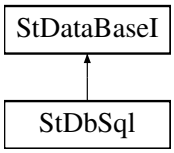
The documentation for this class was generated from the following files:

- StDbServerImpl.hh
- StDbServerImpl.cc

4.34 StDbSql Class Reference

`#include <StDbSql.hh>`

Inheritance diagram for StDbSql::



Public Member Functions

- StDbSql (MysqlDb &db, StDbBuffer &buffer)
- StDbSql (MysqlDb &db, StDbBuffer &buffer, StDbType type, StDbDomain domain)
- StDbSql (MysqlDb &db, StDbBuffer &buffer, const char *typeName, const char *domainName)
- virtual ~StDbSql ()
- virtual void use ()
- virtual void close ()
- virtual void clear ()
- virtual int QueryDb (StDbConfigNode *node)
- virtual int QueryDb (StDbNode *node)
- virtual int QueryDb (StDbTable *table, unsigned int reqTime)
- virtual int QueryDb (StDbTable *table, const char *whereClause)
- virtual unsigned int * QueryDbTimes (StDbTable *table, const char *whereClause, int opt=0)
- virtual int QueryDbFunction (StDbTable *table, const char *whereClause, char *funcName)
- virtual int QueryDescriptor (StDbTable *table)
- virtual int WriteDb (StDbTable *table, unsigned int storeTime)
- virtual int WriteDb (StDbConfigNode *node, int parentID, int &configID)
- virtual int storeConfigNode (StDbConfigNode *node)
- virtual int storeTableNode (StDbTable *table)
- virtual bool insertNodeRelation (int configID, int parent, int child)
- virtual bool rollBack (StDbNode *node)
- virtual bool rollBack (StDbTable *table)
- virtual unsigned int getUnixTime (const char *time)
- virtual char * getDateTime (unsigned int time)
- virtual int * selectElements (const char *elementName, StDbElementIndex *inval, int &numElements)
- char * getDataTable (StDbTable *table, unsigned int time)
- char ** getDataTables (StDbTable *table, int &numTables)
- void setDbUtils (MysqlDb &db, StDbBuffer &buffer)
- void setDefaultReturnValues (StDbTable *table, unsigned int reqTime)
- void setDefaultBeginTime (StDbTable *table, unsigned int reqTime)
- void setDefaultEndTime (StDbTable *table)
- char ** getIndexNames (const char *elementName, int &numIndexes)

Public Attributes

- MysqlDb & Db
- StDbBuffer & buff

Protected Member Functions

- StDbTableDescriptor * findDescriptor (int structID, int schemaID)
- StDbTableDescriptor * getDescriptor (int structID, int schemaID)
- void addDescriptor (StDbTableDescriptor *td)
- void deleteDescriptors ()
- void deleteDescriptor (int structID, int schemaID)
- int prepareNode (StDbNode *node)
- int queryNode (StDbNode *node)
- bool readNodeInfo (StDbNode *node)
- bool readConfigNodeInfo (StDbConfigNode *node)
- bool readTableInfo (StDbTable *table)
- bool checkValue (const char *colName, const char *colValue)
- bool checkForNull (const char *src)
- char * insertNodeString (StDbNode *node)
- char * getFlavorQuery (const char *flavor)
- char * getProdTimeQuery (unsigned int prodTime)
- char * getElementList (int *elements, int num)
- char * getElementListIN (int *elements, int num)
- char * getColumnList (StDbTable *table, char *tableName=0, char *funcName=0)
- char * getEmptyString ()
- bool hasInstance (StDbTable *table)
- void checkTableCatalog ()
- char * checkTablePrepForQuery (StDbTable *table, bool checkIndexed=false)
- void deleteRows (const char *tableName, int *rowID, int nrows)
- void initEndTime ()
- char * mRetString (StString &rs)
- int sendMess (const char *a, const char *b, StDbMessLevel m, int lineNum=0, const char *class-Name=" ", const char *methName=" ")
- bool checkColumn (const char *tableName, const char *columnName)
- bool updateEndTime (StDbTable *table, const char *dataTable, unsigned int reqTime)
- void init ()
- virtual int WriteDbNoIndex (StDbTable *table, unsigned int storeTime)

Protected Attributes

- StDbManager * mgr
- DescList mdescriptors
- char * mretString
- int mtableCatalog
- char * mdefaultEndDateTime
- unsigned int mdefaultEndTime

4.34.1 Constructor & Destructor Documentation

4.34.1.1 StDbSql::StDbSql (MysqlDb & *db*, StDbBuffer & *buffer*)

Definition at line 148 of file StDbSql.cc.

```
148 : StDataBaseI(),Db(db), buff(buffer) { mgr=StDbManager::Instance(); init(); }
```

4.34.1.2 StDbSql::StDbSql (MysqlDb & *db*, StDbBuffer & *buffer*, StDbType *type*, StDbDomain *domain*)

Definition at line 152 of file StDbSql.cc.

```
152                                                                    : StDataBaseI(tpe, dom),
153 mgr=StDbManager::Instance(); init();
154 }
```

4.34.1.3 StDbSql::StDbSql (MysqlDb & *db*, StDbBuffer & *buffer*, const char * *typeName*, const char * *domainName*)

Definition at line 158 of file StDbSql.cc.

```
158                                                                    : StDataBaseI(tpe, d
159   mgr=StDbManager::Instance(); init();
160 }
```

4.34.1.4 StDbSql::~StDbSql () [virtual]

Definition at line 164 of file StDbSql.cc.

```
164                    {
165   if(mretString) delete [] mretString;
166   if(mdefaultEndDateTime) delete [] mdefaultEndDateTime;
167   deleteDescriptors();
168 }
```

4.34.2 Member Function Documentation

4.34.2.1 void StDbSql::addDescriptor (StDbTableDescriptor * *td*) [protected]

Definition at line 1068 of file StDbSql.cc.

```
1068                                                {
1069   mdescriptors.push_back( new StDbTableDescriptor((*td)) ); // add a copy
1070 };
```

4.34.2.2 bool StDbSql::checkColumn (const char * *tableName*, const char * *columnName*) [protected]

Definition at line 442 of file StDbSql.cc.

```
442                                                                    {
443
444   bool retVal=false;
445   Db<<"show columns from "<<tableName<<" like '"<<columnName<<"'"<<endsql;
446   if(Db.NbRows()==1)retVal=true;
447   Db.Release();
448   return retVal;
449 }
```

4.34.2.3 bool StDbSql::checkForNull (const char * *src*) [protected]

Definition at line 1523 of file StDbSql.cc.

```
1523                                    {
1524   return (strstr(src,"null")) ? true : false;
1525 }
```

4.34.2.4 void StDbSql::checkTableCatalog () [protected]

Definition at line 1614 of file StDbSql.cc.

```
1614                           {
1615
1616   //  if(Db.checkForTable("tableCatalog"))mtableCatalog=1
1617   Db<<"show tables like 'tableCatalog'"<<endsql;
1618   if(Db.NbRows()==0){
1619     mtableCatalog=1;
1620   } else {
1621     mtableCatalog=2;
1622   }
1623   Db.Release();
1624 }
```

4.34.2.5 char * StDbSql::checkTablePrepForQuery (StDbTable * *table*, bool *checkIndexed* = false) [protected]

Definition at line 1501 of file StDbSql.cc.

```
1501                                                                    {
1502
1503   if(mretString){
1504      delete [] mretString;
1505      mretString=new char[256];
1506   }else{mretString=new char[256];}
1507
1508   if(!QueryDescriptor(table))
1509      return strcpy(mretString," doesn't have a descriptor ");
1510
1511   if(table->IsBinary())
1512     return strcpy(mretString," Binary Store is currently disabled ");
1513
1514   if(checkIndexed && !table->IsIndexed())
1515     return strcpy(mretString," Table is not time indexed");
1516
1517   delete [] mretString;
1518   return mretString=0;
1519 }
```

4.34.2.6 bool StDbSql::checkValue (const char * *colName*, const char * *colValue*) [protected]

Definition at line 1366 of file StDbSql.cc.

```
1366                                                          {
1367
1368  bool retVal = false;
```

```
1369  char* tmpS=0;
1370  if(buff.ReadScalar(tmpS,colName) && strcmp(tmpS,colValue)==0)retVal=true;
1371  if(tmpS) delete [] tmpS;
1372
1373 return retVal;
1374 }
```

4.34.2.7 void StDbSql::clear () [inline, virtual]

Definition at line 222 of file StDbSql.hh.

```
222 { Db.Release(); buff.Raz(); };
```

4.34.2.8 void StDbSql::close () [inline, virtual]

Implements StDataBaseI.

Definition at line 221 of file StDbSql.hh.

```
221 { Db.Close(); };
```

4.34.2.9 void StDbSql::deleteDescriptor (int *structID*, int *schemaID*) [protected]

Definition at line 1091 of file StDbSql.cc.

```
1091                                                    {
1092
1093   StDbTableDescriptor* desc;
1094   for(DescList::iterator itr = mdescriptors.begin();
1095       itr != mdescriptors.end(); ++itr){
1096       if(structID==(*itr)->getStructID() && schemaID==(*itr)->getSchemaID()){
1097         desc=(*itr);
1098         mdescriptors.erase(itr);
1099         delete [] desc;
1100         break;
1101       }
1102   }
1103 }
```

4.34.2.10 void StDbSql::deleteDescriptors () [protected]

Definition at line 1074 of file StDbSql.cc.

```
1074                          {
1075 DescList::iterator itr;
1076 StDbTableDescriptor* desc;
1077
1078   do {
1079      for(itr=mdescriptors.begin(); itr != mdescriptors.end(); ++itr){
1080          desc=*itr;
1081          mdescriptors.erase(itr);
1082          if(desc)delete desc;
1083          break;
1084      }
1085   } while (mdescriptors.begin() != mdescriptors.end() );
1086
1087 }
```

4.34.2.11 void StDbSql::deleteRows (const char * *tableName*, int * *rowID*, int *nrows*) [protected]

Definition at line 945 of file StDbSql.cc.

```
945                                                                 {
946
947   if(!rowID || nrows==0)return;
948   Db<<" delete from "<<tableName;
949   Db<<" where dataID In("<<getElementList(rowID,nrows)<<")"<<endsql;
950 }
```

4.34.2.12 StDbTableDescriptor * StDbSql::findDescriptor (int *structID*, int *schemaID*) [protected]

Definition at line 1044 of file StDbSql.cc.

```
1044                                                 {
1045
1046  StDbTableDescriptor* td = 0;
1047  for(DescList::iterator itr = mdescriptors.begin();
1048      itr != mdescriptors.end(); ++itr){
1049    if( ((*itr)->getSchemaID()==schemaID) && ((*itr)->getStructID()==structID)){
1050      td = *itr;
1051      break;
1052    }
1053  }
1054  return td;
1055 }
```

4.34.2.13 char * StDbSql::getColumnList (StDbTable * *table*, char * *tableName* = 0, char * *funcName* = 0) [protected]

Definition at line 1466 of file StDbSql.cc.

```
1466                                                                    {
1467
1468   StTableDescriptorI* desc=table->getDescriptor();
1469   int numElements=desc->getNumElements();
1470
1471   StString es; es<<" ";
1472
1473   int icount=0;
1474   for(int i=0;i<numElements;i++){
1475      if(funcName && (desc->getElementLength(i)>1))continue;
1476      char* name=desc->getElementName(i);
1477      if(funcName)es<<funcName<<"(";
1478      if(tableName)es<<tableName<<".";
1479      es<<name;
1480      if(funcName)es<<") as "<<name;
1481      if(i<(numElements-1))es<<", ";
1482      delete [] name;
1483      icount++;
1484   }
1485
1486  if(icount) return mRetString(es);
1487  return (char*) 0;
1488 }
```

4.34.2.14 char * StDbSql::getDataTable (StDbTable * *table*, unsigned int *time*)

Definition at line 1529 of file StDbSql.cc.

```
1529                                                        {
1530
1531   if(mtableCatalog==0) checkTableCatalog();
1532   if(mtableCatalog==1) return table->getCstructName();
1533   char* retVal=0;
1534   clear();
1535
1536 // as of now, just use name... later maybe more so I've sent in StDbTable*
1537 char* tableName=table->printName();
1538
1539 Db<<" select * from tableCatalog where nodeName='"<<tableName<<"'";
1540 Db<<" AND unix_timestamp(beginTime)<="<<time;
1541 Db<<" Order by beginTime desc limit 1"<<endsql;
1542
1543 if(Db.Output(&buff))buff.ReadScalar(retVal,"tableName");
1544 clear();
1545
1546 return (retVal) ? retVal : table->getCstructName();
1547 }
```

4.34.2.15 char ** StDbSql::getDataTables (StDbTable * *table*, int & *numTables*)

Definition at line 1551 of file StDbSql.cc.

```
1551                                                        {
1552
1553  char** retVal;
1554  if(mtableCatalog==0) checkTableCatalog();
1555  if(mtableCatalog==1){
1556     numTables=1;
1557     retVal=new char*[1];
1558     retVal[0]=table->getCstructName();
1559     return retVal;
1560  }
1561  clear();
1562
1563 // as of now, just use name... later maybe more so I've sent in StDbTable*
1564  char* tableName=table->printName();
1565  Db<<" select * from tableCatalog where nodeName='"<<tableName<<"'"<<endsql;
1566
1567  if(!((numTables)=Db.NbRows()) ){ // use tableName as the 1
1568      numTables++;
1569      retVal= new char*[numTables];
1570      retVal[0]=new char[strlen(tableName)+1];
1571      strcpy(retVal[0],tableName);
1572      return retVal;
1573  }
1574
1575  retVal=new char*[numTables];
1576  int i=0;
1577  while(Db.Output(&buff)) if(buff.ReadScalar(retVal[i],"tableName"))i++;
1578
1579  numTables=i;
1580 return retVal;
1581 }
```

4.34.2.16 char * StDbSql::getDateTime (unsigned int *time*) [virtual]

Implements StDataBaseI.

Definition at line 1350 of file StDbSql.cc.

```
1350                                    {
1351
1352   char* retVal=0;
1353 // note the " + 0" part formats result without delimiters
1354 // e.g. 1999-01-01 00:12:20 becomes 19990101001220
1355
1356   clear();
1357   Db<<"select from_unixtime("<<time<<") + 0 as requestTime"<<endsql;
1358   if(Db.Output(&buff)) buff.ReadScalar(retVal,"requestTime");
1359   clear();
1360
1361 return retVal;
1362 }
```

4.34.2.17 StDbTableDescriptor * StDbSql::getDescriptor (int *structID*, int *schemaID*) [protected]

Definition at line 1059 of file StDbSql.cc.

```
1059                                                {
1060   StDbTableDescriptor* retVal=findDescriptor(structID,schemaID);
1061   if(retVal) return new StDbTableDescriptor(*retVal); // make copy
1062   retVal=new StDbTableDescriptor(structID,schemaID); // -else- make new
1063   return retVal;
1064 }
```

4.34.2.18 char * StDbSql::getElementList (int * *elements*, int *num*) [protected]

Definition at line 1419 of file StDbSql.cc.

```
1419                                        {
1420
1421 // prepares comma separated list of integers
1422 // using the between operator
1423
1424  StString es;
1425  if(!e){
1426    es<<0;
1427  } else {
1428    // for(int i=0;i<num-1;i++)//es<<e[i]<<",";
1429      {
1430 //MPD
1431        // if (e[0] == 0){
1432        if (num == 1){
1433          es<<"= "<<e[0];
1434        } else if(e[0]< num) {
1435          es<<"between "<<e[0]<<" and "<<num;
1436        } else {
1437          es<<"between "<<num <<" and "<<e[0];
1438        }
1439      }
1440      // es<<e[num-1];
1441  }
```

```

 return mRetString(es);
}
```

4.34.2.19 char * StDbSql::getElementListIN (int * *elements*, int *num*) [protected]

Definition at line 1447 of file StDbSql.cc.

```
                                        {
// prepares comma separated list of integers
// using the IN operator


 StString es;
 if(!e){
   es<<0;
 } else {
   es<<"In (";
   for(int i=0;i<num-1;i++) es<<e[i]<<",";
   es<<e[num-1];
 }
 es<<")";
 return mRetString(es);
}
```

4.34.2.20 char * StDbSql::getEmptyString () [protected]

Definition at line 1492 of file StDbSql.cc.

```
                       {

  StString es; es<<" ";
  return mRetString(es);

};
```

4.34.2.21 char * StDbSql::getFlavorQuery (const char *flavor*) [protected]

Definition at line 1378 of file StDbSql.cc.

```
                                         {

// prepares SQL of " flavor In('flav1','flav2',...)"

 char *id1,*id2,*id3;
 id1 = new char[strlen(flavor)+1];
 strcpy(id1,flavor);
 id3=id1;

 StString fs;
 fs<<" flavor In(";
 while(( (id2)=strstr(id3,"+")) ){
    *id2='\0';
    fs<<"'"<<id3<<"',";
    *id2='+';
    id2++;
    id3=id2;
```

```
1395  }
1396  fs<<"'"<<id3<<"')";
1397  delete [] id1;
1398
1399  return mRetString(fs);
1400 }
```

4.34.2.22 char ** StDbSql::getIndexNames (const char * *elementName*, int & *numIndexes*)

Definition at line 1022 of file StDbSql.cc.

```
1022                                                                    {
1023
1024   Db<<"select * from elementIndexes ";
1025   Db<<" where elementName='"<<elementName<<"'"<<endsql;
1026
1027   char** indexNames = 0;
1028   if(!((numIndexes)=Db.NbRows())) return indexNames;
1029
1030   indexNames = new char*[numIndexes];
1031   int i = 0;
1032   while(Db.Output(&buff)){
1033       buff.ReadScalar(indexNames[i],"indexName");
1034       i++;
1035       buff.Raz();
1036   }
1037
1038  clear();
1039  return indexNames;
1040 }
```

4.34.2.23 char * StDbSql::getProdTimeQuery (unsigned int *prodTime*) [protected]

Definition at line 1404 of file StDbSql.cc.

```
1404                                                  {
1405
1406 // prepares SQL of " entryTime<="
1407 StString pt;
1408   if(prodTime==0){
1409     pt<<" deactive=0 ";
1410   } else {
1411     pt<<" (deactive=0 OR deactive>="<<prodTime<<")";
1412     pt<<" AND unix_timestamp(entryTime)<="<<prodTime;
1413   }
1414   return mRetString(pt);
1415 }
```

4.34.2.24 unsigned int StDbSql::getUnixTime (const char * *time*) [virtual]

Implements StDataBaseI.

Definition at line 1337 of file StDbSql.cc.

```
1337                                      {
1338
1339   unsigned int retVal = 0;
```

```
  clear();
  Db<<"select unix_timestamp('"<<time<<"') as requestTime"<<endsql;
  if(Db.Output(&buff)) buff.ReadScalar(retVal,"requestTime");
  clear();

return retVal;
}
```

4.34.2.25 bool StDbSql::hasInstance (StDbTable * *table*) [protected]

Definition at line 1585 of file StDbSql.cc.

```
                                    {

  // try and find a data instance of this table
  // for use in keeping track of baseline instances
 bool retVal=false;
 clear();

 int numTables;
 char** dataTables=getDataTables(table,numTables);
 if(numTables==0) return retVal;

 for(int i=0; i<numTables;i++){

   Db<<" select * from "<<dataTables[i];
   Db<<" where "<<getFlavorQuery(table->printFlavor());
   Db<<" AND "<<getProdTimeQuery(table->getProdTime())<<endsql;

   if(Db.NbRows() !=0 ){
     retVal=true;
     break;
   }
 }
 for(int k=0;k<numTables;k++) delete [] dataTables[k];
 delete [] dataTables;
 return retVal;
}
```

4.34.2.26 void StDbSql::init () [inline, protected]

Definition at line 214 of file StDbSql.hh.

```
                          {
    mretString = 0;
    mtableCatalog=0;
    mdefaultEndDateTime = 0;
}
```

4.34.2.27 void StDbSql::initEndTime () [protected]

Definition at line 1131 of file StDbSql.cc.

```
                          {
  mdefaultEndTime=StDbDefaults::Instance()->getEndTime();
  mdefaultEndDateTime=getDateTime(mdefaultEndTime);
}
```

4.34.2.28 bool StDbSql::insertNodeRelation (int *configID*, int *parent*, int *child*) [virtual]

Implements StDataBaseI.

Definition at line 1324 of file StDbSql.cc.

```
1324                                                                {

1326   Db<<" insert into NodeRelation set ParentID="<<parent;
1327   Db<<", NodeID="<<child<<", ConfigID="<<configID<<endsql;

1329   bool retVal=Db.QueryStatus();
1330   Db.Release();

1332   return retVal;
1333 }
```

4.34.2.29 char * StDbSql::insertNodeString (StDbNode * *node*) [protected]

Definition at line 1265 of file StDbSql.cc.

```
1265                                        {

1267   StString dqs;
1268   dqs<<"insert into Nodes set name='"<<node->printName()<<"' ";
1269   if(!StDbDefaults::Instance()->IsDefaultVersion(node->printVersion()))
1270      dqs<<", versionKey='"<<node->printVersion()<<"' ";

1272   dqs<<", nodeType='"<<node->printNodeType()<<"'";

1274   return mRetString(dqs);
1275 }
```

4.34.2.30 char * StDbSql::mRetString (StString & *rs*) [inline, protected]

Definition at line 201 of file StDbSql.hh.

```
201                                             {
202    if(mretString)delete [] mretString;
203    string srs=rs.str();
204    mretString = new char[srs.length()+1];
205    strcpy(mretString,srs.c_str());
206    return mretString;
207 }
```

4.34.2.31 int StDbSql::prepareNode (StDbNode * *node*) [protected]

Definition at line 1138 of file StDbSql.cc.

```
1138                                      {

1140   //-> takes the input dbNode's name & version and calls the db
1141   //    (via method queryNode()) for the other information
1142   //    and sets into dbNode

1144   if(  (strcmp(dbNode->printNodeType(),"DB")!=0) &&
```

```
        (dbNode->IsConfigured())) return dbNode->getNodeID();

  dbNode->setDbName(mdbName);
  dbNode->setDbType(mdbType);
  dbNode->setDbDomain(mdbDomain);

  return queryNode(dbNode);;
}
```

4.34.2.32 int StDbSql::QueryDb (StDbTable * *table*, const char * *whereClause*) [virtual]

Implements StDataBaseI.

Definition at line 484 of file StDbSql.cc.

```
                                                        {

#define __METHOD__ "QueryDb(StDbTable*, const char* where)"

  unsigned int* timeSet=QueryDbTimes(table,whereClause);
  if(!timeSet)return 0;
  delete [] timeSet;
  return table->GetNRows();
#undef __METHOD__
}
```

4.34.2.33 int StDbSql::QueryDb (StDbTable * *table*, unsigned int *reqTime*) [virtual]

Implements StDataBaseI.

Definition at line 255 of file StDbSql.cc.

```
                                                        {

#define __METHOD__ "QueryDb(table,time)"

  //  Using name & version information in StDbTable + request timestamp
  //  fill data from the database into the StDbTable pointer.
  //
  int retVal=1;
  if(!table) return 0;
  char* tName = table->printName();
  table->clearStoreInfo();
  int nodeID;
  if(!((nodeID)=table->getNodeID())){
    if(!(prepareNode((StDbNode*)table)))
    return sendMess(tName," Table not found in DB",dbMErr,__LINE__,__CLASS__,__METHOD__);

    readTableInfo(table);
    clear();
  }

  char* checkString=checkTablePrepForQuery(table,true); // returns null if ok
  if(checkString)
     return sendMess(tName,checkString,dbMErr,__LINE__,__CLASS__,__METHOD__);

 // start preparing the queries

 // common where clause
   StString bs;
   bs<<" Where nodeID="<<nodeID;
```

```
 // prepare "flavor" part of query
   bs<<" AND "<<getFlavorQuery(table->printFlavor());
 // prepare for production time
   bs<<" AND "<<getProdTimeQuery(table->getProdTime());
 // terminate the baseString
   bs<<" ";

   string baseString = bs.str();


 //--> add element ID list part of query
  int numRows;
  int* elementID=table->getElementID(numRows);
  if(!elementID)
    return sendMess(tName,"doesn't have an Element List",dbMErr,__LINE__,__CLASS__,__METHOD__);

  char* elementString=getElementList(elementID,numRows);
  char* dataTable=getDataTable(table,reqTime);

// Query DB for the endtime -> earliest time of next row
  Db << " select beginTime + 0 as mendDateTime, ";
  Db << " unix_timestamp(beginTime) as mendTime from "<<dataTable;
  Db << baseString <<" AND beginTime>from_unixtime("<<reqTime<<")";
//  Db << " And elementID In("<<elementString<<")";
  Db << " And elementID "<<elementString;
  Db << " Order by beginTime limit 1"<<endsql;

  sendMess(DbQInfo,Db.printQuery(),dbMDebug,__LINE__,__CLASS__,__METHOD__);

  if(!Db.QueryStatus())
     return sendMess(DbQFailed,Db.printQuery(),dbMWarn,__LINE__,__CLASS__,__METHOD__);

  if(Db.NbRows()==1 && Db.Output(&buff)){
    char* edTime=0;
    int eTime;
    buff.ReadScalar(edTime,"mendDateTime");
    buff.ReadScalar(eTime,"mendTime");
    table->setEndTime(eTime);
    table->setEndTime(edTime);
    if(edTime) delete [] edTime;
    clear();
  } else {
    setDefaultEndTime(table);
  }

  int eID;

 // --> prep for data query which can be 1 or more queries
   int maxID=1;
   int i;
   for(i=0;i<numRows;i++)if(elementID[i]>maxID)maxID=elementID[i];
   int* idMap = new int[maxID+1];
   int* dataIDList = new int[numRows];
   unsigned int* timeValues = new unsigned int[numRows];
   for(i=0;i<numRows;i++){
        idMap[elementID[i]]=i;
        dataIDList[i]=0;
        timeValues[i]=0;
   }

// --> end of prep <--
  // query is done where elementID is not necessarily distinct.
  // yet the in the query we set limit=numRows.
  //  - we take 1st of instance of each elementID returned.
  //  - we make a new list of those not yet found
  //  - redo query with limit="rowsLeft"
  // continue until we're done
```

```

  int rowsLeft=numRows;
  bool done=false;
  while(!done){  //done if all elementIDs are found or (break) if null

   Db <<" select unix_timestamp(beginTime) as bTime,"<<dataTable<<".* from ";
   Db << dataTable << baseString;
   Db <<" AND beginTime<=from_unixtime("<<reqTime<<")";
   //Db <<" AND elementID In("<<elementString<<") ";
   Db <<" AND elementID "<<elementString;
   Db <<" Order by beginTime desc limit "<< rowsLeft <<endsql;

   sendMess(DbQInfo,Db.printQuery(),dbMDebug,__LINE__,__CLASS__,__METHOD__);

  if(!Db.QueryStatus()){
    sendMess(DbQFailed,Db.printQuery(),dbMWarn,__LINE__,__CLASS__,__METHOD__);
    break;
  }

   if(Db.NbRows()==0)break;

   int numRowsFound=0;
   while(Db.Output(&buff)){
      buff.ReadScalar(eID,"elementID");
    // check to see if this eID is already returned & don't use if so
      if(!dataIDList[idMap[eID]]){
        buff.ReadScalar(dataIDList[idMap[eID]],"dataID");
        buff.ReadScalar(timeValues[idMap[eID]],"bTime");
        table->setRowNumber(idMap[eID]);
        table->dbStreamer(&buff,true); // stream data into table
        numRowsFound++;
      }
      buff.Raz();
   }

   rowsLeft=rowsLeft-numRowsFound;
 // Check for left overs & modify query for repeat
   if(!done && (rowsLeft>0)){
     int* elementsLeft = new int[rowsLeft];
     int j=0;
     for(i=0;i<numRows;i++){
       if(!dataIDList[i]){
         elementsLeft[j]=elementID[i];
         j++;
       }
     }
     elementString = getElementListIN(elementsLeft,rowsLeft);
     delete [] elementsLeft;
   } else {
     done=true;
   }

  } // --> end of while loop

  if(rowsLeft==numRows){
     sendMess(tName," has No data for query",dbMWarn,__LINE__,__CLASS__,__METHOD__);
     setDefaultBeginTime(table,reqTime);
     retVal=0;
   } else if(rowsLeft>0){
     StString tp;
     tp<<" Not all rows filled from DB, Requested="<<numRows;
     tp<<" Returned="<<numRows-rowsLeft<<" for Table="<<tName;
     mgr->printInfo((tp.str()).c_str(),dbMWarn,__LINE__,__CLASS__,__METHOD__);
     //     numRows-=rowsLeft;
     //     table->resizeNumRows(numRows);
   }

```

```
418   if(retVal){
419     table->addWrittenRows(dataIDList,numRows);
420     table->setTimeValues(timeValues);
421     unsigned int t1=table->getMaxTime();
422     table->setBeginTime(t1);
423     char* dt=getDateTime(t1);
424     table->setBeginTime(dt); delete [] dt;
425     table->setRowNumber(); // reset current row to 0
426   } else {
427                 delete [] timeValues;
428         }
429
430   Db.Release();
431    if(retVal) retVal=(int)updateEndTime(table,dataTable,reqTime);
432
433   delete [] idMap;
434   delete [] dataIDList;
435   delete [] dataTable;
436
437  return retVal;
438 #undef __METHOD__
439 }
```

4.34.2.34 int StDbSql::QueryDb (StDbNode * *node*) [virtual]

Implements StDataBaseI.

Definition at line 239 of file StDbSql.cc.

```
239                                {
240 #define __METHOD__ "QueryDb(StDbNode*)"
241
242   if(!prepareNode(node))  return 0;
243   if(node->IsTable()){
244     readTableInfo((StDbTable*) node);
245   } else {
246     readConfigNodeInfo((StDbConfigNode*)node);
247   }
248
249 return node->getNodeID();
250 #undef __METHOD__
251 }
```

4.34.2.35 int StDbSql::QueryDb (StDbConfigNode * *node*) [virtual]

Implements StDataBaseI.

Definition at line 177 of file StDbSql.cc.

```
177                                        {
178
179 #define __METHOD__ "QueryDb(StDbConfigNode*)"
180
181  StDbNode curNode;
182  curNode.setDbName(mdbName);
183  curNode.setDbType(mdbType);
184  curNode.setDbDomain(mdbDomain);
185  int NodeID;
186  int branchID=node->getBranchID();
187
188  if(!((NodeID)=prepareNode(node)))return 0;
```

```

 // Build node query string

 Db<<"Select subNode.*, NodeRelation.ID as branchID from Nodes ";
 Db<<"LEFT JOIN NodeRelation ON Nodes.ID=NodeRelation.ParentID ";
 Db<<"LEFT JOIN Nodes as subNode ON NodeRelation.NodeID=subNode.ID ";
 Db<<" Where Nodes.ID="<<NodeID;
 Db<<" and NodeRelation.BranchID="<<branchID<<endsql;

 if(!Db.QueryStatus())
    return sendMess(DbQFailed,Db.printQuery(),dbMWarn,__LINE__,__CLASS__,__METHOD__);

 if(!Db.NbRows())
    sendMess(node->printName()," Node has no subnodes",dbMDebug,__LINE__,__CLASS__,__METHOD__);

 sendMess(DbQInfo,Db.printQuery(),dbMDebug,__LINE__,__CLASS__,__METHOD__);

//
// Loop over rows in Configuration

  while(Db.Output(&buff)){
    curNode.setConfigured(false);
    if(readNodeInfo(&curNode)){

      StString fs;
      fs<<"Found "<<curNode.printNodeType()<<" Node "<<curNode.printName();
      fs<<" of parent "<<node->printName();
      sendMess(DbQResult,(fs.str()).c_str(),dbMDebug,__LINE__,__CLASS__,__METHOD__);


      if(strcmp(curNode.printNodeType(),"table")!=0){ // it is a ConfigNode

        StDbConfigNode* child= new StDbConfigNodeImpl(node,curNode);
        if(!child->isDbNode())readConfigNodeInfo(child);

      } else {

        StDbTable* table = node->addTable(&curNode);
        readTableInfo(table);
      }
    }
    buff.Raz();
  }
  Db.Release();
  return 1;
#undef __METHOD__
}
```

4.34.2.36 int StDbSql::QueryDbFunction (StDbTable * *table*, const char * *whereClause*, char * *funcName*) [virtual]

Implements StDataBaseI.

Definition at line 628 of file StDbSql.cc.

```
628                                                                                {
629 #define __METHOD__ "QueryDbFunction(table,whereClause,functionName)"
630   /*
631      Method to provide access to Max, Min, Ave,... functions where
632      the result per storage table is put in a row in the StDbTable.
633      This only works for those columns stored in basic Mysql types
634   */
635
636   char* checkString=checkTablePrepForQuery(table); // null is good
```

```
637   if(checkString)
638      return sendMess(table->printName(),checkString,dbMErr,__LINE__,__CLASS__,__METHOD__);
639
640    int numTables;
641    int numRowsReturned=0;
642
643    char** dataTables=getDataTables(table,numTables);
644    int i;
645    for(i=0;i<numTables;i++){
646
647    char* columnList=getColumnList(table,dataTables[i],funcName);
648    if(!columnList)return 0;
649
650      Db<<" select "<<columnList<<" from "<<dataTables[i];
651      Db<<" "<<whereClause<<endsql;
652
653      sendMess(DbQInfo,Db.printQuery(),dbMDebug,__LINE__,__CLASS__,__METHOD__);
654
655      if(!Db.QueryStatus())
656         return sendMess(DbQFailed,Db.printQuery(),dbMWarn,__LINE__,__CLASS__,__METHOD__);
657
658      int retRows=Db.NbRows();
659      if(retRows==0) continue;
660      numRowsReturned+=retRows;
661
662      while(Db.Output(&buff)){ table->dbStreamer(&buff,true); buff.Raz(); }
663      Db.Release();
664    }
665
666    for(i=0;i<numTables;i++) delete [] dataTables[i];
667    delete [] dataTables;
668    table->setRowNumber();
669
670    return numRowsReturned;
671 #undef __METHOD__
672 };
```

4.34.2.37 unsigned int ∗ StDbSql::QueryDbTimes (StDbTable ∗ *table*, const char ∗ *whereClause*, int *opt* = 0) [virtual]

Implements StDataBaseI.

Definition at line 497 of file StDbSql.cc.

```
497                                                                    {
498
499 #define __METHOD__ "QueryDb(StDbTable*, const char* where)"
500
501   /*
502      rules for # of rows returned (by user request via setRowLimit(int nrows);)
503      1. table->GetNRows()= 0 or N means no limit or limit N
504      2. returned table->GetNRows()=M where M is how many returned
505
506      rules for beginTime & endTime
507      1. Non-indexed tables have arbitrary returned times
508      2. beginTime="lastest" begin time of returned rows
509      3. endTime="earliest" begin time of set of "next" rows
510   */
511
512   unsigned int* retVal=0;
513   char* tName=table->printName();
514
515   char* checkString=checkTablePrepForQuery(table);
516   if(checkString){  // then error message
```

```
    sendMess(tName,checkString,dbMErr,__LINE__,__CLASS__,__METHOD__);
    return retVal;
  }

  setDefaultReturnValues(table,0);
  int numRows = table->getRowLimit();
  int numTables;
  int numRowsReturned=0;
  unsigned int t1=0;


  table->setElementID((int*)retVal,0); // no rows to begin with
  table->setRowNumber();

  char** dataTables=getDataTables(table,numTables);
  int i;
  for(i=0;i<numTables;i++){

    char* columnList=0;
    if(!opt)columnList=getColumnList(table,dataTables[i]);
    if(!opt && !columnList){
      sendMess(tName," has no elements?",dbMErr,__LINE__,__CLASS__,__METHOD__);
      return retVal;
    }

     Db<<" select unix_timestamp("<<dataTables[i]<<".beginTime) as bTime,";
     Db<<" elementID ";
     if(!opt)Db<<","<<columnList;
     Db<<" from "<<dataTables[i]<<" "<<whereClause;
     if(numRows)Db<<" limit "<<numRows;
     Db<<endsql;
     sendMess(DbQInfo,Db.printQuery(),dbMDebug,__LINE__,__CLASS__,__METHOD__);
     if(!Db.QueryStatus()){
      sendMess(DbQFailed,Db.printQuery(),dbMWarn,__LINE__,__CLASS__,__METHOD__);
      return retVal;
     }

     int retRows=Db.NbRows();
     if(retRows==0) continue;

     int* elements = new int[retRows];
     int* dataIDList = new int[retRows];
     unsigned int* timeList = new unsigned int[retRows];
     if(!opt){
       table->addNRows(retRows);
     } else {
       table->resizeElementID(retRows+table->GetNRows());
     }
     // table->setRowNumber();

     int j=0;
     while(Db.Output(&buff)){
       buff.ReadScalar(timeList[j],"bTime");
       buff.ReadScalar(elements[j],"elementID");
       buff.ReadScalar(dataIDList[j],"dataID");
       if(!opt)table->dbStreamer(&buff,true);
       if(timeList[j]>t1)t1=timeList[j];
       j++;
       buff.Raz();
     }
     table->addNElements(elements,retRows);
     table->addWrittenRows(dataIDList,retRows);
     table->setBeginTime(t1);

     unsigned int* tmpRet=new unsigned int[numRowsReturned+retRows];
     if(retVal){
       memcpy(tmpRet,retVal,numRowsReturned*sizeof(int));
```

```
584         delete [] retVal;
585       }
586       tmpRet+=numRowsReturned;
587       memcpy(tmpRet,timeList,retRows*sizeof(int));
588       retVal=tmpRet;
589       numRowsReturned+=retRows;
590       Db.Release();
591
592       if(table->IsIndexed() && t1>0 && !opt){
593         Db<<" select unix_timestamp(beginTime) as eTime from "<<dataTables[i];
594         Db<<" where beginTime>from_unixtime("<<t1<<")";
595         //Db<<" and elementID In("<<getElementList(elements,retRows)<<")";
596         Db<<" and elementID "<<getElementList(elements,retRows);
597         Db<<" Order by beginTime limit 1"<<endsql;
598
599         sendMess(DbQInfo,Db.printQuery(),dbMDebug,__LINE__,__CLASS__,__METHOD__);
600
601         if(Db.Output(&buff)){
602           unsigned int eTime=0;
603           if(buff.ReadScalar(eTime,"eTime") && eTime && eTime<table->getEndTime())table->setEndTime(eTi
604           buff.Raz();
605         }
606         Db.Release();
607       }
608       delete [] elements;
609       delete [] dataIDList;
610       delete [] timeList;
611     }// loop over tables
612
613     for(i=0;i<numTables;i++) delete [] dataTables[i];
614     delete [] dataTables;
615     if(retVal){
616         char* dateTime=getDateTime(table->getBeginTime());
617         table->setBeginTime(dateTime); if(dateTime) delete [] dateTime;
618         dateTime=getDateTime(table->getEndTime());
619         table->setEndTime(dateTime); if(dateTime) delete [] dateTime;
620         table->setRowNumber();
621     }
622     return retVal;
623 #undef __METHOD__
624 }
```

4.34.2.38 int StDbSql::QueryDescriptor (StDbTable * *table*) [virtual]

Implements StDataBaseI.

Definition at line 833 of file StDbSql.cc.

```
833                                         {
834 #define __METHOD__ "QueryDescriptor(StDbTable*)"
835
836   // Query the database for the elements associated with this
837   // tableName and with the schemaID that is added at the
838   // constructure or over-written by an input file ... e.g. requested SchemaID
839
840 if(table->hasDescriptor())return 1;
841
842     clear();
843
844     Db<<" SELECT structure.lastSchemaID, structure.ID from structure left join Nodes on structure.name
845     Db<<" WHERE Nodes.name='"<<table->printName() <<"'"<<endsql;
846
847     if(!Db.Output(&buff))return 0;
848
```

```
849     int schemaID;
850     int structID;
851     buff.ReadScalar(schemaID,"lastSchemaID");
852     buff.ReadScalar(structID,"ID");
853     clear();
854
855     int requestSchemaID;
856     if(!(requestSchemaID=table->getSchemaID())){
857       requestSchemaID=schemaID;
858       table->setSchemaID(schemaID);
859     }
860
861     StDbTableDescriptor* descriptor = getDescriptor(structID,requestSchemaID);
862     table->setDescriptor(descriptor);
863
864     if(descriptor->IsValid())return 1;
865
866     Db<<"SELECT  schema.schemaID, schema.name, schema.type, schema.length, ";
867     Db<<"schema.position from schema WHERE schema.structID="<<structID;
868     Db<<" AND schema.schemaID="<<requestSchemaID;
869     Db<<" ORDER by schema.position"<<endsql;
870
871     sendMess(DbQInfo,Db.printQuery(),dbMDebug,__LINE__,__CLASS__,__METHOD__);
872
873     if(!Db.QueryStatus())
874        return sendMess(DbQFailed,Db.printQuery(),dbMWarn,__LINE__,__CLASS__,__METHOD__);
875
876
877     if(Db.NbRows()==0) {
878       deleteDescriptor(structID,requestSchemaID);
879       return 0;
880     }
881
882     while(Db.Output(&buff)){
883       descriptor->fillElement(&buff,requestSchemaID);
884       buff.Raz();
885     }
886     Db.Release();
887     descriptor->endRowPadding();
888
889     addDescriptor(descriptor);
890
891   return 1;
892 #undef __METHOD__
893 }
```

4.34.2.39 int StDbSql::queryNode (StDbNode * *node*) [protected]

Definition at line 1156 of file StDbSql.cc.

```
1156                                {
1157
1158 int retVal=0;
1159
1160     Db<< "Select * from Nodes where Nodes.name='"<<node->printName()<<"'";
1161     Db<<" AND Nodes.versionKey='"<<node->printVersion()<<"'"<<endsql;
1162
1163     if(Db.Output(&buff) && readNodeInfo(node))retVal=node->getNodeID();
1164     if(!retVal) clear();
1165
1166 return retVal;
1167 }
```

4.34.2.40 bool StDbSql::readConfigNodeInfo (StDbConfigNode * *node*) [protected]

Definition at line 1195 of file StDbSql.cc.

```
1195                                                {

1196
1197   char* iname;
1198   if(!buff.ReadScalar(iname,"indexName")) return false;
1199   if(strcmp(iname,"None")!=0){
1200      int id;
1201      buff.ReadScalar(id,"indexVal");
1202      node->setElementIndexInfo(iname,id);
1203   }
1204   delete [] iname;
1205
1206   int branchID;
1207   buff.ReadScalar(branchID,"branchID");
1208   node->setBranchID(branchID);
1209
1210   return true;
1211 }
```

4.34.2.41 bool StDbSql::readNodeInfo (StDbNode * *node*) [protected]

Definition at line 1171 of file StDbSql.cc.

```
1171                                    {

1172
1173   int nodeID;
1174   char* tmpString;
1175
1176   if(!buff.ReadScalar(nodeID,"ID")) return false;
1177   if(!buff.ReadScalar(tmpString,"nodeType"))return false;
1178   node->setNodeType(tmpString);
1179   delete [] tmpString;
1180
1181   if(!buff.ReadScalar(tmpString,"name"))return false;
1182   node->setName(tmpString); delete [] tmpString;
1183
1184   if(!buff.ReadScalar(tmpString,"versionKey"))return false;
1185   node->setVersion(tmpString); delete [] tmpString;
1186
1187   node->setNodeID(nodeID);
1188   node->setConfigured(true);
1189
1190   return true;
1191 }
```

4.34.2.42 bool StDbSql::readTableInfo (StDbTable * *table*) [protected]

Definition at line 1215 of file StDbSql.cc.

```
1215                                        {

1216
1217 char* tmpString;
1218
1219     if(!buff.ReadScalar(tmpString,"structName"))return false;
1220     table->setCstructName(tmpString); delete [] tmpString;
1221
```

```
    tmpString=0;
    char* hy=0;
    if(buff.ReadScalar(tmpString,"elementID")){
      hy=strstr(tmpString,"-");
      if(hy){
        *hy='\0';
        hy++;
        int first=atoi(tmpString);
        int last = atoi(hy);
        hy--;
        *hy='-';
        int len = last-first+1;
        int * tmpElements = new int[len];
        int j=0;
        int k;
        for(k=first; k<=last;k++){
          tmpElements[j]=k;
          j++;
        }
        table->setElementID(tmpElements,len);
        delete [] tmpElements;
      } else {
       delete [] tmpString;
       if(!buff.ReadScalar(tmpString,"indexName")) return false;
       table->setElementName(tmpString);
      }
      delete [] tmpString;
    }


    table->setBaseLine(checkValue("baseLine","Y"));
    table->setBinary(checkValue("isBinary","Y"));
    table->setIndexed(checkValue("isIndexed","Y"));

    table->setDbName(mdbName);
    table->setDbType(mdbType);
    table->setDbDomain(mdbDomain);

return true;
};
```

4.34.2.43 bool StDbSql::rollBack (StDbTable * *table*) [virtual]

Implements StDataBaseI.

Definition at line 963 of file StDbSql.cc.

```
                                  {

  int numRows;
  int* numWrittenRows = table->getWrittenRows(numRows);
  char* dataTable = table->getDataTable();
  char* elementList = getElementList(numWrittenRows,numRows);

  Db<<"delete from "<<dataTable<<" where dataID In("<<elementList<<")"<<endsql;
  Db.Release();

  bool retVal=Db.QueryStatus();
  Db.Release();
  delete [] dataTable;

  return retVal;
}
```

4.34.2.44 bool StDbSql::rollBack (StDbNode * *node*) [virtual]

Implements StDataBaseI.

Definition at line 954 of file StDbSql.cc.

```
954                                {
955
956   if(!(node->canRollBack()) || !(node->getNodeID())) return false;
957   Db<<"delete from Nodes where ID="<<node->getNodeID()<<endsql;
958   return Db.QueryStatus();
959 }
```

4.34.2.45 int * StDbSql::selectElements (const char * *elementName*, StDbElementIndex * *inval*, int & *numElements*) [virtual]

Implements StDataBaseI.

Definition at line 982 of file StDbSql.cc.

```
982                                                                                                    {
983   // if table does not have element rows (elementName=="None")
984   // then will return default elementID=0, numElements=1
985   // if table does not have elements based on StDbElementIndex,
986   // then will return null pointer & numElements=0 --> NO DATA CAN BE Gotten
987   // else, will return element list & numElements based on query
988
989   int* retElements = 0;
990   numElements=1;
991   if(!elementName) return retElements;
992   if(strcmp(elementName,"None")==0){
993       retElements = new int[1]; retElements[0]=0;
994       return retElements;
995   }
996
997   int numIndeces = inval->getNumIndeces();
998   clear();
999
1000   Db<<" select elementID from "<<elementName<<"IDs";
1001   if(numIndeces>0){
1002     Db<<" where "<<inval->printIndexName(0)<<"="<<inval->getIndexVal(0);
1003     for(int i=1; i<numIndeces; i++)
1004       Db<<" AND "<<inval->printIndexName(i)<<"="<<inval->getIndexVal(i);
1005   }
1006   Db<<endsql;
1007   numElements=Db.NbRows();
1008   if(numElements<=0) return retElements;
1009   retElements = new int[numElements];
1010   int j=0;
1011   while(Db.Output(&buff)){
1012     buff.ReadScalar(retElements[j],"elementID");
1013     j++;
1014     buff.Raz();
1015   }
1016   clear();
1017   return retElements;
1018 }
```

4.34.2.46 int StDbSql::sendMess (const char * *a*, const char * *b*, StDbMessLevel *m*, int *lineNum* = 0, const char * *className* = " ", const char * *methName* = " ") [inline, protected]

Definition at line 209 of file StDbSql.hh.

```
209
210   if(m==dbMDebug && !(mgr->IsVerbose()))return 0;
211   return mgr->printInfo(a,b,m,lineNum,className,methName);
212 }
```

4.34.2.47 void StDbSql::setDbUtils (MysqlDb & *db*, StDbBuffer & *buffer*)

Definition at line 1627 of file StDbSql.cc.

```
1627                                                    {
1628   Db=db;
1629   buff=buffer;
1630 };
```

4.34.2.48 void StDbSql::setDefaultBeginTime (StDbTable * *table*, unsigned int *reqTime*)

Definition at line 1115 of file StDbSql.cc.

```
1115                                                                    {
1116   // set default return times
1117   char* stime=getDateTime(reqTime);
1118   table->setBeginTime(reqTime);                    // uint version
1119   table->setBeginTime(stime);                     // char* version
1120   if(stime) delete [] stime;
1121 }
```

4.34.2.49 void StDbSql::setDefaultEndTime (StDbTable * *table*)

Definition at line 1124 of file StDbSql.cc.

```
1124                                              {
1125   if(!mdefaultEndDateTime)initEndTime();
1126   table->setEndTime(mdefaultEndDateTime);
1127   table->setEndTime(mdefaultEndTime);
1128 }
```

4.34.2.50 void StDbSql::setDefaultReturnValues (StDbTable * *table*, unsigned int *reqTime*)

Definition at line 1107 of file StDbSql.cc.

```
1107                                                                  {
1108   if(!table)return;
1109   setDefaultBeginTime(table,reqTime);
1110   setDefaultEndTime(table);
1111   table->clearStoreInfo();
1112 }
```

4.34.2.51 int StDbSql::storeConfigNode (StDbConfigNode * *node*) [virtual]

Implements StDataBaseI.

Definition at line 1279 of file StDbSql.cc.

```
1279                                                {

1281   int retVal=0;

1283   Db<<insertNodeString((StDbNode*) node);

1285   if(node->getNumIndeces()){
1286     char* ename=0; int eid;
1287     node->getElementIndexInfo(ename,eid);
1288     Db<<", indexName='"<<ename<<"'";
1289     Db<<", indexVal="<<eid;
1290     if(ename) delete [] ename;
1291   }
1292   Db<<endsql;

1294   if(Db.QueryStatus())retVal=Db.GetLastInsertID();
1295   clear();

1297   return retVal;
1298 }
```

4.34.2.52 int StDbSql::storeTableNode (StDbTable * *table*) [virtual]

Implements StDataBaseI.

Definition at line 1302 of file StDbSql.cc.

```
1302                                                {

1304   int retVal=0;
1305   if(!table->printCstructName()) return retVal;

1307   Db<<insertNodeString((StDbNode*) table);
1308   Db<<", structName='"<<table->printCstructName()<<"'";

1310   if(table->IsBaseLine())Db<<", baseLine='Y'";
1311   if(table->IsBinary())Db<<", isBinary='Y'";
1312   if(!table->IsIndexed())Db<<", isIndexed='N'";

1314   Db<<endsql;

1316   if(Db.QueryStatus())retVal=Db.GetLastInsertID();
1317   clear();

1319   return retVal;
1320 }
```

4.34.2.53 bool StDbSql::updateEndTime (StDbTable * *table*, const char * *dataTable*, unsigned int *reqTime*) [protected]

Definition at line 452 of file StDbSql.cc.

```
452                                                {
```

```
453
454    /*******************************
455      resets the table's endtime based on set endTime (if table has this column)
456      instead of the running beginTime timestamp. Returns False if endTime is
457      earlier than requestTime
458     *******************************/
459
460   bool retVal = true;
461   if(!checkColumn(dataTable,"endTime")) return retVal;
462   int nrows;
463   int* wrows = table->getWrittenRows(nrows);
464
465   Db<<" select unix_timestamp(Min(endTime)) as mendTime from "<<dataTable;
466  // Db<<" where dataID In("<<getElementList(wrows,nrows)<<")"<<endsql;
467   Db<<" where dataID "<<getElementList(wrows,nrows)<<endsql;
468
469   if( (Db.NbRows()>0) && Db.Output(&buff)){
470     unsigned int t1 = table->getEndTime();
471     unsigned int t2;
472     buff.ReadScalar(t2,"mendTime");
473     if(t2<t1)table->setEndTime(t2);
474     if(t2<requestTime) retVal=false;
475     buff.Raz();
476   }
477
478   Db.Release();
479   return retVal;
480 }
```

4.34.2.54 void StDbSql::use () [inline, virtual]

Implements StDataBaseI.

Definition at line 220 of file StDbSql.hh.

```
220 { Db.setDefaultDb(mdbName); };
```

4.34.2.55 int StDbSql::WriteDb (StDbConfigNode * *node*, int *parentID*, int & *configID*) [virtual]

Implements StDataBaseI.

Definition at line 897 of file StDbSql.cc.

```
897                                                                    {
898 #define __METHOD__ "WriteDb(node,parentID,configID)"
899
900   if(!node) return 0;
901
902   char* nName = node->printName();
903
904   if(!parentID){
905
906     if(strcmp(node->printNodeType(),"Config")!=0)
907       return sendMess("No Config tag for new config=",nName,dbMErr,__LINE__,__CLASS__,__METHOD__);
908
909     if(!node->printVersion())
910       return sendMess("No version label for new config=",nName,dbMErr,__LINE__,__CLASS__,__METHOD__);
911
912   } else {
913     node->setNodeType("directory");
```

```
914   }
915
916   int nodeID;
917   if(!((nodeID)=storeConfigNode(node)))
918       return sendMess(" Could not store ",nName,dbMErr,__LINE__,__CLASS__,__METHOD__);
919
920   // write this node
921   if(parentID) {
922     insertNodeRelation(configID,parentID,nodeID);
923   } else {
924     configID=node->getNodeID();
925   }
926
927   // do this node's tables
928   if(node->hasData()){
929     StDbTableIter* itr=node->getStDbTableIter();
930     while(!itr->done()){
931       int childID=0;
932       StDbTable* table= itr->next();
933       table->setNodeType("table");
934       if(!((childID)=storeTableNode(table)))
935           return sendMess(" Could not store table in Node=",nName,dbMErr,__LINE__,__CLASS__,__METHOD__
936       insertNodeRelation(configID,nodeID,childID);
937     }
938   }
939
940 return nodeID;
941 }
```

4.34.2.56 int StDbSql::WriteDb (StDbTable * *table*, unsigned int *storeTime*) [virtual]

Implements StDataBaseI.

Definition at line 676 of file StDbSql.cc.

```
676                                                        {
677
678 #define __METHOD__ "WriteDb(StDbTable*,uint storeTime)"
679
680   int retVal=1;
681   char* tName=table->printName();
682
683   if(!table->hasData())
684     return sendMess(tName," has no data to store",dbMWarn,__LINE__,__CLASS__,__METHOD__)+1;
685
686   int nodeID;
687   if(!((nodeID)=table->getNodeID()))
688     if(!(prepareNode((StDbNode*)table)))
689          return sendMess(tName," Not found in DB",dbMErr,__LINE__,__CLASS__,__METHOD__);
690
691   readTableInfo(table);
692   clear();
693
694
695   if(table->IsBaseLine() && hasInstance(table))
696      return sendMess("BaseLine instance already exists",tName,dbMErr,__LINE__,__CLASS__,__METHOD__);
697
698   if(!QueryDescriptor(table))
699      return sendMess(tName," doesn't have a descriptor",dbMErr,__LINE__,__CLASS__,__METHOD__);
700
701
702   table->setRowNumber(); // set to 0
703   if(!table->IsIndexed())return WriteDbNoIndex(table,storeTime);
704
```

```
705   char* dataTable;
706   if(!((dataTable)=getDataTable(table,storeTime)))
707      return sendMess(tName," has no storage table",dbMErr,__LINE__,__CLASS__,__METHOD__);
708
709   int numRows;
710   int* elements = table->getElementID(numRows);
711
712   if(!elements)
713     return sendMess(tName,"doesn't have an Element List",dbMErr,__LINE__,__CLASS__,__METHOD__);
714
715   int* storedData = new int[numRows];
716   memset(storedData,0,numRows*sizeof(int));
717   char* sTime = getDateTime(storeTime);
718   int rowsWritten = 0;
719
720   // write each row & roll back full writes if any fail.
721   // - Also, for the time being, write to the old index
722   //   so that one can read with older versions of the code
723
724   table->commitData();
725   table->clearStoreInfo();
726   table->setDataTable(dataTable);
727
728   char* eTime=0;
729   if( table->getEndStoreTime() && checkColumn(dataTable,"endTime"))
730       eTime=getDateTime(table->getEndStoreTime());
731
732   for(int i=0;i<numRows;i++){
733       clear();
734       buff.WriteScalar(nodeID,"nodeID");
735       buff.WriteScalar(table->getSchemaID(),"schemaID");
736       buff.WriteScalar(sTime,"beginTime");
737       buff.WriteScalar(elements[i],"elementID");
738       if(eTime)buff.WriteScalar(eTime,"endTime");
739       if(!table->defaultFlavor())buff.WriteScalar(table->printFlavor(),"flavor");
740       table->dbStreamer(&buff,false);
741
742       if(!Db.Input(dataTable,&buff)){
743         deleteRows(dataTable,storedData,i);
744         retVal=0;
745         break;
746       } else {
747         storedData[i]=Db.GetLastInsertID();
748       }
749       clear();
750       rowsWritten++;
751    }
752   if(rowsWritten==numRows)table->addWrittenRows(storedData,numRows,true);
753
754   delete [] storedData;
755   delete [] sTime;
756   if(eTime) delete [] eTime;
757
758   table->setRowNumber();
759   delete [] dataTable;
760
761   if(!retVal)sendMess(" Write failed for table=",tName,dbMWarn,__LINE__,__CLASS__,__METHOD__);
762   return retVal;
763 #undef __METHOD__
764 }
```

4.34.2.57 int StDbSql::WriteDbNoIndex (StDbTable * *table*, unsigned int *storeTime*) [protected, virtual]

Definition at line 768 of file StDbSql.cc.

```
                                                                {
#define __METHOD__ "WriteDbNoIndex(table,storeTime)"

  int retVal=0;
  char* dataTable;
  if(!((dataTable)=getDataTable(table,storeTime)))
     return sendMess(table->printName()," has no storage table",dbMErr,__LINE__,__CLASS__,__METHOD__);

  StString cList;
  cList<<"beginTime,"<<getColumnList(table);

  char* sTime=getDateTime(storeTime);

  int numRows=table->GetNRows();
  char* colList = new char[strlen((cList.str()).c_str())+1];
  strcpy(colList,(cList.str()).c_str());

  int i;
  bool hasBinary=false;
  if(Db.InputStart(dataTable,&buff,colList,numRows,hasBinary)){

    if(hasBinary){ // got to go through the buffer

      cout<<" In Binary write???"<<endl;

      for(i=0;i<numRows;i++){
        buff.WriteScalar(sTime,"beginTime");
        table->dbStreamerWrite(&buff); //,false);
        if(!Db.InputRow(&buff,i)) break;
      }
      if( i==numRows && Db.InputEnd() ){
        retVal=1;
        table->commitData();
      }
    } else {

      table->setBeginTime(storeTime);
      StString fsql;
      StDbFastSqlWriter writer(fsql);
      writer.ioTable(table);
      Db<<fsql.str();
      if(Db.InputEnd()){
        retVal=1;
        table->commitData();
      }
    }

  }

  clear();

  table->setRowNumber();
  delete [] colList;
  delete [] sTime;

  delete [] dataTable;

  if(!retVal)sendMess(" Write failed for Non-Indexed table=",table->printName(),dbMWarn,__LINE__,__CLA
  return retVal;

#undef __METHOD__
}
```

4.34.3 Member Data Documentation

4.34.3.1 StDbBuffer& StDbSql::buff

Definition at line 197 of file StDbSql.hh.

4.34.3.2 MysqlDb& StDbSql::Db

Definition at line 196 of file StDbSql.hh.

4.34.3.3 char* StDbSql::mdefaultEndDateTime [protected]

Definition at line 103 of file StDbSql.hh.

4.34.3.4 unsigned int StDbSql::mdefaultEndTime [protected]

Definition at line 104 of file StDbSql.hh.

4.34.3.5 DescList StDbSql::mdescriptors [protected]

Definition at line 99 of file StDbSql.hh.

4.34.3.6 StDbManager* StDbSql::mgr [protected]

Definition at line 96 of file StDbSql.hh.

4.34.3.7 char* StDbSql::mretString [protected]

Definition at line 101 of file StDbSql.hh.

4.34.3.8 int StDbSql::mtableCatalog [protected]

Definition at line 102 of file StDbSql.hh.

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

- StDbSql.hh
- StDbSql.cc

4.35 StDbStoreInfo Class Reference

```
#include <StDbStoreInfo.hh>
```

Public Member Functions

- StDbStoreInfo ()
- ~StDbStoreInfo ()
- int * getDataIDs (int &numRows)
- void addWrittenRow (int dataID)
- void commit ()
- bool canRollBack ()
- bool hasData ()
- void resetStoreInfo ()

Protected Member Functions

- void reSizeInternalStore ()

Protected Attributes

- int * mdataIDs
- int mnumRows
- int mMaxRows
- bool mcanRollBack

4.35.1 Constructor & Destructor Documentation

4.35.1.1 StDbStoreInfo::StDbStoreInfo ()

Definition at line 38 of file StDbStoreInfo.cc.

```
38 : mdataIDs(0), mnumRows(0), mMaxRows(0), mcanRollBack(false) {};
```

4.35.1.2 StDbStoreInfo::~StDbStoreInfo ()

Definition at line 40 of file StDbStoreInfo.cc.

```
40 { resetStoreInfo(); }
```

4.35.2 Member Function Documentation

4.35.2.1 void StDbStoreInfo::addWrittenRow (int *dataID*)

Definition at line 68 of file StDbStoreInfo.cc.

```
68                                          {
69
70   if(mnumRows==mMaxRows)reSizeInternalStore();
71   mdataIDs[mnumRows]=dataID;
72   mnumRows++;
73
74 };
```

4.35.2.2 bool StDbStoreInfo::canRollBack () [inline]

Definition at line 67 of file StDbStoreInfo.hh.

```
67 { return mcanRollBack; }
```

4.35.2.3 void StDbStoreInfo::commit () [inline]

Definition at line 66 of file StDbStoreInfo.hh.

```
66 {  mcanRollBack=false; }
```

4.35.2.4 int * StDbStoreInfo::getDataIDs (int & *numRows*)

Definition at line 61 of file StDbStoreInfo.cc.

```
61                                          {
62   numRows=mnumRows;
63   return mdataIDs;
64 }
```

4.35.2.5 bool StDbStoreInfo::hasData () [inline]

Definition at line 68 of file StDbStoreInfo.hh.

```
68 { return (bool) mnumRows; };
```

4.35.2.6 void StDbStoreInfo::resetStoreInfo ()

Definition at line 77 of file StDbStoreInfo.cc.

```
77                                    {
78   if(mdataIDs) delete [] mdataIDs;
79   mdataIDs=0;
80   mnumRows=0;
81   mMaxRows=0;
82   mcanRollBack=false;
83 }
```

4.35.2.7 void StDbStoreInfo::reSizeInternalStore () [protected]

Definition at line 44 of file StDbStoreInfo.cc.

```
44                                        {
45
46    int newMaxRows = mMaxRows+N_ROWS_INCREMENT;
47    int* newDataIDs = new int[newMaxRows];
48
49    memset(newDataIDs,0,newMaxRows*sizeof(int));
50    if(mdataIDs){
51      memcpy(newDataIDs,mdataIDs,mnumRows*sizeof(int));
52      delete [] mdataIDs;
53    }
54    mdataIDs = newDataIDs;
55    mMaxRows = newMaxRows;
56
57 }
```

4.35.3 Member Data Documentation

4.35.3.1 bool StDbStoreInfo::mcanRollBack [protected]

Definition at line 45 of file StDbStoreInfo.hh.

4.35.3.2 int* StDbStoreInfo::mdataIDs [protected]

Definition at line 41 of file StDbStoreInfo.hh.

4.35.3.3 int StDbStoreInfo::mMaxRows [protected]

Definition at line 43 of file StDbStoreInfo.hh.

4.35.3.4 int StDbStoreInfo::mnumRows [protected]

Definition at line 42 of file StDbStoreInfo.hh.

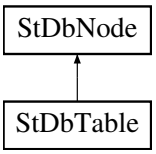
The documentation for this class was generated from the following files:

- StDbStoreInfo.hh
- StDbStoreInfo.cc

4.36 StDbTable Class Reference

`#include <StDbTable.h>`

Inheritance diagram for StDbTable::



Public Member Functions

- StDbTable (const char *tableName)
- StDbTable (const char *tableName, int schemaID)
- StDbTable (StDbTable &table)
- virtual ~StDbTable ()
- virtual bool IsTable () const
- virtual void setNodeInfo (StDbNode *node)
- virtual unsigned int getTableSize () const
- virtual char * getCstructName ()
- virtual char * printCstructName ()
- virtual void setCstructName (const char *name)
- virtual void setDataTable (const char *name)
- virtual char * getDataTable ()
- virtual char * printDataTable ()
- virtual char * getFlavor ()
- virtual char * printFlavor ()
- virtual void setFlavor (const char *flavor)
- virtual bool defaultFlavor () const
- virtual void setDefaultFlavor ()
- virtual unsigned int getProdTime ()
- virtual void setProdTime (unsigned int ptime)
- virtual unsigned int getEndTime () const
- virtual char * getEndDateTime ()
- virtual void setEndTime (unsigned int time)
- virtual void setEndTime (const char *time)
- virtual unsigned int getBeginTime () const
- virtual char * getBeginDateTime ()
- virtual void setBeginTime (unsigned int time)
- virtual void setBeginTime (const char *time)
- virtual unsigned int getEndStoreTime () const
- virtual char * getEndStoreDateTime ()
- virtual void setEndStoreTime (unsigned int time)
- virtual void setEndStoreTime (const char *time)
- virtual int * getElementID (int &nrows)
- virtual int getRowID (int rowNumber) const
- virtual char * getElementName ()
- virtual char * printElementName ()

- virtual void setElementName (const char *ename)
- virtual void setElementID (int *elements, int nrows)
- virtual void setBaseLine (bool baseLine)
- virtual void setIndexed (bool indexed)
- virtual void setBinary (bool abinary)
- virtual bool IsBaseLine () const
- virtual bool IsIndexed () const
- virtual bool IsBinary () const
- virtual int getSchemaID () const
- virtual void setSchemaID (int id)
- virtual void addWrittenRows (int *dataID, int numRows, bool canRollBack=false)
- virtual int * getWrittenRows (int &numRows)
- virtual void commitData ()
- virtual void clearStoreInfo ()
- virtual unsigned int * getTimeValues ()
- virtual unsigned int getMaxTime ()
- virtual void setTimeValues (unsigned int *timeValues)
- virtual StTableDescriptorI * getDescriptorCpy () const
- virtual StTableDescriptorI * getDescriptor ()
- virtual void setDescriptor (StTableDescriptorI *descriptor)
- virtual bool hasDescriptor () const
- void checkDescriptor ()
- virtual StDbTable * Clone ()
- virtual char * GetTable ()
- virtual void * GetTableCpy ()
- virtual void SetTable (char *data, int nrows, int *idList=0)

  calloc'd version of data for StRoot

- virtual void AddRows (char *data, int nrows)
- virtual int GetNRows () const
- virtual void setRowNumber (int row=0)
- virtual bool hasData () const
- virtual void * getDataValue (const char *name, int rowNumber=0)
- virtual int getRowLimit () const
- virtual void setRowLimit (int nrows)
- virtual void addNRows (int newRows)
- virtual void addNElements (int *elements, int newRows)
- virtual void resizeNumRows (int nrows)
- virtual void resizeElementID (int numRows)
- virtual void StreamAccessor (typeAcceptor *accept, bool isReading)
- virtual void dbStreamer (typeAcceptor *accept, bool isReading)
- virtual void StreamAccessor (StDbBufferI *buff, bool isReading)
- virtual void dbStreamer (StDbBufferI *buff, bool isReading)
- virtual void dbStreamerWrite (StDbBufferI *buff)
- virtual void dbTableStreamer (StDbBufferI *buff, const char *name, bool isReading)

Protected Member Functions

- virtual void ReadElement (char *&ptr, char *name, int length, StTypeE type, StDbBuffer *buff)
- virtual void WriteElement (char *ptr, char *name, int length, StTypeE type, StDbBuffer *buff)
- virtual void PassOutElement (char *ptr, char *name, int length, StTypeE type, typeAcceptor *accept)
- virtual void PassInElement (char *ptr, char *name, int length, StTypeE type, typeAcceptor *accept)
- virtual void getElementSpecs (int elementNum, char *&c, char *&name, unsigned int &length, StTypeE &type)
- bool createMemory ()
- bool createMemory (int nrows)
- char * duplicateData ()
- void init ()
- void printNoDataReturned (const char *elementName)

Protected Attributes

- char * mflavor
- bool mdefaultFlavor
- unsigned int mprodTime
- StDbTime mbeginTime

  validity interval

- StDbTime mendTime
- StDbTime mendStoreTime
- char * mstructName
- char * melementName
- bool mIsBaseLine
- bool mIsBinary
- bool mIsIndexed
- char * mdataTable
- int mschemaID
- int * melementID
- StDbStoreInfo mstoredData

  for rolling back stores

- bool mhasDescriptor

  c-struct descriptor information

- StTableDescriptorI * mdescriptor
- char * mdata

  data & num of rows

- int mrows
- int mrowNumber
- bool mhasData
- int mrowsRequested
- unsigned int * mtimeVals

4.36.1 Constructor & Destructor Documentation

4.36.1.1 StDbTable::StDbTable (const char * *tableName*)

Definition at line 346 of file StDbTable.cc.

```
346 : StDbNode(tableName) { init();};
```

4.36.1.2 StDbTable::StDbTable (const char * *tableName*, int *schemaID*)

Definition at line 348 of file StDbTable.cc.

```
348 : StDbNode(tableName) {  init(); mschemaID=schemaID; }
```

4.36.1.3 StDbTable::StDbTable (StDbTable & *table*)

Definition at line 351 of file StDbTable.cc.

```
351                                     : StDbNode(table) {
352
353  init();
354  mflavor=table.getFlavor();
355  mdefaultFlavor = table.defaultFlavor();
356  mprodTime = table.getProdTime();
357
358  mschemaID = table.getSchemaID();
359  mrows = table.GetNRows();
360  mhasDescriptor=table.hasDescriptor();
361  mdescriptor=table.getDescriptorCpy();
362
363  mbeginTime.setDateTime(table.getBeginDateTime());
364  mbeginTime.setUnixTime(table.getBeginTime());
365  mendTime.setDateTime(table.getEndDateTime());
366  mendTime.setUnixTime(table.getEndTime());
367
368  char* tmp = table.GetTable();
369  if(mrows==0) mrows = 1;
370  if(tmp) {
371    unsigned int size =  mrows*table.getTableSize();
372    mdata = new char[size];
373    memcpy(mdata,tmp,size);
374    mhasData = true;
375  }
376
377  setCstructName(table.printCstructName());
378  setDataTable(table.printDataTable());
379
380 }
```

4.36.1.4 virtual StDbTable::~StDbTable () [inline, virtual]

Definition at line 210 of file StDbTable.h.

```
210                     {         if(melementID) delete [] melementID;
211                               if(mdescriptor)delete mdescriptor;
```

```
                                if(mdata) delete [] mdata;
                                if(mdataTable) delete [] mdataTable;
                                if(mstructName) delete []  mstructName;
                                if(melementName) delete [] melementName;
                                if(mtimeVals) delete [] mtimeVals;
                                if(mflavor) delete [] mflavor;
                        };
```

4.36.2 Member Function Documentation

4.36.2.1 void StDbTable::addNElements (int * *elements*, int *newRows*) [virtual]

Definition at line 730 of file StDbTable.cc.

```
                                                {


  if(!melementID) return;

  int i,j,k;
  k=mrows-newRows;
  if(k<0)return;
  j=0;
  for(i=k;i<mrows;i++){
    melementID[i]=elements[j];
    j++;
  }

}
```

4.36.2.2 void StDbTable::addNRows (int *newRows*) [virtual]

Definition at line 689 of file StDbTable.cc.

```
                              {

  if(!mdescriptor) return;

  int newRows = numRows+mrows;
  //unsigned int rowsize=mdescriptor->getTotalSizeInBytes();
  unsigned int rowsize;
  if(!mdescriptor->getTrowSize()) {
     rowsize=mdescriptor->getTotalSizeInBytes();
  }else{
      rowsize=mdescriptor->getTrowSize();
   }
  unsigned int len = newRows*rowsize;
  char* newData = new char[len];
  memset(newData,0,len);
  if(mdata)memcpy(newData,mdata,mrows*rowsize);
  char* p1 = newData;
  p1+=mrows*rowsize;
  memset(p1,0,numRows*rowsize);
  if(mdata)delete [] mdata;
  mdata=newData;

  resizeElementID(newRows);
};
```

4.36.2.3 void StDbTable::AddRows (char * *data*, int *nrows*) [virtual]

Definition at line 512 of file StDbTable.cc.

```
512                                     {
513
514 char* tmpData = duplicateData();
515 int len1 = mrows*getTableSize();
516 int len2 = nrows*getTableSize();
517
518 int newRows = nrows+mrows;
519 if(mdata){
520    delete [] mdata;
521    mdata = 0;
522  }
523
524 createMemory(newRows);
525
526 char* ptr= &mdata[0];
527 memcpy(mdata,tmpData,len1);
528 ptr+=len1;
529 memcpy(ptr,c,len2);
530
531 delete [] tmpData;
532 mhasData=true;
533
534 }
```

4.36.2.4 void StDbTable::addWrittenRows (int * *dataID*, int *numRows*, bool *canRollBack* = false) [virtual]

Definition at line 445 of file StDbTable.cc.

```
445                                                                  {
446
447   for(int i=0; i<numRows; i++) mstoredData.addWrittenRow(dataID[i]);
448   if(canRollBack)mcanRollBack=true;
449
450 }
```

4.36.2.5 void StDbTable::checkDescriptor ()

Definition at line 1367 of file StDbTable.cc.

```
1367                          {
1368
1369 int i = mdescriptor->getNumElements();
1370 unsigned int size = mdescriptor->getTotalSizeInBytes();
1371  cout <<"Descriptor for Table = " << mname<<endl;
1372  cout <<" number of elements = "<<i<< " with size = " << size << endl;
1373  for(int k=0; k<i;k++){
1374    cout <<"Name = " << mdescriptor->getElementName(k);
1375    cout <<" size = " << mdescriptor->getElementSize(k);
1376    cout <<" offset = " <<mdescriptor->getElementOffset(k);
1377    cout <<" type = " <<(int)mdescriptor->getElementType(k)<<endl;
1378  }
1379 }
```

4.36.2.6 void StDbTable::clearStoreInfo () [virtual]

Definition at line 457 of file StDbTable.cc.

```
457 { mstoredData.resetStoreInfo(); }
```

4.36.2.7 StDbTable * StDbTable::Clone () [inline, virtual]

Definition at line 372 of file StDbTable.h.

```
372 {return (new StDbTable(*this));}
```

4.36.2.8 void StDbTable::commitData () [virtual]

Definition at line 456 of file StDbTable.cc.

```
456 { mstoredData.commit(); }
```

4.36.2.9 bool StDbTable::createMemory (int *nrows*) [protected]

Definition at line 579 of file StDbTable.cc.

```
579                                 {
580  mrows = nrows;
581  bool retVal = true;
582  if(mrows==0) {
583    if(mdata)delete [] mdata;
584    mdata=0;
585    return retVal;
586  }
587
588 // mdescriptor->getNumElements();
589 //if (mdescriptor) {
590 //cout <<"**************CHECKIT************"<<endl;
591 //}
592  if(mdescriptor && mdescriptor->getNumElements()>0){
593     //     if(mrows==0) mrows = 1;
594       //int len = mrows*mdescriptor->getTotalSizeInBytes();
595      int len;
596      if (!mdescriptor->getTrowSize()){
597        len =  mrows*mdescriptor->getTotalSizeInBytes();
598      }else{
599       len = mrows*mdescriptor->getTrowSize();
600      }
601       if(len>0){
602       if(mdata)delete [] mdata;
603       mdata=new char[len];
604       memset(mdata,0,len);
605       int max = mdescriptor->getNumElements();
606       char* name;
607       StTypeE type;
608       unsigned int length;
609       char * ptr;
610       for(int i=0; i<max;i++){
611         getElementSpecs(i,ptr,name,length,type);
612         if(type==Stchar)ptr='\0';
```

```
613         delete [] name;
614       }
615      }
616   } else {
617     if(!mname){mname=mstrDup("Unknown");}
618     retVal = false;
619   }
620
621 return retVal;
622 }
```

4.36.2.10 bool StDbTable::createMemory () [protected]

Definition at line 626 of file StDbTable.cc.

```
626                        {
627   if(mdata)return true;
628   if(mrows==0) mrows=1;
629   return createMemory(mrows);
630 }
```

4.36.2.11 void StDbTable::dbStreamer (StDbBufferI * *buff*, bool *isReading*) [virtual]

Definition at line 833 of file StDbTable.cc.

```
833                                                        {
834
835 int max = mdescriptor->getNumElements();
836 char* name;
837 StTypeE type;
838 unsigned int length;
839 char* ptr;
840
841   bool ClientMode;
842   if(!(ClientMode=buff->IsClientMode()))buff->SetClientMode();
843
844  if(createMemory() && mrowNumber < mrows){
845
846  for(int i=0; i<max; i++){
847     getElementSpecs(i,ptr,name,length,type);
848     if(isReading){
849      ReadElement(ptr,name,length,type,(StDbBuffer*)buff);
850      } else {
851      WriteElement(ptr,name,length,type,(StDbBuffer*)buff);
852     }
853    delete [] name;
854  }
855
856   mrowNumber++;
857   if(isReading)mhasData=true;
858
859  } else {
860    cerr << "dbStreamer:: more rows delivered than allocated " << endl;
861  }
862  if(!ClientMode)buff->SetStorageMode();  // reset to StorageMode
863 }
```

4.36.2.12 void StDbTable::dbStreamer (typeAcceptor * *accept*, bool *isReading*) [virtual]

Definition at line 926 of file StDbTable.cc.

```
926                                                         {
927
928 int max = mdescriptor->getNumElements();
929 char* name;
930 StTypeE type;
931 unsigned int length;
932 char* ptr;
933
934  if(createMemory() && mrowNumber < mrows){
935
936    if(isReading){
937      for(int i=0; i<max; i++){
938       getElementSpecs(i,ptr,name,length,type);
939       PassInElement(ptr,name,length,type,accept);
940       delete [] name;
941      }
942      mhasData=true;
943    } else {
944      for(int i=0; i<max; i++){
945       getElementSpecs(i,ptr,name,length,type);
946       PassOutElement(ptr,name,length,type,accept);
947       delete [] name;
948      }
949    }
950
951  mrowNumber++;
952  }
953
954 }
```

4.36.2.13 void StDbTable::dbStreamerWrite (StDbBufferI * *buff*) [virtual]

Definition at line 868 of file StDbTable.cc.

```
868                                             {
869
870 int max = mdescriptor->getNumElements();
871 char* name;
872 StTypeE type;
873 unsigned int length;
874 char* ptr;
875
876 //  bool ClientMode;
877 //  if(!(ClientMode=buff->IsClientMode()))buff->SetClientMode();
878
879 // if(createMemory() && mrowNumber < mrows){
880
881  if(mrowNumber<mrows){
882    for(int i=0; i<max; i++){
883      getElementSpecs(i,ptr,name,length,type);
884      WriteElement(ptr,name,length,type,(StDbBuffer*)buff);
885      delete [] name;
886    }
887   mrowNumber++;
888  }
889 }
```

4.36.2.14 void StDbTable::dbTableStreamer (StDbBufferI * *buff*, const char * *name*, bool *isReading*) [virtual]

Definition at line 894 of file StDbTable.cc.

```
894                                                                                    {
895
896 int max = mdescriptor->getNumElements();
897 //int size = mdescriptor->getTotalSizeInBytes();
898 StTypeE type = mdescriptor->getElementType(0);
899 unsigned int length = (unsigned int) mrows*max;
900
901 char* ptr;
902
903   bool ClientMode;
904   if(!(ClientMode=buff->IsClientMode()))buff->SetClientMode();
905
906  if(createMemory() && mrowNumber < mrows){
907
908    ptr = &mdata[0];
909    //    getElementSpecs(i,ptr,name,length,type);
910    if(isReading){
911      ReadElement(ptr,(char *) name,length,type,(StDbBuffer*)buff);
912    } else {
913      WriteElement(ptr,(char *) name,length,type,(StDbBuffer*)buff);
914    }
915   mrowNumber=mrows;
916   if(isReading) mhasData=true;
917  }
918
919  if(!ClientMode)buff->SetStorageMode();  // reset to StorageMode
920 }
```

4.36.2.15 bool StDbTable::defaultFlavor () const [inline, virtual]

Definition at line 336 of file StDbTable.h.

```
336 {return mdefaultFlavor;}
```

4.36.2.16 char * StDbTable::duplicateData () [protected]

Definition at line 565 of file StDbTable.cc.

```
565                          {
566
567  char* dup=0;
568  int len1 = mrows*getTableSize();
569  if(len1 !=0){
570   dup=new char[len1];
571   memcpy(dup,mdata,len1);
572  }
573 return dup;
574 }
```

4.36.2.17 char * StDbTable::getBeginDateTime () [inline, virtual]

Definition at line 344 of file StDbTable.h.

```
344 {return mbeginTime.mdateTime; }
```

4.36.2.18 unsigned int StDbTable::getBeginTime () const [inline, virtual]

Definition at line 343 of file StDbTable.h.

```
343 {return mbeginTime.munixTime; }
```

4.36.2.19 char * StDbTable::getCstructName () [virtual]

Definition at line 418 of file StDbTable.cc.

```
418 { return mstrDup(mstructName); }
```

4.36.2.20 char * StDbTable::getDataTable () [virtual]

Definition at line 424 of file StDbTable.cc.

```
424 { return mstrDup(mdataTable); }
```

4.36.2.21 void * StDbTable::getDataValue (const char * *name*, int *rowNumber* = 0) [virtual]

Definition at line 538 of file StDbTable.cc.

```
538                                                         {
539
540   void* retVal=0;
541   int saveRowNum=mrowNumber;
542   mrowNumber=rowNumber;
543   int max = mdescriptor->getNumElements();
544   char* ename=0;
545   StTypeE type;
546   unsigned int length;
547   char * ptr;
548
549   for(int i=0;i<max;i++){
550     getElementSpecs(i,ptr,ename,length,type);
551     if(strcmp(name,ename)==0)break;
552     if(ename) delete [] ename;
553     ename=0;
554   }
555
556   mrowNumber=saveRowNum;
557   if(!ename) return retVal;
558
559   delete [] ename;
560   return (void*)ptr;
561 }
```

4.36.2.22 StTableDescriptorI * StDbTable::getDescriptor () [inline, virtual]

Definition at line 370 of file StDbTable.h.

```
370 { return mdescriptor; }
```

4.36.2.23 StTableDescriptorI * StDbTable::getDescriptorCpy () const [virtual]

Definition at line 461 of file StDbTable.cc.

```
461 { return mdescriptor->getCpy(); }
```

4.36.2.24 int * StDbTable::getElementID (int & *nrows*) [inline, virtual]

Definition at line 351 of file StDbTable.h.

```
351 { nrows = mrows; return melementID; }
```

4.36.2.25 char * StDbTable::getElementName () [virtual]

Definition at line 635 of file StDbTable.cc.

```
635 { return mstrDup(melementName); };
```

4.36.2.26 void StDbTable::getElementSpecs (int *elementNum*, char *& *c*, char *& *name*, unsigned int & *length*, StTypeE & *type*) [protected, virtual]

Definition at line 810 of file StDbTable.cc.

```
810                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                {
811
812      int tRow = mdescriptor->getTrowSize();
813     unsigned int tSize=mdescriptor->getTotalSizeInBytes();
814     unsigned int rowIndex;
815     if (!tRow) {
816         rowIndex = ((unsigned int)mrowNumber)*tSize;
817      }else{
818         rowIndex = ((unsigned int)mrowNumber)*tRow;
819      }
820     int i = elementNum;
821     c = &mdata[rowIndex];
822     int current = mdescriptor->getElementOffset(i);
823     c += current; // for(int k=0;k<current;k++)c++;
824     name   = mdescriptor->getElementName(i);
825     length = mdescriptor->getElementLength(i);;
826     type   = mdescriptor->getElementType(i);
827
828 return;
829 }
```

4.36.2.27 char * StDbTable::getEndDateTime () [inline, virtual]

Definition at line 340 of file StDbTable.h.

```
340 { return mendTime.mdateTime; }
```

4.36.2.28 char * StDbTable::getEndStoreDateTime () [inline, virtual]

Definition at line 348 of file StDbTable.h.

```
348 { return mendStoreTime.mdateTime; }
```

4.36.2.29 unsigned int StDbTable::getEndStoreTime () const [inline, virtual]

Definition at line 347 of file StDbTable.h.

```
347 { return mendStoreTime.munixTime; }
```

4.36.2.30 unsigned int StDbTable::getEndTime () const [inline, virtual]

Definition at line 339 of file StDbTable.h.

```
339 {return mendTime.munixTime; }
```

4.36.2.31 char * StDbTable::getFlavor () [virtual]

Definition at line 440 of file StDbTable.cc.

```
440 { return mstrDup(mflavor); }
```

4.36.2.32 unsigned int StDbTable::getMaxTime () [inline, virtual]

Definition at line 362 of file StDbTable.h.

```
362                                          {
363   unsigned int retVal=0;
364   for(int i=0; i<mrows;i++)if(mtimeVals[i]>retVal)retVal=mtimeVals[i];
365   return retVal;
366 }
```

4.36.2.33 int StDbTable::GetNRows () const [inline, virtual]

Definition at line 373 of file StDbTable.h.

```
373 { return mrows; }
```

4.36.2.34 unsigned int StDbTable::getProdTime () [inline, virtual]

Definition at line 338 of file StDbTable.h.

```
338 { return mprodTime; }
```

4.36.2.35 int StDbTable::getRowID (int *rowNumber*) const [inline, virtual]

Definition at line 353 of file StDbTable.h.

```
353                                                 {
354   if(rowNumber<mrows)return melementID[rowNumber];
355   return 0;
356 }
```

4.36.2.36 int StDbTable::getRowLimit () const [inline, virtual]

Definition at line 375 of file StDbTable.h.

```
375 { return mrowsRequested; };
```

4.36.2.37 int StDbTable::getSchemaID () const [inline, virtual]

Definition at line 368 of file StDbTable.h.

```
368 { return mschemaID; }
```

4.36.2.38 char * StDbTable::GetTable () [virtual]

Definition at line 476 of file StDbTable.cc.

```
476 { if(!mdata)createMemory(); return mdata;};
```

4.36.2.39 void * StDbTable::GetTableCpy () [virtual]

Definition at line 480 of file StDbTable.cc.

```
480                       {
481
482 if(!mdata)return (void*)GetTable();
483
484  int len =  mrows*getTableSize();
485  char* c = (char*)calloc(mrows,getTableSize());
486  memcpy(c,mdata,len);
487
488 return (void*)c;
489 };
```

4.36.2.40 unsigned int StDbTable::getTableSize () const [inline, virtual]

Definition at line 328 of file StDbTable.h.

```
328                                                 {
329 if(mhasDescriptor && !mdescriptor->getTrowSize()) return mdescriptor->getTotalSizeInBytes();
330 if(mhasDescriptor) return mdescriptor->getTrowSize();
331 return 0;
332 }
```

4.36.2.41 unsigned int * StDbTable::getTimeValues () [inline, virtual]

Definition at line 357 of file StDbTable.h.

```
357 { return mtimeVals; }
```

4.36.2.42 int * StDbTable::getWrittenRows (int & *numRows*) [virtual]

Definition at line 453 of file StDbTable.cc.

```
453                                     {
454        return mstoredData.getDataIDs(nRows);
455 }
```

4.36.2.43 bool StDbTable::hasData () const [inline, virtual]

Definition at line 377 of file StDbTable.h.

```
377 { return mhasData; };
```

4.36.2.44 bool StDbTable::hasDescriptor () const [inline, virtual]

Definition at line 371 of file StDbTable.h.

```
371 { return mhasDescriptor; }
```

4.36.2.45 void StDbTable::init () [protected]

Definition at line 382 of file StDbTable.cc.

```
382                      {
383  mflavor            = 0;
384  mstructName        = 0;
385  melementName       = 0;
386  mdataTable         = 0;
387  melementID         = 0;
388  mhasDescriptor     = false;
389  mdescriptor        = 0;
390  mdata              = 0;
```

```
391  mhasData           = false;
392  mrowsRequested     = 0;
393  mtimeVals          = 0;
394  mendTime.munixTime = 0;
395  mrows              = 0;
396  mrowNumber         = 0;
397  mprodTime          = StDbDefaults::Instance()->getProdTime();
398  setDefaultFlavor();
399  mschemaID = 0;
400
401 }
```

4.36.2.46 bool StDbTable::IsBaseLine () const [inline, virtual]

Definition at line 382 of file StDbTable.h.

```
382 { return mIsBaseLine; }
```

4.36.2.47 bool StDbTable::IsBinary () const [inline, virtual]

Definition at line 381 of file StDbTable.h.

```
381 { return mIsBinary; }
```

4.36.2.48 bool StDbTable::IsIndexed () const [inline, virtual]

Definition at line 383 of file StDbTable.h.

```
383 { return mIsIndexed; }
```

4.36.2.49 bool StDbTable::IsTable () const [inline, virtual]

Reimplemented from StDbNode.

Definition at line 327 of file StDbTable.h.

```
327 { return true; };
```

4.36.2.50 void StDbTable::PassInElement (char * *ptr*, char * *name*, int *length*, StTypeE *type*, typeAcceptor * *accept*) [protected, virtual]

Definition at line 1160 of file StDbTable.cc.

```
1160                                                                    {
1161
1162
1163   switch (type) {
1164   case Stchar:
1165     {
1166     char* data;
```

```
    accept->pass(name,data,len);
    memcpy(ptr,data,len);
    delete [] data;
    break;
    }
  case Stuchar:
    {
     unsigned char* data;
     accept->pass(name,data,len);
     memcpy(ptr,data,len);
     delete [] data;
     break;
    }
  case Stshort:
    {
     short* data;
     accept->pass(name,data,len);
     memcpy(ptr,data,len*sizeof(short));
     delete [] data;
     break;
    }
  case Stushort:
    {
     unsigned short* data;
     accept->pass(name,data,len);
     memcpy(ptr,data,len*sizeof(short));
     delete [] data;
     break;
    }
  case Stint:
    {
     int* data;
     accept->pass(name,data,len);
     memcpy(ptr,data,len*sizeof(int));
     delete [] data;
     break;
    }
  case Stuint:
    {
     unsigned int* data;
     accept->pass(name,data,len);
     memcpy(ptr,data,len*sizeof(int));
     delete [] data;
     break;
    }
  case Stlong:
    {
     long* data;
     accept->pass(name,data,len);
     memcpy(ptr,data,len*sizeof(long));
     delete [] data;
     break;
    }
  case Stulong:
    {
     unsigned long* data;
     accept->pass(name,data,len);
     memcpy(ptr,data,len*sizeof(long));
     delete [] data;
     break;
    }
  case Stlonglong:
    {
     long long* data;
     accept->pass(name,data,len);
     memcpy(ptr,data,len*sizeof(long long));
     delete [] data;
```

```
      break;
     }
   case Stfloat:
     {
      float* data;
      accept->pass(name,data,len);
      memcpy(ptr,data,len*sizeof(float));
      delete [] data;
      break;
     }
   case Stdouble:
     {
      double* data;
      accept->pass(name,data,len);
      memcpy(ptr,data,len*sizeof(double));
      delete [] data;
      break;
     }
   }
}
```

4.36.2.51 void StDbTable::PassOutElement (char * *ptr*, char * *name*, int *length*, StTypeE *type*, typeAcceptor * *accept*) [protected, virtual]

Definition at line 1258 of file StDbTable.cc.

```
{

   switch (type) {
   case Stchar:
     {
     accept->pass(name,ptr,len);
     break;
     }
   case Stuchar:
     {
      unsigned char* muchar = (unsigned char*)ptr;
      accept->pass(name, muchar,len);
     break;
     }
   case Stshort:
     {
     short* mshort = (short*)ptr;
     if(len==1){
       accept->pass(name, *mshort ,len);
     } else {
       accept->pass(name,mshort,len);
     }
     break;
     }
   case Stushort:
     {
     unsigned short* mushort = (unsigned short*)ptr;
     if(len==1){
       accept->pass(name, *mushort ,len);
     } else {
       accept->pass(name,mushort,len);
     }
     break;
     }
   case Stint:
     {
     int* mint = (int*)ptr;
     if(len==1){
```

```
      accept->pass(name, *mint ,len);
    } else {
      accept->pass(name,mint,len);
    }
    break;
    }
  case Stuint:
    {
    unsigned int* muint = (unsigned int*)ptr;
    if(len==1){
      accept->pass(name, *muint ,len);
    } else {
      accept->pass(name,muint,len);
    }
    break;
    }
  case Stlong:
    {
    long* mlong = (long*)ptr;
    if(len==1){
      accept->pass(name, *mlong ,len);
    } else {
      accept->pass(name,mlong,len);
    }
    break;
    }
  case Stulong:
    {
    unsigned long* mulong = (unsigned long*)ptr;
    if(len==1){
      accept->pass(name, *mulong ,len);
    } else {
      accept->pass(name,mulong,len);
    }
    break;
    }
  case Stlonglong:
    {
    long long* mlonglong = (long long*)ptr;
    if(len==1){
      accept->pass(name, *mlonglong ,len);
    } else {
      accept->pass(name,mlonglong,len);
    }
    break;
    }
  case Stfloat:
    {
    float* mfloat = (float*)ptr;
    if(len==1){
      accept->pass(name, *mfloat ,len);
    } else {
      accept->pass(name,mfloat,len);
    }
    break;
    }
  case Stdouble:
    {
    double* mdouble = (double*)ptr;
    if(len==1){
      accept->pass(name, *mdouble ,len);
    } else {
      accept->pass(name,mdouble,len);
    }
    break;
    }
  }
```

```
1363 }
```

4.36.2.52 char * StDbTable::printCstructName () [inline, virtual]

Definition at line 333 of file StDbTable.h.

```
333 { return mstructName; }
```

4.36.2.53 char * StDbTable::printDataTable () [inline, virtual]

Definition at line 334 of file StDbTable.h.

```
334 { return mdataTable; }
```

4.36.2.54 char * StDbTable::printElementName () [inline, virtual]

Definition at line 367 of file StDbTable.h.

```
367 { return melementName; }
```

4.36.2.55 char * StDbTable::printFlavor () [inline, virtual]

Definition at line 335 of file StDbTable.h.

```
335 { return mflavor; }
```

4.36.2.56 void StDbTable::printNoDataReturned (const char * *elementName*) [protected]

Definition at line 1382 of file StDbTable.cc.

```
1382                                                        {
1383
1384   StString emess;
1385   emess<<" No data return from table="<<printName()<<" column="<<elementName;
1386   StDbManager::Instance()->printInfo((emess.str()).c_str(),dbMWarn,__LINE__,__CLASS__,"ReadElement(pt
1387 }
```

4.36.2.57 void StDbTable::ReadElement (char *& *ptr*, char * *name*, int *length*, StTypeE *type*, StDbBuffer * *buff*) [protected, virtual]

Definition at line 959 of file StDbTable.cc.

```
                                                                                {

char* mchar; unsigned char* muchar; short* mshort; unsigned short* mushort;
int* mint; unsigned int* muint; long* mlong; unsigned long* mulong;
long long* mlonglong;
float* mfloat; double* mdouble;

 int blen; // length returned from db  ### use lesser of len & blen

  switch (type) {
  case Stchar:
    {
        StString cn;
        cn<<name<<".text"; const char* commentName = (cn.str()).c_str();
        mchar = 0;
        if(!buff->ReadScalar(mchar,commentName))buff->ReadScalar(mchar,name);
        if(mchar){
             int len1=strlen(mchar);
             if(len>len1) len=len1;
             strncpy(ptr,mchar,len);
             delete [] mchar;
        } else {
             *ptr='\0';
             printNoDataReturned(name);
        }
    break;
    }
  case Stuchar:
    {
      if(buff->ReadArray(muchar,blen,name)){
        if(len>blen)len=blen;
        memcpy(ptr,muchar,len*sizeof(unsigned char));
        delete [] muchar;
      } else { printNoDataReturned(name); }
    break;
    }
  case Stshort:
    {
      if(buff->ReadArray(mshort,blen,name)){
        if(len>blen)len=blen;
        memcpy(ptr,mshort,len*sizeof(short));
         delete [] mshort;
       } else { printNoDataReturned(name); }
     break;
     }
   case Stushort:
     {
       if(buff->ReadArray(mushort,blen,name)){
         if(len>blen)len=blen;
         memcpy(ptr,mushort,len*sizeof(unsigned short));
         delete [] mushort;
       } else { printNoDataReturned(name); }
     break;
     }
   case Stint:
     {
       if(buff->ReadArray(mint,blen,name)){
         if(len>blen)len=blen;
          memcpy(ptr,mint,len*sizeof(int));
          delete [] mint;
       } else { printNoDataReturned(name); }
     break;
     }
   case Stuint:
     {
       if(buff->ReadArray(muint,blen,name)){
         if(len>blen)len=blen;
```

```
1026        memcpy(ptr,muint,len*sizeof(unsigned int));
1027        delete [] muint;
1028       } else { printNoDataReturned(name); }
1029     break;
1030     }
1031   case Stlong:
1032     {
1033       if(buff->ReadArray(mlong,blen,name)){
1034         if(len>blen)len=blen;
1035        memcpy(ptr,mlong,len*sizeof(long));
1036        delete [] mlong;
1037       } else { printNoDataReturned(name); }
1038     break;
1039     }
1040   case Stulong:
1041     {
1042       if(buff->ReadArray(mulong,blen,name)){
1043         if(len>blen)len=blen;
1044        memcpy(ptr,mulong,len*sizeof(unsigned long));
1045        delete [] mulong;
1046       } else { printNoDataReturned(name); }
1047     break;
1048     }
1049   case Stlonglong:
1050     {
1051       if(buff->ReadArray(mlonglong,blen,name)){
1052         if(len>blen)len=blen;
1053        memcpy(ptr,mlonglong,len*sizeof(long long));
1054        delete [] mlonglong;
1055       } else { printNoDataReturned(name); }
1056     break;
1057     }
1058   case Stfloat:
1059     {
1060       if(buff->ReadArray(mfloat,blen,name)){
1061        if(len>blen)len=blen;
1062        memcpy(ptr,mfloat,len*sizeof(float));
1063        delete [] mfloat;
1064       } else { printNoDataReturned(name); }
1065     break;
1066     }
1067   case Stdouble:
1068     {
1069       if(buff->ReadArray(mdouble,blen,name)){
1070        if(len>blen)len=blen;
1071        memcpy(ptr,mdouble,len*sizeof(double));
1072        delete [] mdouble;
1073       } else { printNoDataReturned(name); }
1074     break;
1075     }
1076   }
1077
1078 }
```

4.36.2.58 void StDbTable::resizeElementID (int *numRows*) [virtual]

Definition at line 716 of file StDbTable.cc.

```
716                                     {
717
718   int * newElements=new int[numRows];
719   if(melementID) {
720     memcpy(newElements,melementID,mrows*sizeof(int));
721     delete [] melementID;
```

```
722   }
723   melementID=newElements;
724   mrows = numRows;
725
726 };
```

4.36.2.59 void StDbTable::resizeNumRows (int *nrows*) [virtual]

Definition at line 655 of file StDbTable.cc.

```
655                                       {
656   // if only some rows are returned, this is called to
657   // compress memory
658
659   //unsigned int rowsize=mdescriptor->getTotalSizeInBytes();
660   unsigned int rowsize;
661   if (!mdescriptor->getTrowSize()) {
662      rowsize=mdescriptor->getTotalSizeInBytes();
663   }else{
664       rowsize=mdescriptor->getTrowSize();
665    }
666   unsigned int len = mrows*rowsize;
667   unsigned int newlen = nrows*rowsize;
668
669   if(mdata){
670     char* oldData=new char[len];
671     memcpy(oldData,mdata,len);
672     delete [] mdata;
673     mdata = new char[newlen];
674     if(newlen<=len){
675       memcpy(mdata,oldData,newlen);
676     } else {
677       memcpy(mdata,oldData,len);
678     }
679     delete [] oldData;
680   }
681
682   mrows=nrows;
683   return;
684 }
```

4.36.2.60 void StDbTable::setBaseLine (bool *baseLine*) [inline, virtual]

Definition at line 379 of file StDbTable.h.

```
379 { mIsBaseLine=baseLine; }
```

4.36.2.61 void StDbTable::setBeginTime (const char * *time*) [inline, virtual]

Definition at line 346 of file StDbTable.h.

```
346 { mbeginTime.setDateTime(time); }
```

4.36.2.62 void StDbTable::setBeginTime (unsigned int *time*) [inline, virtual]

Definition at line 345 of file StDbTable.h.

```
345 {mbeginTime.munixTime = time; }
```

4.36.2.63 void StDbTable::setBinary (bool *abinary*) [inline, virtual]

Definition at line 378 of file StDbTable.h.

```
378 { mIsBinary=abinary; }
```

4.36.2.64 void StDbTable::setCstructName (const char * *name*) [virtual]

Definition at line 415 of file StDbTable.cc.

```
415 { mstructName=mstrDup(name); }
```

4.36.2.65 void StDbTable::setDataTable (const char * *name*) [virtual]

Definition at line 421 of file StDbTable.cc.

```
421 { mdataTable=mstrDup(name); }
```

4.36.2.66 void StDbTable::setDefaultFlavor () [virtual]

Definition at line 428 of file StDbTable.cc.

```
428                             { setFlavor(StDbDefaults::Instance()->printFlavor());
429 }
```

4.36.2.67 void StDbTable::setDescriptor (StTableDescriptorI * *descriptor*) [virtual]

Definition at line 465 of file StDbTable.cc.

```
465                                                     {
466
467  if(mdescriptor) delete mdescriptor;
468  mdescriptor=descriptor;
469  mhasDescriptor=true;
470
471  //checkDescriptor();
472
473 };
```

4.36.2.68 void StDbTable::setElementID (int * *elements*, int *nrows*) [virtual]

Definition at line 641 of file StDbTable.cc.

```
641                                                 {
642
643    createMemory(nrows);
644    // set up & fill char* will element list
645    if(melementID) delete [] melementID;
646    if(nrows==0){
647      melementID=0;
648      return;
649    }
650    melementID = new int[nrows];
651    memcpy(melementID, elements, nrows*sizeof(int));
652 }
```

4.36.2.69 void StDbTable::setElementName (const char * *ename*) [virtual]

Definition at line 638 of file StDbTable.cc.

```
638 { melementName=mstrDup(name);};
```

4.36.2.70 void StDbTable::setEndStoreTime (const char * *time*) [inline, virtual]

Definition at line 350 of file StDbTable.h.

```
350 { mendStoreTime.setDateTime(time); }
```

4.36.2.71 void StDbTable::setEndStoreTime (unsigned int *time*) [inline, virtual]

Definition at line 349 of file StDbTable.h.

```
349 {mendStoreTime.munixTime = time; }
```

4.36.2.72 void StDbTable::setEndTime (const char * *time*) [inline, virtual]

Definition at line 342 of file StDbTable.h.

```
342 {mendTime.setDateTime(time);}
```

4.36.2.73 void StDbTable::setEndTime (unsigned int *time*) [inline, virtual]

Definition at line 341 of file StDbTable.h.

```
341 { mendTime.munixTime=time;}
```

4.36.2.74 void StDbTable::setFlavor (const char * *flavor*) [virtual]

Definition at line 432 of file StDbTable.cc.

```
432                                                 {
433   if(!flavor) return;
434   if(mflavor) delete [] mflavor;
435   mflavor=new char[strlen(flavor)+1];
436   strcpy(mflavor,flavor);
437   mdefaultFlavor = StDbDefaults::Instance()->IsDefaultFlavor(mflavor);
438 }
```

4.36.2.75 void StDbTable::setIndexed (bool *indexed*) [inline, virtual]

Definition at line 380 of file StDbTable.h.

```
380 { mIsIndexed=indexed; }
```

4.36.2.76 void StDbTable::setNodeInfo (StDbNode * *node*) [virtual]

Definition at line 404 of file StDbTable.cc.

```
404                                                 {
405
406     mdbType=node->getDbType();
407     mdbDomain=node->getDbDomain();
408     setDbName(node->printDbName());
409     setVersion(node->printVersion());
410     setNodeType(node->printNodeType());
411     mnodeID=node->getNodeID();
412 };
```

4.36.2.77 void StDbTable::setProdTime (unsigned int *ptime*) [inline, virtual]

Definition at line 337 of file StDbTable.h.

```
337 { mprodTime=ptime; }
```

4.36.2.78 void StDbTable::setRowLimit (int *nrows*) [inline, virtual]

Definition at line 374 of file StDbTable.h.

```
374 { mrowsRequested=nrows; };
```

4.36.2.79 void StDbTable::setRowNumber (int *row* = 0) [inline, virtual]

Definition at line 376 of file StDbTable.h.

```
376 {if(row < mrows)mrowNumber = row;}
```

4.36.2.80 void StDbTable::setSchemaID (int *id*) [inline, virtual]

Definition at line 369 of file StDbTable.h.

```
369 {mschemaID = id; }
```

4.36.2.81 void StDbTable::SetTable (char * *data*, int *nrows*, int * *idList* = 0) [virtual]

calloc'd version of data for StRoot

Definition at line 493 of file StDbTable.cc.

```
493                                                    {
494
495 if(mdata){
496   delete [] mdata;
497   mdata = 0;
498 }
499  if(!idList){
500    createMemory(nrows);
501  } else {
502    setElementID(idList,nrows); // createMemory is called here
503  }
504 int len = nrows*getTableSize();
505 memcpy(mdata,c,len);
506 mhasData=true;
507
508 }
```

4.36.2.82 void StDbTable::setTimeValues (unsigned int * *timeValues*) [inline, virtual]

Definition at line 358 of file StDbTable.h.

```
358                                                                    {
359   if(mtimeVals) delete [] mtimeVals;
360   mtimeVals=timeValues;
361 }
```

4.36.2.83 void StDbTable::StreamAccessor (StDbBufferI * *buff*, bool *isReading*) [virtual]

Definition at line 770 of file StDbTable.cc.

```
770                                                        {
771
772   bool ClientMode;
773   if(!(ClientMode=buff->IsClientMode()))buff->SetClientMode();
774
775   int rowID;
776   if(!melementID){
777      melementID = new int[mrows];
778      for(int i=0;i<mrows;i++)melementID[i]=i;
779   }
780
781   if(isReading){
782     buff->ReadScalar(rowID,"elementID");
783     melementID[mrowNumber]=rowID;
```

```

    if(mrowNumber==0){
     buff->ReadScalar(mschemaID,"schemaID");
     if(mversion) delete [] mversion;
     buff->ReadScalar(mversion,"version");
    } else {
      unsigned int bTime;// , eTime;
      buff->ReadScalar(bTime,"beginTime");
      if(bTime>mbeginTime.munixTime)mbeginTime.munixTime=bTime;
    }

  } else {

   buff->WriteScalar(mschemaID,"schemaID");
   buff->WriteScalar(mbeginTime.munixTime,"beginTime");
   if(mversion)buff->WriteScalar(mversion,"version");
   rowID = melementID[mrowNumber];
   buff->WriteScalar(rowID,"elementID");
  }

 if(!ClientMode)buff->SetStorageMode();  // reset to StorageMode

}
```

4.36.2.84 void StDbTable::StreamAccessor (typeAcceptor * *accept*, bool *isReading*) [virtual]

Definition at line 748 of file StDbTable.cc.

```
748                                                                  {
749
750   int len = 1;
751   accept->pass((char*)"schemaID",mschemaID,len);
752
753   if(isReading){
754     if(mbeginTime.mdateTime) delete [] mbeginTime.mdateTime;
755     if(mversion)delete [] mversion;
756     if(melementID)delete [] melementID;
757   } else {
758     if(!melementID){
759       melementID = new int[mrows];
760       for(int i=0;i<mrows;i++)melementID[i]=i;
761     }
762   }
763   accept->pass((char*)"beginTime",mbeginTime.mdateTime,len);
764   accept->pass((char*)"version",mversion,len);
765   accept->pass((char*)"elementID",melementID, mrows);
766 }
```

4.36.2.85 void StDbTable::WriteElement (char * *ptr*, char * *name*, int *length*, StTypeE *type*, StDbBuffer * *buff*) [protected, virtual]

Definition at line 1083 of file StDbTable.cc.

```
1083                                                                                                  {
1084
1085   switch (type) {
1086   case Stchar:
1087     {
1088     char* mchar = ptr;
1089     buff->WriteScalar(mchar,name);
```

```
    break;
    }
  case Stuchar:
    {
    unsigned char* muchar = (unsigned char*)ptr;
    buff->WriteArray(muchar,len,name);
    break;
    }
  case Stshort:
    {
    short* mshort = (short*) ptr;
    buff->WriteArray(mshort ,len,name);
    break;
    }
  case Stushort:
    {
    unsigned short* mushort = (unsigned short*) ptr;
    buff->WriteArray(mushort,len,name);
    break;
    }
  case Stint:
    {
    int* mint = (int*)ptr;
    buff->WriteArray(mint,len,name);
    break;
    }
  case Stuint:
    {
    unsigned int* muint = (unsigned int*) ptr;
    buff->WriteArray(muint,len,name);
    break;
    }
  case Stlong:
    {
    long* mlong = (long*) ptr;
    //if(len==1) cout << name << " = "<< *mlong << endl;
    buff->WriteArray(mlong,len,name);
    break;
    }
  case Stulong:
    {
    unsigned long* mulong = (unsigned long*) ptr;
    buff->WriteArray(mulong,len,name);
    break;
    }
  case Stlonglong:
    {
    long long* mlonglong = (long long*) ptr;
    buff->WriteArray(mlonglong,len,name);
    break;
    }
  case Stfloat:
    {
    float* mfloat = (float*) ptr;
    //if(len==1) cout << name << " = "<< *mfloat << endl;
    buff->WriteArray(mfloat,len,name);
    break;
    }
  case Stdouble:
    {
    double* mdouble = (double*) ptr;
    buff->WriteArray(mdouble,len,name);
    break;
    }
  }

}
```

4.36.3 Member Data Documentation

4.36.3.1 StDbTime StDbTable::mbeginTime [protected]

validity interval

Definition at line 161 of file StDbTable.h.

4.36.3.2 char* StDbTable::mdata [protected]

data & num of rows

Definition at line 184 of file StDbTable.h.

4.36.3.3 char* StDbTable::mdataTable [protected]

Definition at line 171 of file StDbTable.h.

4.36.3.4 bool StDbTable::mdefaultFlavor [protected]

Definition at line 157 of file StDbTable.h.

4.36.3.5 StTableDescriptorI* StDbTable::mdescriptor [protected]

Definition at line 181 of file StDbTable.h.

4.36.3.6 int* StDbTable::melementID [protected]

Definition at line 174 of file StDbTable.h.

4.36.3.7 char* StDbTable::melementName [protected]

Definition at line 167 of file StDbTable.h.

4.36.3.8 StDbTime StDbTable::mendStoreTime [protected]

Definition at line 163 of file StDbTable.h.

4.36.3.9 StDbTime StDbTable::mendTime [protected]

Definition at line 162 of file StDbTable.h.

4.36.3.10 char* StDbTable::mflavor [protected]

Definition at line 156 of file StDbTable.h.

4.36.3.11 bool StDbTable::mhasData [protected]

Definition at line 187 of file StDbTable.h.

4.36.3.12 bool StDbTable::mhasDescriptor [protected]

c-struct descriptor information

Definition at line 180 of file StDbTable.h.

4.36.3.13 bool StDbTable::mIsBaseLine [protected]

Definition at line 168 of file StDbTable.h.

4.36.3.14 bool StDbTable::mIsBinary [protected]

Definition at line 169 of file StDbTable.h.

4.36.3.15 bool StDbTable::mIsIndexed [protected]

Definition at line 170 of file StDbTable.h.

4.36.3.16 unsigned int StDbTable::mprodTime [protected]

Definition at line 158 of file StDbTable.h.

4.36.3.17 int StDbTable::mrowNumber [protected]

Definition at line 186 of file StDbTable.h.

4.36.3.18 int StDbTable::mrows [protected]

Definition at line 185 of file StDbTable.h.

4.36.3.19 int StDbTable::mrowsRequested [protected]

Definition at line 188 of file StDbTable.h.

4.36.3.20 int StDbTable::mschemaID [protected]

Definition at line 173 of file StDbTable.h.

4.36.3.21 StDbStoreInfo StDbTable::mstoredData [protected]

for rolling back stores

Definition at line 177 of file StDbTable.h.

4.36.3.22 char* StDbTable::mstructName [protected]

Definition at line 166 of file StDbTable.h.

4.36.3.23 unsigned int* StDbTable::mtimeVals [protected]

Definition at line 189 of file StDbTable.h.

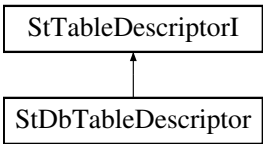
The documentation for this class was generated from the following files:

- StDbTable.h
- StDbTable.cc

4.37 StDbTableDescriptor Class Reference

`#include <StDbTableDescriptor.h>`

Inheritance diagram for StDbTableDescriptor::



Public Member Functions

- StDbTableDescriptor ()
- StDbTableDescriptor (int structID, int schemaID)
- StDbTableDescriptor (StDbTableDescriptor &d)
- virtual ~StDbTableDescriptor ()
- virtual void fillElement (StDbBuffer *buff, int tableID)
- virtual StTableDescriptorI * getCpy ()
- virtual tableDescriptor * getTableDescriptor () const
- virtual unsigned int getNumElements () const
- virtual unsigned int getTotalSizeInBytes () const
- virtual char * getElementName (int elementNum) const
- virtual unsigned int getElementOffset (int elementNum) const
- virtual unsigned int getElementSize (int elementNum) const
- virtual StTypeE getElementType (int elementNum) const
- virtual unsigned int * getElementDimensions (int elementNum) const
- virtual unsigned int getElementLength (int elementNum) const
- virtual unsigned int getElementNumDimensions (int elementNum) const
- virtual unsigned int getElementIndexLength (int elementNum, int dimensionNum) const
- int getSchemaID () const
- int getStructID () const
- void setSchemaID (int id)
- void setStructID (int id)
- bool IsValid () const
- int getCurrentInternalSize ()
- int getTrowSize ()
- void endRowPadding ()
- void storeRowSize (int rowSize)

Protected Member Functions

- void init ()
- virtual void reSize ()
- virtual void fillSizeAndOffset (char *length, int elementNum)
- virtual void fillLengths (char *length, int elementNum)
- virtual StTypeE getType (char *type)
- virtual unsigned int getSize (StTypeE type)

Protected Attributes

- unsigned int mnumElements
- unsigned int mtableSize
- int offsetToNextEmptyByte
- int offsetToLast4Bytes
- StTypeE lastType
- int padsize
- tableDescriptor * mcols
- int mMax
- int mCur
- int mstructID
- int mschemaID
- bool misValid
- bool mhasDouble

Static Protected Attributes

- int rowSizeTT = 0

4.37.1 Constructor & Destructor Documentation

4.37.1.1 StDbTableDescriptor::StDbTableDescriptor ()

Definition at line 137 of file StDbTableDescriptor.cc.

```
137 {  init(); }
```

4.37.1.2 StDbTableDescriptor::StDbTableDescriptor (int *structID*, int *schemaID*)

Definition at line 141 of file StDbTableDescriptor.cc.

```
141                                                                    {
142
143   init();
144   mstructID=structID;
145   mschemaID=schemaID;
146
147 };
```

4.37.1.3 StDbTableDescriptor::StDbTableDescriptor (StDbTableDescriptor & *d*)

Definition at line 172 of file StDbTableDescriptor.cc.

```
172                                                                    {
173
174 mCur = (int)d.getNumElements()-1;
175 mMax = d.getCurrentInternalSize();
176 offsetToNextEmptyByte = 0;
177 offsetToLast4Bytes = 0;
178 mcols = d.getTableDescriptor();
```

```
mnumElements=d.getNumElements();
mtableSize = d.getTotalSizeInBytes();
mschemaID=d.getSchemaID();
mstructID=d.getStructID();
misValid=d.IsValid();
mhasDouble=d.mhasDouble;

}
```

4.37.1.4 virtual StDbTableDescriptor::~StDbTableDescriptor () [inline, virtual]

Definition at line 111 of file StDbTableDescriptor.h.

```
{if(mcols) delete [] mcols; }
```

4.37.2 Member Function Documentation

4.37.2.1 void StDbTableDescriptor::endRowPadding ()

Definition at line 407 of file StDbTableDescriptor.cc.

```
                                        {

  //simple item for solaris & use of doubles & longlong.
  // if the struct contains such an entity, it's size must
  // be divisable by 8.

#ifndef __linux__
  if(mhasDouble){
     int checkPadding=mtableSize%8;
     if(checkPadding>0 && checkPadding<5)mtableSize+=4;
  }
#endif

};
```

4.37.2.2 void StDbTableDescriptor::fillElement (StDbBuffer * *buff*, int *tableID*) [virtual]

Definition at line 211 of file StDbTableDescriptor.cc.

```
                                                                {

  //bool ClientMode;
//  if(!(ClientMode=buff->IsClientMode()))buff->SetClientMode();
  int schemaID;
  if(tableID){  // mask off elements if tableID is non-zero
   if(!(buff->ReadScalar(schemaID,"schemaID") && (schemaID==tableID)) ){
       // skip this one
       return;
   }
  }

   reSize(); // increase array if needed
   int i = mCur;
   char* mtype = 0;
   char* mname = 0;
   buff->ReadScalar(mname,"name");
```

```
228    if(mname)strcpy(mcols[i].name,mname);
229    if(mname) delete [] mname;
230    if(buff->ReadScalar(mtype,"type")){
231        mcols[i].type = getType(mtype);
232        if(mtype)delete [] mtype;
233        char* length=0;
234        if(buff->ReadScalar(length,"length"))fillSizeAndOffset(length,i);
235        if(length) delete [] length;
236    }
237
238    if(mcols[i].type==Stlonglong || mcols[i].type==Stdouble ) mhasDouble=true;
239
240    mCur++;
241    mnumElements++;
242
243    // for multiple rows, add padding as needed at end of structure (tableSize)
244    int rowpad;
245    if((rowpad=offsetToNextEmptyByte%2))rowpad=2-rowpad;
246
247    mtableSize = offsetToNextEmptyByte+rowpad;
248
249    //if(!ClientMode)buff->SetStorageMode();  // reset to StorageMode
250
251  misValid=true;
252
253 }
```

4.37.2.3 void StDbTableDescriptor::fillLengths (char * *length*, int *elementNum*) [protected, virtual]

Definition at line 277 of file StDbTableDescriptor.cc.

```
277                                                                  {
278
279 int ip = 0;
280 int i=elementNum;
281 char* id= strstr(length,",");
282
283 // preset lengths to 1;
284
285   int k= (int)(sizeof(mcols[i].dimensionlen)/sizeof(ip));
286   for(int j=0;j<k;j++)mcols[i].dimensionlen[j]=1;
287
288  while (id && ip<3) {
289    id[0]='\0';
290    mcols[i].dimensionlen[ip] = atoi(length);
291    ip++;
292    id++;
293    length=id;
294    id= strstr(length,",");
295  }
296  mcols[i].dimensionlen[ip]=atoi(length);
297
298 }
```

4.37.2.4 void StDbTableDescriptor::fillSizeAndOffset (char * *length*, int *elementNum*) [protected, virtual]

Definition at line 303 of file StDbTableDescriptor.cc.

```
                                                                    {


  int i = elementNum;
  fillLengths(length,i);
  int space = 4-(offsetToNextEmptyByte-offsetToLast4Bytes);

  int j,jj;
  StTypeE type = mcols[i].type;
  j=0;
  mcols[i].size = getSize(mcols[i].type);
  int k= (int)(sizeof(mcols[i].dimensionlen)/sizeof(j));
  for(j=0;j<k;j++)mcols[i].size*=mcols[i].dimensionlen[j];

  if(offsetToLast4Bytes < 0){

    mcols[i].offset = 0;
    offsetToNextEmptyByte = mcols[i].offset+mcols[i].size;
    j = 4* ((int) floor ( (float) (mcols[i].size-1)/4 ));
    offsetToLast4Bytes = mcols[i].offset + j;// + 4;


  } else if(type==Stchar || type==Stuchar){

     mcols[i].offset=offsetToNextEmptyByte;
     offsetToNextEmptyByte = mcols[i].offset+mcols[i].size;
     jj=mcols[i].offset+mcols[i].size-offsetToLast4Bytes;
     if(jj>1)
     j = 4* ((int) floor ((float) (jj-1)/4 ));
     // j= number of bytes (in 4s) past last offset
      if(j)
      offsetToLast4Bytes = offsetToLast4Bytes+j;

  } else if( (space>=2) && (type == Stshort || type == Stushort) ){

      mcols[i].offset=offsetToLast4Bytes+2;
      j = 4* ((int) floor ((float) (mcols[i].size-2-1)/4 ));    // note the 2
      offsetToNextEmptyByte=mcols[i].offset+mcols[i].size;
     jj=mcols[i].offset+mcols[i].size-offsetToLast4Bytes;
     if(jj>1)
     j = 4* ((int) floor ((float) (jj-1)/4 ));
     // j= number of bytes (in 4s) past last offset
      if(j)
      offsetToLast4Bytes = offsetToLast4Bytes+j;

  } else {


    if( (type==Stdouble || type==Stlonglong) &&
        (lastType != Stdouble) && (lastType != Stlonglong) && padsize < 8){
      offsetToLast4Bytes+=padsize;
      offsetToNextEmptyByte+=padsize;
    }

     mcols[i].offset=offsetToLast4Bytes+4;
     offsetToNextEmptyByte = mcols[i].offset + mcols[i].size;
     j = 4* ((int) floor ((float) (mcols[i].size-1)/4 ));
     offsetToLast4Bytes = mcols[i].offset + j;// + 4;

  }

  if(offsetToLast4Bytes<0)offsetToLast4Bytes=0;

#ifdef __linux__
  lastType=Stdouble;
#else
  if(type==Stdouble || type==Stlonglong )padsize = 0;
```

```
  lastType=type;

  unsigned int onesize = getSize(mcols[i].type);
  for(j=0;j<k;j++){
    if(mcols[i].dimensionlen[j]==1 && j>0)continue;
    for(int jk=0; jk< (int)mcols[i].dimensionlen[j]; jk++){
      padsize=padsize+(int)onesize;
      if(padsize>=8)padsize=padsize-8;
    }
  }
 if(padsize>0 && padsize<=4) padsize=4;
 // padsize = 4* ((int) floor ((float) (padsize)/4 ));

#endif
}
```

4.37.2.5 StTableDescriptorI * StDbTableDescriptor::getCpy () [virtual]

Implements StTableDescriptorI.

Definition at line 202 of file StDbTableDescriptor.cc.

```
202                                {

StTableDescriptorI* dScr = new StDbTableDescriptor(*this);
return dScr;

}
```

4.37.2.6 int StDbTableDescriptor::getCurrentInternalSize () [inline]

Definition at line 207 of file StDbTableDescriptor.h.

```
207 { return mMax; };
```

4.37.2.7 unsigned int * StDbTableDescriptor::getElementDimensions (int *elementNum*) const [inline, virtual]

Implements StTableDescriptorI.

Definition at line 167 of file StDbTableDescriptor.h.

```
167                                                                {
return &mcols[elementNum].dimensionlen[0];
}
```

4.37.2.8 unsigned int StDbTableDescriptor::getElementIndexLength (int *elementNum*, int *dimensionNum*) const [inline, virtual]

Implements StTableDescriptorI.

Definition at line 197 of file StDbTableDescriptor.h.

```
197                                                                                {
return mcols[elementNum].dimensionlen[dimensionNum];
}
```

4.37.2.9 unsigned int StDbTableDescriptor::getElementLength (int *elementNum*) const [inline, virtual]

Implements StTableDescriptorI.

Definition at line 174 of file StDbTableDescriptor.h.

```
174                                                         {
175
176 int retVal=1;
177 int j;
178   int k= (int)(sizeof(mcols[elementNum].dimensionlen)/sizeof(j));
179   for(j=0;j<k;j++)retVal *= mcols[elementNum].dimensionlen[j];
180 return retVal;
181 }
```

4.37.2.10 char * StDbTableDescriptor::getElementName (int *elementNum*) const [inline, virtual]

Implements StTableDescriptorI.

Definition at line 147 of file StDbTableDescriptor.h.

```
147                                                       {
148 char* retVal = new char[strlen(mcols[elementNum].name)+1];
149 strcpy(retVal,mcols[elementNum].name);
150 return retVal;
151 }
```

4.37.2.11 unsigned int StDbTableDescriptor::getElementNumDimensions (int *elementNum*) const [inline, virtual]

Implements StTableDescriptorI.

Definition at line 186 of file StDbTableDescriptor.h.

```
186                                                             {
187 int retVal=1;
188 int j;
189   int k= (int)(sizeof(mcols[elementNum].dimensionlen)/sizeof(j));
190   for(j=0;j<k;j++){
191     if(mcols[elementNum].dimensionlen[j]>1)retVal=j+1;  // last dimension >1
192   }
193 return retVal;
194 }
```

4.37.2.12 unsigned int StDbTableDescriptor::getElementOffset (int *elementNum*) const [inline, virtual]

Implements StTableDescriptorI.

Definition at line 154 of file StDbTableDescriptor.h.

```
154                                                         {
155 return mcols[elementNum].offset;
156 }
```

4.37.2.13 unsigned int StDbTableDescriptor::getElementSize (int *elementNum*) const [inline, virtual]

Implements StTableDescriptorI.

Definition at line 158 of file StDbTableDescriptor.h.

```
158                                                                    {
159 return mcols[elementNum].size;
160 }
```

4.37.2.14 StTypeE StDbTableDescriptor::getElementType (int *elementNum*) const [inline, virtual]

Implements StTableDescriptorI.

Definition at line 162 of file StDbTableDescriptor.h.

```
162                                                                    {
163 return mcols[elementNum].type;
164 }
```

4.37.2.15 unsigned int StDbTableDescriptor::getNumElements () const [inline, virtual]

Implements StTableDescriptorI.

Definition at line 141 of file StDbTableDescriptor.h.

```
141 {return mnumElements;}
```

4.37.2.16 int StDbTableDescriptor::getSchemaID () const [inline]

Definition at line 202 of file StDbTableDescriptor.h.

```
202 { return mschemaID; }
```

4.37.2.17 unsigned int StDbTableDescriptor::getSize (StTypeE *type*) [inline, protected, virtual]

Definition at line 201 of file StDbTableDescriptor.h.

```
201 { return mycsize[type];}
```

4.37.2.18 int StDbTableDescriptor::getStructID () const [inline]

Definition at line 203 of file StDbTableDescriptor.h.

```
203 { return mstructID; }
```

4.37.2.19 tableDescriptor * StDbTableDescriptor::getTableDescriptor () const [virtual]

Definition at line 191 of file StDbTableDescriptor.cc.

```
191                                                 {
192
193   tableDescriptor* dScr = new tableDescriptor[mMax];
194   memset(dScr,0,(mMax)*sizeof(tableDescriptor));
195   memcpy(dScr,mcols,(mMax)*sizeof(tableDescriptor));
196
197 return dScr;
198 }
```

4.37.2.20 unsigned int StDbTableDescriptor::getTotalSizeInBytes () const [inline, virtual]

Implements StTableDescriptorI.

Definition at line 144 of file StDbTableDescriptor.h.

```
144 {return mtableSize;}
```

4.37.2.21 int StDbTableDescriptor::getTrowSize () [inline, virtual]

Implements StTableDescriptorI.

Definition at line 209 of file StDbTableDescriptor.h.

```
209 { return rowSizeTT; };
```

4.37.2.22 StTypeE StDbTableDescriptor::getType (char * *type*) [protected, virtual]

Definition at line 389 of file StDbTableDescriptor.cc.

```
389                                         {
390
391 StTypeE retVal=Stchar;
392
393 //char* typenames[] = {"Stchar","Stuchar","Stshort","Stushort","Stint","Stuint","Stlong","Stulong","St
394 const char* typenames[] = {"char","uchar","short","ushort","int","uint","long","ulong","longlong","flo
395
396  for(int i=0; i<12;i++){
397    if(strcmp(type,typenames[i])==0){
398      retVal=(StTypeE)i;
399      break;
400    }
401  }
402
403 return retVal;
404 }
```

4.37.2.23 void StDbTableDescriptor::init () [protected]

Definition at line 152 of file StDbTableDescriptor.cc.

```
152                         {
153
154 mtableSize=0;
155 mCur = 0;
156 mMax = 100;
157 offsetToNextEmptyByte = 0;
158 offsetToLast4Bytes = -4;
159 mnumElements = 0;
160 lastType=Stdouble;
161 padsize = 0;
162 mcols = new tableDescriptor[mMax];
163 memset(mcols,0,mMax*sizeof(tableDescriptor));
164 mschemaID=mstructID=0;
165 misValid=false;
166 mhasDouble=false;
167
168 }
```

4.37.2.24 bool StDbTableDescriptor::IsValid () const [inline]

Definition at line 206 of file StDbTableDescriptor.h.

```
206 { return misValid; }
```

4.37.2.25 void StDbTableDescriptor::reSize () [protected, virtual]

Definition at line 259 of file StDbTableDescriptor.cc.

```
259                           {
260
261   // simply add 10 elements
262
263   if(mCur<mMax) return;
264
265   int newMax = mMax+10;
266   tableDescriptor* dScr = new tableDescriptor[newMax];
267   memcpy(dScr,mcols,(mMax)*sizeof(tableDescriptor));
268   if(mcols)delete [] mcols;
269   mcols=dScr;
270   mMax=newMax;
271
272 }
```

4.37.2.26 void StDbTableDescriptor::setSchemaID (int *id*) [inline]

Definition at line 204 of file StDbTableDescriptor.h.

```
204 { mschemaID=id; }
```

4.37.2.27 void StDbTableDescriptor::setStructID (int *id*) [inline]

Definition at line 205 of file StDbTableDescriptor.h.

```
205 { mstructID=id; }
```

4.37.2.28 void StDbTableDescriptor::storeRowSize (int *rowSize*) [inline]

Definition at line 208 of file StDbTableDescriptor.h.

```
208 { rowSizeTT = rowSize; }
```

4.37.3 Member Data Documentation

4.37.3.1 StTypeE StDbTableDescriptor::lastType [protected]

Definition at line 84 of file StDbTableDescriptor.h.

4.37.3.2 tableDescriptor* StDbTableDescriptor::mcols [protected]

Definition at line 87 of file StDbTableDescriptor.h.

4.37.3.3 int StDbTableDescriptor::mCur [protected]

Definition at line 89 of file StDbTableDescriptor.h.

4.37.3.4 bool StDbTableDescriptor::mhasDouble [protected]

Definition at line 97 of file StDbTableDescriptor.h.

4.37.3.5 bool StDbTableDescriptor::misValid [protected]

Definition at line 96 of file StDbTableDescriptor.h.

4.37.3.6 int StDbTableDescriptor::mMax [protected]

Definition at line 88 of file StDbTableDescriptor.h.

4.37.3.7 unsigned int StDbTableDescriptor::mnumElements [protected]

Definition at line 80 of file StDbTableDescriptor.h.

4.37.3.8 int StDbTableDescriptor::mschemaID [protected]

Definition at line 93 of file StDbTableDescriptor.h.

4.37.3.9 int StDbTableDescriptor::mstructID [protected]

Definition at line 92 of file StDbTableDescriptor.h.

4.37.3.10 unsigned int StDbTableDescriptor::mtableSize [protected]

Definition at line 81 of file StDbTableDescriptor.h.

4.37.3.11 int StDbTableDescriptor::offsetToLast4Bytes [protected]

Definition at line 83 of file StDbTableDescriptor.h.

4.37.3.12 int StDbTableDescriptor::offsetToNextEmptyByte [protected]

Definition at line 82 of file StDbTableDescriptor.h.

4.37.3.13 int StDbTableDescriptor::padsize [protected]

Definition at line 85 of file StDbTableDescriptor.h.

4.37.3.14 int StDbTableDescriptor::rowSizeTT = 0 [static, protected]

Definition at line 133 of file StDbTableDescriptor.cc.

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

- StDbTableDescriptor.h
- StDbTableDescriptor.cc

4.38 StDbTableFactory Class Reference

```
#include <StDbTableFactory.hh>
```

Public Member Functions

- virtual ~StDbTableFactory ()
- virtual StDbTable * newDbTable (const char *dbName, const char *tabName)

4.38.1 Constructor & Destructor Documentation

4.38.1.1 virtual StDbTableFactory::~StDbTableFactory () [inline, virtual]

Definition at line 42 of file StDbTableFactory.hh.

```
42 {};
```

4.38.2 Member Function Documentation

4.38.2.1 StDbTable * StDbTableFactory::newDbTable (const char * *dbName*, const char * *tabName*) [virtual]

Definition at line 36 of file StDbTableFactory.cc.

```
36                                                                  {
37   return new StDbTable(tabName);
38 }
```

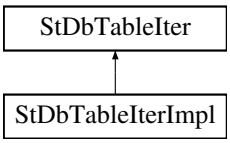
The documentation for this class was generated from the following files:

- StDbTableFactory.hh
- StDbTableFactory.cc

4.39 StDbTableIter Class Reference

`#include <StDbTableIter.hh>`

Inheritance diagram for StDbTableIter::



Public Member Functions

- StDbTableIter ()
- virtual ~StDbTableIter ()
- virtual StDbTable * next ()=0
- virtual StDbTable * operator++ ()=0
- virtual bool done ()=0

4.39.1 Constructor & Destructor Documentation

4.39.1.1 StDbTableIter::StDbTableIter () [inline]

Definition at line 59 of file StDbTableIter.hh.

```
59 {};
```

4.39.1.2 virtual StDbTableIter::~StDbTableIter () [inline, virtual]

Definition at line 60 of file StDbTableIter.hh.

```
60 {};
```

4.39.2 Member Function Documentation

4.39.2.1 virtual bool StDbTableIter::done () [pure virtual]

Implemented in StDbTableIterImpl.

4.39.2.2 virtual StDbTable* StDbTableIter::next () [pure virtual]

Implemented in StDbTableIterImpl.

4.39.2.3 virtual StDbTable* StDbTableIter::operator++ () [pure virtual]

Implemented in StDbTableIterImpl.

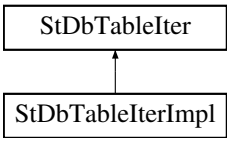
The documentation for this class was generated from the following file:

- StDbTableIter.hh

4.40 StDbTableIterImpl Class Reference

`#include <StDbTableIterImpl.hh>`

Inheritance diagram for StDbTableIterImpl::



Public Member Functions

- StDbTableIterImpl ()
- StDbTableIterImpl (StDbConfigNodeImpl *node)
- virtual ~StDbTableIterImpl ()
- void init (StDbConfigNodeImpl *node)
- virtual StDbTable * next ()
- virtual StDbTable * operator++ ()
- virtual bool done ()

Private Attributes

- ListIter itr
- StDbConfigNodeImpl * mnode

4.40.1 Constructor & Destructor Documentation

4.40.1.1 StDbTableIterImpl::StDbTableIterImpl () `[inline]`

Definition at line 43 of file StDbTableIterImpl.hh.

```
43 : mnode(0){};
```

4.40.1.2 StDbTableIterImpl::StDbTableIterImpl (StDbConfigNodeImpl * *node*) `[inline]`

Definition at line 44 of file StDbTableIterImpl.hh.

```
44 { init(node);};
```

4.40.1.3 virtual StDbTableIterImpl::~StDbTableIterImpl () `[inline, virtual]`

Definition at line 45 of file StDbTableIterImpl.hh.

```
45 {};
```

4.40.2 Member Function Documentation

4.40.2.1 bool StDbTableIterImpl::done () [virtual]

Implements StDbTableIter.

Definition at line 57 of file StDbTableIterImpl.cc.

```
57 {return (itr != mnode->mTables.end()) ? false : true ; }
```

4.40.2.2 void StDbTableIterImpl::init (StDbConfigNodeImpl * *node*)

Definition at line 37 of file StDbTableIterImpl.cc.

```
37                                                    {
38    mnode=node;
39    itr = mnode->mTables.begin();
40 }
```

4.40.2.3 StDbTable * StDbTableIterImpl::next () [virtual]

Implements StDbTableIter.

Definition at line 44 of file StDbTableIterImpl.cc.

```
44                                    {
45  StDbTable* ret = 0;
46  if(done()) return ret;
47  ret = *itr; itr++;
48 return ret;
49 }
```

4.40.2.4 StDbTable * StDbTableIterImpl::operator++ () [virtual]

Implements StDbTableIter.

Definition at line 53 of file StDbTableIterImpl.cc.

```
53 { return next(); }
```

4.40.3 Member Data Documentation

4.40.3.1 ListIter StDbTableIterImpl::itr [private]

Definition at line 38 of file StDbTableIterImpl.hh.

4.40.3.2 StDbConfigNodeImpl* StDbTableIterImpl::mnode [private]

Definition at line 39 of file StDbTableIterImpl.hh.

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

- StDbTableIterImpl.hh
- StDbTableIterImpl.cc

4.41 StDbTime Class Reference

```
#include <StDbTime.h>
```

Public Member Functions

- StDbTime ()
- StDbTime (unsigned int utime)
- StDbTime (const char *dtime)
- StDbTime (StDbTime &time)
- ~StDbTime ()
- void setUnixTime (unsigned int utime)
- void setDateTime (const char *dtime)
- void setTime (unsigned int utime, const char *dtime)
- unsigned int getUnixTime ()
- char * getDateTime ()

Public Attributes

- unsigned int munixTime
- char * mdateTime

4.41.1 Constructor & Destructor Documentation

4.41.1.1 StDbTime::StDbTime () [inline]

Definition at line 56 of file StDbTime.h.

```
56 : munixTime(0), mdateTime(0) {};
```

4.41.1.2 StDbTime::StDbTime (unsigned int *utime*) [inline]

Definition at line 57 of file StDbTime.h.

```
57 : mdateTime(0) { munixTime = utime;};
```

4.41.1.3 StDbTime::StDbTime (const char * *dtime*) [inline]

Definition at line 59 of file StDbTime.h.

```
59                              : munixTime(0) { if(dtime){
60                                          mdateTime=new char[strlen(dtime)+1];
61                                          strcpy(mdateTime,dtime);
62                                          } else {mdateTime=0;}};
```

4.41.1.4 StDbTime::StDbTime (StDbTime & *time*) [inline]

Definition at line 64 of file StDbTime.h.

```
64                              { munixTime=0; mdateTime=0;
65   if(time.mdateTime){
66     mdateTime=new char[strlen(time.mdateTime)+1];
67     strcpy(mdateTime,time.mdateTime);
68   }
69   munixTime=time.munixTime;
70   }
```

4.41.1.5 StDbTime::~StDbTime () [inline]

Definition at line 71 of file StDbTime.h.

```
71 { if(mdateTime) delete [] mdateTime; }
```

4.41.2 Member Function Documentation

4.41.2.1 char* StDbTime::getDateTime () [inline]

Definition at line 83 of file StDbTime.h.

```
83 { return mdateTime; }
```

4.41.2.2 unsigned int StDbTime::getUnixTime () [inline]

Definition at line 82 of file StDbTime.h.

```
82 { return munixTime; }
```

4.41.2.3 void StDbTime::setDateTime (const char * *dtime*) [inline]

Definition at line 74 of file StDbTime.h.

```
74                                      { if(!dtime) return;
75                                  if(mdateTime) delete [] mdateTime;
76                                  mdateTime=new char[strlen(dtime)+1];
77                                  strcpy(mdateTime,dtime); };
```

4.41.2.4 void StDbTime::setTime (unsigned int *utime*, const char * *dtime*) [inline]

Definition at line 79 of file StDbTime.h.

```
79                                                      {
80                                                  setUnixTime(utime);
81                                                  setDateTime(dtime);}
```

4.41.2.5 void StDbTime::setUnixTime (unsigned int *utime*) [inline]

Definition at line 73 of file StDbTime.h.

```
73 { munixTime = utime;}
```

4.41.3 Member Data Documentation

4.41.3.1 char* StDbTime::mdateTime

Definition at line 54 of file StDbTime.h.

4.41.3.2 unsigned int StDbTime::munixTime

Definition at line 53 of file StDbTime.h.

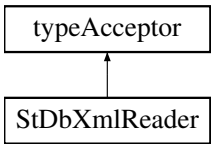
The documentation for this class was generated from the following file:

- StDbTime.h

4.42 StDbXmlReader Class Reference

`#include <StDbXmlReader.h>`

Inheritance diagram for StDbXmlReader::



Public Member Functions

- StDbXmlReader ()
- virtual ~StDbXmlReader ()
- void readTable (ifstream &is)
- virtual void pass (char *name, short &i, int &len)
- virtual void pass (char *name, int &i, int &len)
- virtual void pass (char *name, long &i, int &len)
- virtual void pass (char *name, unsigned short &i, int &len)
- virtual void pass (char *name, unsigned int &i, int &len)
- virtual void pass (char *name, unsigned long &i, int &len)
- virtual void pass (char *name, long long &i, int &len)
- virtual void pass (char *name, float &i, int &len)
- virtual void pass (char *name, double &i, int &len)
- virtual void pass (char *name, char *&i, int &len)
- virtual void pass (char *name, unsigned char &i, int &len)
- virtual void pass (char *name, unsigned char *&i, int &len)
- virtual void pass (char *name, short *&i, int &len)
- virtual void pass (char *name, int *&i, int &len)
- virtual void pass (char *name, long *&i, int &len)
- virtual void pass (char *name, unsigned short *&i, int &len)
- virtual void pass (char *name, unsigned int *&i, int &len)
- virtual void pass (char *name, unsigned long *&i, int &len)
- virtual void pass (char *name, long long *&i, int &len)
- virtual void pass (char *name, float *&i, int &len)
- virtual void pass (char *name, double *&i, int &len)

Protected Member Functions

- void buildDbTable ()
- void buildStruct ()
- void fillElements (accessor *a)
- elem * findElement (char *name)

Protected Attributes

- char * loca [20024]
- dbTable * tab
- int maxlines

4.42.1 Constructor & Destructor Documentation

4.42.1.1 StDbXmlReader::StDbXmlReader ()

Definition at line 87 of file StDbXmlReader.cc.

```
87 { tab = new dbTable(); }
```

4.42.1.2 StDbXmlReader::~StDbXmlReader () [virtual]

Definition at line 91 of file StDbXmlReader.cc.

```
91 { delete tab; }
```

4.42.2 Member Function Documentation

4.42.2.1 void StDbXmlReader::buildDbTable () [protected]

Definition at line 182 of file StDbXmlReader.cc.

```
182                             {
183
184  int done = 0;
185  int j=0;
186
187  // find table start
188  while (!done) {
189    if(strstr(loca[j],tab->startKey)){
190      tab->istart = j;
191      done = 1;
192    }
193    j++;
194  }
195
196  // find accessor start
197  j=tab->istart+1;
198  done = 0;
199  while (!done) {
200    //   cout << loca[j] << endl;
201    if(strstr(loca[j],(tab->a).startKey)){
202      (tab->a).istart = j;
203      done = 1;
204    }
205    j++;
206  }
207
208  // if(j>=maxlines)cout << "1 j to big " << j << " max= " << maxlines << endl;
209
210  elem* e0;
211  elem *e = new elem(); // key holder
212  // (tab->a).e.push_back(e)
213  done = 0;
214  int nelems = 0;
215  j=(tab->a).istart+1;
216  // if(j>=maxlines)cout << "2 j to big " << j << " max= " << maxlines << endl;
217  while (!done) {
218    if(strstr(loca[j],e->startKey)){
219      e0 = new elem();
220      e0->istart = j;
```

```
      (tab->a).e.push_back(e0);
      nelems++;
      if(strstr(loca[j],e->endKey))e0->iend = j;
    } else if(strstr(loca[j],e->endKey)){
      e0->iend = j;
    } else if(strstr(loca[j],(tab->a).endKey)){
      if(strstr(loca[j],e->endKey))e0->iend = j;
      (tab->a).iend = j;
      (tab->a).nelems = nelems;
      done = 1;
    }
    j++;
  }

  // end of accessor
  // find  all rows
  j=(tab->a).iend+1;
  // if(j>=maxlines)cout << "3 j to big " << j << " max= " << maxlines << endl;
  dbRow row;
  dbRow* row0;

  done = 0;
  int numRows = 0;
  while (!done) {
    if(strstr(loca[j],row.startKey)){
        row0 = new dbRow();
        row0->istart = j;
        if(strstr(loca[j],row.endKey))row0->iend = j;
        tab->row.push_back(row0);
        row0->rowNumber = numRows;
        numRows++;
      } else if(strstr(loca[j],tab->endKey)){
        tab->iend = j;
        tab->numRows = numRows;
        tab->curRow = -1;
        done = 1;
      }
    j++;
  }

  // cout << "Numrows = " << numRows<< endl;
  // cout << "Current Row = " << tab->curRow << endl;
    //
    // Loop over rows
    //
  for(int nrows=0; nrows<tab->numRows; nrows++){
  done = 0;
  nelems = 0;
  j=tab->row[nrows]->istart;
  while (!done) {
    //   cout << loca[j] << endl;
    if(strstr(loca[j],e->startKey)){
      e0 = new elem();
      e0->istart = j;
      tab->row[nrows]->e.push_back(e0);
      nelems++;
      if(strstr(loca[j],e->endKey))e0->iend = j;
      //     cout<<"estart"<< loca[j] <<endl;
    } else if(strstr(loca[j],e->endKey)){
      e0->iend = j;
      //     cout<<"eend"<< loca[j] <<endl;
    } else if(strstr(loca[j],row.endKey)){
      tab->row[nrows]->nelems = nelems;
      done = 1;
      //     cout<<"rowend"<< loca[j] <<endl;
    }
    j++;
```

```
288  }
289  }
290
291 buildStruct();
292
293 }
```

4.42.2.2 void StDbXmlReader::buildStruct () [protected]

Definition at line 298 of file StDbXmlReader.cc.

```
298                          {
299
300  char * p1 = strstr(loca[tab->istart],tab->startKey);
301  int len = strlen(tab->startKey);
302  int k;
303  for(k=0;k<len+1;k++)p1++;
304
305  char* tmpName = new char[strlen(p1)+1];
306  strcpy(tmpName,p1);
307  char* id;
308
309  if((id=strstr(tmpName,"<")))*id='\0';
310  tab->name=new char[strlen(tmpName)+1];
311  strcpy(tab->name,tmpName);
312  delete [] tmpName;
313
314  accessor * a = &(tab->a);
315  fillElements(a);
316  // a = (accessor*)(tab);
317
318  len = strlen(tab->row[0]->startKey);
319
320  for(int n=0; n<tab->numRows;n++){
321
322    // 1st get rowID (elementID) of this row
323    p1 = strstr(loca[tab->row[n]->istart],tab->row[n]->startKey);
324    for(k=0;k<len+1;k++)p1++;
325    tmpName = new char[strlen(p1)+1];
326    strcpy(tmpName,p1);
327    if((id=strstr(tmpName,"<")))*id='\0';
328    tab->row[n]->rowID = atoi(tmpName);
329    delete [] tmpName;
330
331    // now get row values
332
333     a = (accessor*)(tab->row[n]);
334     fillElements(a);
335
336  }
337
338 }
```

4.42.2.3 void StDbXmlReader::fillElements (accessor * a) [protected]

Definition at line 343 of file StDbXmlReader.cc.

```
343                                        {
344
345 elem* e;
346 int len,j;
```

```
char *p1,*p2,*hlen;
int ilist;
int iline;

 for(int k=0;k<a->nelems;k++){

   e=a->e[k];

     // get Type

   p1=strstr(loca[e->istart],e->startKey) + strlen(e->startKey);

   p2=strstr(loca[e->istart],">");
   len = strlen(p1) - strlen(p2);
   e->type = new char[len];
   memcpy(e->type,p1,len);
   e->type[len]='\0';

   // get Name
   p2++;
   while(p2[0]==' ')p2++;
   p1 = strstr(p2,"<");
   len = strlen(p2) - strlen(p1);
   e->name = new char[len];
   memcpy(e->name,p2,len);
   e->name[len]='\0';

   // get size

   p1=strstr(loca[e->istart],e->size.startKey);

if(p1){
   p1+=strlen(e->size.startKey);
   p2=strstr(loca[e->istart],e->size.endKey);
   len = strlen(p1) - strlen(p2)+1;
   hlen = new char[len];
   memcpy(hlen,p1,len);
   hlen[len-1]='\0';
   e->size.isize = atoi(hlen);
}
 if(e->size.isize==0)e->size.isize=1; // <length> not required for len=1


   // get values : if array it is on separate lines

   p1=strstr(loca[e->istart],(e->val).startKey);
   if(p1){
     // <element Type> ...     <value> data </value>
     // </element Type>
      p1+=strlen((e->val).startKey);
      while(p1[0]==' ')p1++;
      p2=strstr(loca[e->istart],(e->val).endKey);
      len = strlen(p1) - strlen(p2);
      (e->val).data = new char[len];
      memcpy((e->val).data,p1,len);
      (e->val).data[len]='\0';
   } else {
     // <element Type> ...
     // <value>
     // data
     // data
     // </value> </element Type>
     ilist =e->iend - e->istart;
     iline =0;
     for(j=2;j<ilist;j++)iline += strlen(loca[e->istart+j]);
     StString fs;
     for(j=2;j<ilist;j++) fs<<loca[e->istart+j]; //<<endl;
```

```
      string fs2=fs.str();
      e->val.data = new char[fs2.length()+1];
      //cout << "TEST:: " << fullLine << " len = " << strlen(fullLine) << endl;
      strcpy(e->val.data,fs2.c_str());

    }

    //   cout << "Test:: " << endl;
    // cout << e->name << " "<<e->type << " " << e->val.data << endl;
  }

}
```

4.42.2.4 elem * StDbXmlReader::findElement (char * *name*) [protected]

Definition at line 95 of file StDbXmlReader.cc.

```
95                                      {
96
97 elem* e = 0;
98 elem* retVal = 0;
99
100 int k;
101 // check accessor 1st
102 // here it's a bit scary : if accessor element has same name as data-element
103  if(tab->curRow==-1){
104  for(k=0;k<tab->a.nelems;k++){
105     e=tab->a.e[k];
106     if(strcmp(name,e->name)==0){
107       retVal=e;
108       break;
109     }
110  }
111  }
112
113  if(!retVal){ // now check rows
114    if(tab->curRow==-1)tab->curRow=0; // init row
115    if(tab->curRow==tab->numRows)return retVal;
116   for(k=0;k<tab->row[tab->curRow]->nelems;k++){
117    //   cout <<k<<" elements = " << tab->nelems << endl;
118     e=tab->row[tab->curRow]->e[k];
119     if(strcmp(name,e->name)==0){
120       retVal=e;
121       break;
122     }
123  }
124  if(k==tab->row[tab->curRow]->nelems-1)tab->curRow++;
125  }
126
127 return retVal;
128 }
```

4.42.2.5 void StDbXmlReader::pass (char * *name*, double *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 602 of file StDbXmlReader.cc.

```
602                                                     {
603
604  elem* e = findElement(name); if(!e){ cerr<<name<<" not found"<<endl; return;}
```

```
605  if(!strstr(e->type,"Double"))return;
606  passAux(e,i,len);
607 }
```

4.42.2.6 void StDbXmlReader::pass (char * *name*, float *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 593 of file StDbXmlReader.cc.

```
593                                                    {
594
595  elem* e = findElement(name); if(!e){ cerr<<name<<" not found"<<endl; return;}
596  if(!strstr(e->type,"Float")) return;
597  passAux(e,i,len);
598 }
```

4.42.2.7 void StDbXmlReader::pass (char * *name*, long long *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 573 of file StDbXmlReader.cc.

```
573                                                        {
574   //
575   cout<<"Not Yet Implemented"<<endl;
576 }
```

4.42.2.8 void StDbXmlReader::pass (char * *name*, unsigned long *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 582 of file StDbXmlReader.cc.

```
582                                                          {
583
584  elem* e = findElement(name); if(!e){ cerr<<name<<" not found"<<endl; return;}
585  if(!strstr(e->type,"ULong"))return;
586  passAux(e,i,len);
587 }
```

4.42.2.9 void StDbXmlReader::pass (char * *name*, unsigned int *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 527 of file StDbXmlReader.cc.

```
527                                                          {
528
529  elem* e = findElement(name); if(!e){ cerr<<name<<" not found"<<endl; return;}
530  if(!strstr(e->type,"UInt"))return;
531  passAux(e,i,len);
532 }
```

4.42.2.10 void StDbXmlReader::pass (char * *name*, unsigned short *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 493 of file StDbXmlReader.cc.

```
493                                                                 {
494
495  elem* e = findElement(name); if(!e){ cerr<<name<<" not found"<<endl; return;}
496  if(!strstr(e->type,"UShort"))return;
497  passAux(e,i,len);
498 }
```

4.42.2.11 void StDbXmlReader::pass (char * *name*, long *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 546 of file StDbXmlReader.cc.

```
546                                                         {
547
548  elem* e = findElement(name); if(!e){ cerr<<name<<" not found"<<endl; return;}
549  if(!strstr(e->type,"Long")) return;
550  passAux(e,i,len);
551 }
```

4.42.2.12 void StDbXmlReader::pass (char * *name*, int *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 511 of file StDbXmlReader.cc.

```
511                                                         {
512
513   //  cout << "In IntArray " << endl;
514  elem* e = findElement(name); if(!e){ cerr<<name<<" not found"<<endl; return;}
515  if(!strstr(e->type,"Int")) return;
516  passAux(e,i,len);
517 }
```

4.42.2.13 void StDbXmlReader::pass (char * *name*, short *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 473 of file StDbXmlReader.cc.

```
473                                                         {
474
475  elem* e = findElement(name); if(!e){ cerr<<name<<" not found"<<endl; return;}
476  if(!strstr(e->type,"Short"))return;
477  passAux(e,i,len);
478 }
```

4.42.2.14 void StDbXmlReader::pass (char * *name*, unsigned char *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 450 of file StDbXmlReader.cc.

```
451 {
452  elem* e = findElement(name); if(!e){ cerr<<name<<" not found"<<endl; return;}
453  if(!strstr(e->type,"UChar")) return;
454  passAux(e,i,len);
455 }
```

4.42.2.15 void StDbXmlReader::pass (char * *name*, unsigned char & *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 442 of file StDbXmlReader.cc.

```
442                                                                 {
443  elem* e = findElement(name); if(!e){ cerr<<name<<" not found"<<endl; return;}
444  if(strcmp(e->type,"Char")==0)i=atoi((e->val).data);
445 }
```

4.42.2.16 void StDbXmlReader::pass (char * *name*, char *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 431 of file StDbXmlReader.cc.

```
431                                                    {
432  elem* e = findElement(name); if(!e){ cerr<<name<<" not found"<<endl; return;}
433  if(strcmp(e->type,"String")==0){
434    i=new char[strlen((e->val).data)+1];
435    strcpy(i,(e->val).data);
436   }
437 }
```

4.42.2.17 void StDbXmlReader::pass (char * *name*, double & *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 619 of file StDbXmlReader.cc.

```
619                                                     {
620  elem* e = findElement(name); if(!e){ cerr<<name<<" not found"<<endl; return;}
621  if(strcmp(e->type,"Double")==0)i=atof((e->val).data);
622
623 }
```

4.42.2.18 void StDbXmlReader::pass (char * *name*, float & *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 611 of file StDbXmlReader.cc.

```
611                                                    {
612  elem* e = findElement(name); if(!e){ cerr<<name<<" not found"<<endl; return;}
613  if(strcmp(e->type,"Float")==0)i=(float)atof((e->val).data);
614
615 }
```

4.42.2.19 void StDbXmlReader::pass (char * *name*, long long & *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 564 of file StDbXmlReader.cc.

```
564                                                        {
565  elem* e = findElement(name); if(!e){ cerr<<name<<" not found"<<endl; return;}
566 #ifndef __osf__
567  if(strcmp(e->type,"LongLong")==0)i=atoll((e->val).data);
568 #else
569  if(strcmp(e->type,"LongLong")==0)i=atol((e->val).data);
570 #endif
571 }
```

4.42.2.20 void StDbXmlReader::pass (char * *name*, unsigned long & *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 556 of file StDbXmlReader.cc.

```
556                                                          {
557  elem* e = findElement(name); if(!e){ cerr<<name<<" not found"<<endl; return;}
558  if(strcmp(e->type,"ULong")==0)i=atol((e->val).data);
559 }
```

4.42.2.21 void StDbXmlReader::pass (char * *name*, unsigned int & *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 519 of file StDbXmlReader.cc.

```
519                                                        {
520  elem* e = findElement(name); if(!e){ cerr<<name<<" not found"<<endl; return;}
521  if(strcmp(e->type,"UInt")==0)i=atoi((e->val).data);
522 }
```

4.42.2.22 void StDbXmlReader::pass (char * *name*, unsigned short & *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 484 of file StDbXmlReader.cc.

```
484                                                          {
485  elem* e = findElement(name); if(!e){ cerr<<name<<" not found"<<endl; return;}
486  if(strcmp(e->type,"UShort")==0)i=atoi((e->val).data);
487 }
```

4.42.2.23 void StDbXmlReader::pass (char * *name*, long & *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 537 of file StDbXmlReader.cc.

```
537                                                 {
538  elem* e = findElement(name); if(!e){ cerr<<name<<" not found"<<endl; return;}
539  if(strcmp(e->type,"Long")==0)i=atol((e->val).data);
540 }
```

4.42.2.24 void StDbXmlReader::pass (char * *name*, int & *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 501 of file StDbXmlReader.cc.

```
501                                                 {
502   //  cout << "In Int " << endl;
503
504  elem* e = findElement(name); if(!e){ cerr<<name<<" not found"<<endl; return;}
505  if(strcmp(e->type,"Int")==0)i=atoi((e->val).data);
506 }
```

4.42.2.25 void StDbXmlReader::pass (char * *name*, short & *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 464 of file StDbXmlReader.cc.

```
464                                                 {
465  elem* e = findElement(name); if(!e){ cerr<<name<<" not found"<<endl; return;}
466  if(strcmp(e->type,"Short")==0)i=atoi((e->val).data);
467 }
```

4.42.2.26 void StDbXmlReader::readTable (ifstream & *is*)

Definition at line 135 of file StDbXmlReader.cc.

```
135                                     {
136
137  int done = 0;
138  int i = 0;
139  int j,icount,k;
140  char tmp[2];
141  char line[256];
142  char line2[256];
143
144  while(!done){
145    if(is.eof()){
146      done = 1;
147    } else {
148      //     cout << "line . " << i << endl;
149      is.getline(line,255);
150      k = strlen(line);
151      j = 0;
```

```
      line2[j]='\0';
      char* ptr=&line2[0];
      for(icount=0;icount<k;icount++){
        if(!(line[icount]==' ')){
           *tmp=line[icount];
           tmp[1]='\0';
           j++;
           strncpy(ptr,tmp,1);
           ptr++;
        }
      }
      line2[j]='\0';
      //     cout << line << endl;
      loca[i] = new char[strlen(line2)+1];
      strcpy(loca[i],line2);
      //     cout << "line i = "<< loca[i] << endl;
      i++;
    }
  }

maxlines = i-1;

buildDbTable();

}
```

4.42.3 Member Data Documentation

4.42.3.1 char* StDbXmlReader::loca[20024] [protected]

Definition at line 66 of file StDbXmlReader.h.

4.42.3.2 int StDbXmlReader::maxlines [protected]

Definition at line 68 of file StDbXmlReader.h.

4.42.3.3 dbTable* StDbXmlReader::tab [protected]

Definition at line 67 of file StDbXmlReader.h.

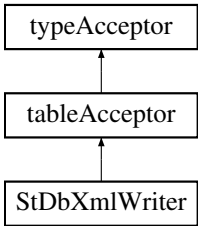
The documentation for this class was generated from the following files:

- StDbXmlReader.h
- StDbXmlReader.cc

4.43 StDbXmlWriter Class Reference

`#include <StDbXmlWriter.h>`

Inheritance diagram for StDbXmlWriter::



Public Member Functions

- StDbXmlWriter ()
- StDbXmlWriter (ostream &ofs)
- virtual ~StDbXmlWriter ()
- virtual void streamHeader (const char *name)
- virtual void streamTableName (const char *name)
- virtual void streamEndTableName ()
- virtual void streamAccessor ()
- virtual void endAccessor ()
- virtual void streamRow (int row)
- virtual void streamEndRow ()
- virtual void streamTail ()
- virtual void ioTable (StDbTable *table)
- virtual void pass (char *name, short &i, int &len)
- virtual void pass (char *name, int &i, int &len)
- virtual void pass (char *name, long &i, int &len)
- virtual void pass (char *name, unsigned short &i, int &len)
- virtual void pass (char *name, unsigned int &i, int &len)
- virtual void pass (char *name, unsigned long &i, int &len)
- virtual void pass (char *name, long long &i, int &len)
- virtual void pass (char *name, float &i, int &len)
- virtual void pass (char *name, double &i, int &len)
- virtual void pass (char *name, char *&i, int &len)
- virtual void pass (char *name, unsigned char &i, int &len)
- virtual void pass (char *name, unsigned char *&i, int &len)
- virtual void pass (char *name, short *&i, int &len)
- virtual void pass (char *name, int *&i, int &len)
- virtual void pass (char *name, long *&i, int &len)
- virtual void pass (char *name, unsigned short *&i, int &len)
- virtual void pass (char *name, unsigned int *&i, int &len)
- virtual void pass (char *name, unsigned long *&i, int &len)
- virtual void pass (char *name, long long *&i, int &len)
- virtual void pass (char *name, float *&i, int &len)
- virtual void pass (char *name, double *&i, int &len)

Protected Attributes

- ostream * os

4.43.1 Constructor & Destructor Documentation

4.43.1.1 StDbXmlWriter::StDbXmlWriter () [inline]

Definition at line 62 of file StDbXmlWriter.h.

```
62 : os(0) {};
```

4.43.1.2 StDbXmlWriter::StDbXmlWriter (ostream & *ofs*) [inline]

Definition at line 63 of file StDbXmlWriter.h.

```
63 { os=&ofs;};
```

4.43.1.3 virtual StDbXmlWriter::~StDbXmlWriter () [inline, virtual]

Definition at line 64 of file StDbXmlWriter.h.

```
64 {};
```

4.43.2 Member Function Documentation

4.43.2.1 void StDbXmlWriter::endAccessor () [virtual]

Implements tableAcceptor.

Definition at line 55 of file StDbXmlWriter.cc.

```
55                              {
56   *os << "</StDbAccessor>" << endl;
57 }
```

4.43.2.2 void StDbXmlWriter::ioTable (StDbTable * *table*) [virtual]

Implements tableAcceptor.

Definition at line 95 of file StDbXmlWriter.cc.

```
95                                      {

97     char * name = table->getName();
98     streamTableName(name); delete [] name;
99     streamAccessor();
100     table->StreamAccessor((typeAcceptor*)this, false);
101     endAccessor();
102
```

```
    int k;
    int nrows;
    int* elements = table->getElementID(nrows);

    if(!elements){
      elements = new int[nrows];
      for(k=0;k<nrows;k++)elements[k]=k;
    }

    table->setRowNumber(); // set to 0

    for(k=0;k<nrows;k++){
      streamRow(k);
      table->dbStreamer((typeAcceptor*)this, false);
      streamEndRow();
    }
    streamEndTableName();
}
```

4.43.2.3 void StDbXmlWriter::pass (char * *name*, double *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 319 of file StDbXmlWriter.cc.

```
319                                                    {
320
321   *os << "<dbDoubleArray> " << name << " <length> ";
322   *os << len << " </length>" << endl;
323   *os << "<value>" << endl;
324
325   int icount = 0;
326   for(int j=0; j<len-1;j++){
327     *os << i[j] << ", ";
328     icount++;
329     if(icount==8){
330       *os << endl;
331       icount = 0;
332     }
333   }
334
335   *os << i[len-1] << endl;
336   *os << "</value> </dbDoubleArray> " << endl;
337 }
```

4.43.2.4 void StDbXmlWriter::pass (char * *name*, float *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 289 of file StDbXmlWriter.cc.

```
289                                                    {
290
291   *os << "<dbFloatArray> " << name << " <length> ";
292   *os << len << " </length>" << endl;
293   *os << "<value>" << endl;
294
295   int icount = 0;
296   for(int j=0; j<len-1;j++){
297     *os << i[j] << ", ";
298     icount++;
```

```
    if(icount==8){
      *os << endl;
      icount = 0;
    }
  }

  *os << i[len-1] << endl;
  *os << "</value> </dbFloatArray> " << endl;
}
```

4.43.2.5 void StDbXmlWriter::pass (char * *name*, long long *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 404 of file StDbXmlWriter.cc.

```
                                                        {

  *os << "<dbLongLongArray> " << name << " <length> ";
  *os << len << " </length>" << endl;
  *os << "<value>" << endl;

  int icount = 0;
  for(int j=0; j<len-1;j++){
    *os << i[j] << ", ";
    icount++;
    if(icount==8){
      *os << endl;
      icount = 0;
    }
  }

  *os << i[len-1] << endl;
  *os << "</value> </dbLongLongArray> " << endl;
}
```

4.43.2.6 void StDbXmlWriter::pass (char * *name*, unsigned long *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 382 of file StDbXmlWriter.cc.

```
                                                          {

  *os << "<dbULongArray> " << name << " <length> ";
  *os << len << " </length>" << endl;
  *os << "<value>" << endl;

  int icount = 0;
  for(int j=0; j<len-1;j++){
    *os << i[j] << ", ";
    icount++;
    if(icount==8){
      *os << endl;
      icount = 0;
    }
  }

  *os << i[len-1] << endl;
  *os << "</value> </dbULongArray> " << endl;
}
```

4.43.2.7 void StDbXmlWriter::pass (char * *name*, unsigned int *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 259 of file StDbXmlWriter.cc.

```
259                                                                 {
260
261   *os << "<dbUIntArray> " << name << " <length> ";
262   *os << len << " </length>" << endl;
263   *os << "<value>" << endl;
264
265   int icount = 0;
266   for(int j=0; j<len-1;j++){
267     *os << i[j] << ", ";
268     icount++;
269     if(icount==8){
270       *os << endl;
271       icount = 0;
272     }
273   }
274
275   *os << i[len-1] << endl;
276   *os << "</value> </dbUIntArray> " << endl;
277 }
```

4.43.2.8 void StDbXmlWriter::pass (char * *name*, unsigned short *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 199 of file StDbXmlWriter.cc.

```
199                                                                 {
200
201   *os << "<dbUShortArray> " << name << " <length> ";
202   *os << len << " </length>" << endl;
203   *os << "<value>" << endl;
204
205   int icount = 0;
206   for(int j=0; j<len-1;j++){
207     *os << i[j] << ", ";
208     icount++;
209     if(icount==8){
210       *os << endl;
211       icount = 0;
212     }
213   }
214
215   *os << i[len-1] << endl;
216   *os << "</value> </dbUShortArray> " << endl;
217 }
```

4.43.2.9 void StDbXmlWriter::pass (char * *name*, long *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 348 of file StDbXmlWriter.cc.

```
348                                                 {
349
```

```
  *os << "<dbLongArray> " << name << " <length> ";
  *os << len << " </length>" << endl;
  *os << "<value>" << endl;

  int icount = 0;
  for(int j=0; j<len-1;j++){
    *os << i[j] << ", ";
    icount++;
    if(icount==8){
      *os << endl;
      icount = 0;
    }
  }

  *os << i[len-1] << endl;
  *os << "</value> </dbLongArray> " << endl;
}
```

4.43.2.10 void StDbXmlWriter::pass (char * *name*, int *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 229 of file StDbXmlWriter.cc.

```
229                                                    {
230
231   *os << "<dbIntArray> " << name << " <length> ";
232   *os << len << " </length>" << endl;
233   *os << "<value>" << endl;
234
235   int icount = 0;
236   for(int j=0; j<len-1;j++){
237     *os << i[j] << ", ";
238     icount++;
239     if(icount==8){
240       *os << endl;
241       icount = 0;
242     }
243   }
244
245   *os << i[len-1] << endl;
246   *os << "</value> </dbIntArray> " << endl;
247 }
```

4.43.2.11 void StDbXmlWriter::pass (char * *name*, short *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 169 of file StDbXmlWriter.cc.

```
169                                                      {
170
171   *os << "<dbShortArray> " << name << " <length> ";
172   *os << len << " </length>" << endl;
173   *os << "<value>" << endl;
174
175   int icount = 0;
176   for(int j=0; j<len-1;j++){
177     *os << i[j] << ", ";
178     icount++;
179     if(icount==8){
```

```
      *os << endl;
      icount = 0;
    }
  }

  *os << i[len-1] << endl;
  *os << "</value> </dbShortArray> " << endl;
}
```

4.43.2.12 void StDbXmlWriter::pass (char * *name*, unsigned char *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 139 of file StDbXmlWriter.cc.

```
139                                                                 {
140   *os << "<dbUCharArray> " << name << " <length> ";
141   *os << len << " </length>" << endl;
142   *os << "<value>" << endl;
143
144   int icount = 0;
145   for(int j=0; j<len-1;j++){
146     *os << (int)i[j] << ", ";
147     icount++;
148     if(icount==8){
149       *os << endl;
150       icount = 0;
151     }
152   }
153
154   *os << (int)i[len-1] << endl;
155   *os << "</value> </dbUCharArray> " << endl;
156 }
```

4.43.2.13 void StDbXmlWriter::pass (char * *name*, unsigned char & *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 132 of file StDbXmlWriter.cc.

```
132                                                                 {
133 *os << "<dbUChar> "<<name<<" <value> "<< (int)i <<" </value> </dbUChar>" << endl;
134 }
```

4.43.2.14 void StDbXmlWriter::pass (char * *name*, char *& *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 125 of file StDbXmlWriter.cc.

```
125                                                         {
126 *os << "<dbString> "<<name<<" <value> "<< i <<" </value> </dbString>" << endl;
127 }
```

4.43.2.15 void StDbXmlWriter::pass (char * *name*, double & *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 312 of file StDbXmlWriter.cc.

```
312                                                    {
313 *os << "<dbDouble> "<<name<<" <value> "<< i <<" </value> </dbDouble>" << endl;
314 }
```

4.43.2.16 void StDbXmlWriter::pass (char * *name*, float & *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 282 of file StDbXmlWriter.cc.

```
282                                                   {
283 *os << "<dbFloat> "<<name<<" <value> "<< i <<" </value> </dbFloat>" << endl;
284 }
```

4.43.2.17 void StDbXmlWriter::pass (char * *name*, long long & *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 375 of file StDbXmlWriter.cc.

```
375                                                        {
376 *os << "<dbLongLong> "<<name<<" <value> "<< i <<" </value> </dbLongLong>" << endl;
377 }
```

4.43.2.18 void StDbXmlWriter::pass (char * *name*, unsigned long & *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 371 of file StDbXmlWriter.cc.

```
371                                                          {
372 *os << "<dbULong> "<<name<<" <value> "<< i <<" </value> </dbULong>" << endl;
373 }
```

4.43.2.19 void StDbXmlWriter::pass (char * *name*, unsigned int & *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 252 of file StDbXmlWriter.cc.

```
252                                                         {
253 *os << "<dbUInt> "<<name<<" <value> "<< i <<" </value> </dbUInt>" << endl;
254 }
```

4.43.2.20 void StDbXmlWriter::pass (char * *name*, unsigned short & *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 192 of file StDbXmlWriter.cc.

```
192                                                                 {
193 *os << "<dbUShort> "<<name<<" <value> "<< i <<" </value> </dbUShort>" << endl;
194 }
```

4.43.2.21 void StDbXmlWriter::pass (char * *name*, long & *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 341 of file StDbXmlWriter.cc.

```
341                                                     {
342 *os << "<dbLong> "<<name<<" <value> "<< i <<" </value> </dbLong>" << endl;
343 }
```

4.43.2.22 void StDbXmlWriter::pass (char * *name*, int & *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 222 of file StDbXmlWriter.cc.

```
222                                                     {
223 *os << "<dbInt> "<<name<<" <value> "<< i <<" </value> </dbInt>" << endl;
224 }
```

4.43.2.23 void StDbXmlWriter::pass (char * *name*, short & *i*, int & *len*) [virtual]

Implements typeAcceptor.

Definition at line 162 of file StDbXmlWriter.cc.

```
162                                                     {
163 *os << "<dbShort> "<<name<<" <value> "<< i <<" </value> </dbShort>" << endl;
164 }
```

4.43.2.24 void StDbXmlWriter::streamAccessor () [virtual]

Implements tableAcceptor.

Definition at line 48 of file StDbXmlWriter.cc.

```
48                                {
49   *os << "<StDbAccessor>" << endl;
50 }
```

4.43.2.25 void StDbXmlWriter::streamEndRow () [virtual]

Implements tableAcceptor.

Definition at line 76 of file StDbXmlWriter.cc.

```
76                              {
77   *os << "</TabRow>" << endl;
78 }
```

4.43.2.26 void StDbXmlWriter::streamEndTableName () [virtual]

Implements tableAcceptor.

Definition at line 88 of file StDbXmlWriter.cc.

```
88                                  {
89   *os << "</StDbTable>"<< endl;
90 }
```

4.43.2.27 void StDbXmlWriter::streamHeader (const char * *name*) [virtual]

Implements tableAcceptor.

Definition at line 41 of file StDbXmlWriter.cc.

```
41                                          {
42   *os << "<StDbName>"<<name<< endl;
43 }
```

4.43.2.28 void StDbXmlWriter::streamRow (int *row*) [virtual]

Implements tableAcceptor.

Definition at line 69 of file StDbXmlWriter.cc.

```
69                                  {
70   *os << "<TabRow>" << row << endl;
71 }
```

4.43.2.29 void StDbXmlWriter::streamTableName (const char * *name*) [virtual]

Implements tableAcceptor.

Definition at line 82 of file StDbXmlWriter.cc.

```
82                                              {
83   *os << "<StDbTable>"<<name << endl;
84 }
```

4.43.2.30 void StDbXmlWriter::streamTail () [virtual]

Implements tableAcceptor.

Definition at line 62 of file StDbXmlWriter.cc.

```
62                          {
63   *os << "</StDbName>" << endl;
64 }
```

4.43.3 Member Data Documentation

4.43.3.1 ostream* StDbXmlWriter::os [protected]

Definition at line 58 of file StDbXmlWriter.h.

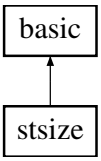
The documentation for this class was generated from the following files:

- StDbXmlWriter.h
- StDbXmlWriter.cc

4.44 stsize Class Reference

`#include <dbStruct.hh>`

Inheritance diagram for stsize::



Public Member Functions

- stsize ()

Public Attributes

- int isize

4.44.1 Constructor & Destructor Documentation

4.44.1.1 stsize::stsize () [inline]

Definition at line 59 of file dbStruct.hh.

```
59            : isize(0) { setStartKey("<length>");
60    setEndKey("</length>") ;};
```

4.44.2 Member Data Documentation

4.44.2.1 int stsize::isize

Definition at line 58 of file dbStruct.hh.

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

- dbStruct.hh

4.45 StString Class Reference

`#include <StString.h>`

Public Member Functions

- StString ()
- StString & operator<< (const char *txt)
- StString & operator<< (int I)
- StString & operator<< (unsigned int I)
- StString & operator<< (short I)
- StString & operator<< (unsigned short I)
- StString & operator<< (long I)
- StString & operator<< (unsigned long I)
- StString & operator<< (long long I)
- StString & operator<< (unsigned long long I)
- StString & operator<< (float F)
- StString & operator<< (double D)
- void precision (int prec)
- const std::string & str ()

Private Attributes

- int fPrec

4.45.1 Constructor & Destructor Documentation

4.45.1.1 StString::StString () [inline]

Definition at line 7 of file StString.h.

```
7 {fPrec=0;}
```

4.45.2 Member Function Documentation

4.45.2.1 StString & StString::operator<< (double *D*)

Definition at line 50 of file StString.cc.

```
51 {
52   char cbuf[200];
53   if (!fPrec) {
54      sprintf(cbuf,"%g",I);
55   } else {
56      sprintf(cbuf,"%.*g",fPrec,I);
57   }
58   fPrec = 0;
59   (*this) += cbuf;
60   return *this;
61 }
```

4.45.2.2 StString & StString::operator<< (float *F*)

Definition at line 64 of file StString.cc.

```
65 { double d = I; *this << d; return *this;}
```

4.45.2.3 StString & StString::operator<< (unsigned long long *I*)

Definition at line 28 of file StString.cc.

```
29 {
30   char cbuf[100];
31   if (!fPrec) {
32      sprintf(cbuf,"%lu",(unsigned long)I);
33   } else {
34      sprintf(cbuf,"%.*lu",fPrec,(unsigned long)I);
35   }
36   fPrec = 0;
37   (*this) += cbuf;
38   return (*this);
39 }
```

4.45.2.4 StString & StString::operator<< (long long *I*)

Definition at line 6 of file StString.cc.

```
7 {
8   char cbuf[200];
9   if (!fPrec) {
10      sprintf(cbuf,"%ld",(long)I);
11   } else {
12      sprintf(cbuf,"%.*ld",fPrec,(long)I);
13   }
14   fPrec = 0;
15   (*this) += cbuf;
16   return *this;
17 }
```

4.45.2.5 StString & StString::operator<< (unsigned long *I*)

Definition at line 41 of file StString.cc.

```
42 { unsigned long long l = I; *this << l;return *this;}
```

4.45.2.6 StString & StString::operator<< (long *I*)

Definition at line 19 of file StString.cc.

```
20 { long long l = I; *this << l;return *this;}
```

4.45.2.7 StString & StString::operator<< (unsigned short *I*)

Definition at line 47 of file StString.cc.

```
48 { unsigned long long l = I; *this << l;return *this;}
```

4.45.2.8 StString & StString::operator<< (short *I*)

Definition at line 25 of file StString.cc.

```
26 { long long l = I; *this << l;return *this;}
```

4.45.2.9 StString & StString::operator<< (unsigned int *I*)

Definition at line 44 of file StString.cc.

```
45 { unsigned long long l = I; *this << l;return *this;}
```

4.45.2.10 StString & StString::operator<< (int *I*)

Definition at line 22 of file StString.cc.

```
23 { long long l = I; *this << l;return *this;}
```

4.45.2.11 StString& StString::operator<< (const char * *txt*) [inline]

Definition at line 9 of file StString.h.

```
9 { (*this)+=txt; return *this;}
```

4.45.2.12 void StString::precision (int *prec*) [inline]

Definition at line 20 of file StString.h.

```
20 {fPrec = prec;}
```

4.45.2.13 const std::string& StString::str () [inline]

Definition at line 22 of file StString.h.

```
22 {return *((const std::string*)this);}
```

4.45.3 Member Data Documentation

4.45.3.1 int StString::fPrec [private]

Definition at line 25 of file StString.h.

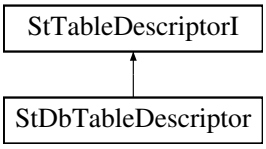
The documentation for this class was generated from the following files:

- StString.h
- StString.cc

4.46 StTableDescriptorI Class Reference

`#include <StTableDescriptorI.h>`

Inheritance diagram for StTableDescriptorI::



Public Member Functions

- virtual ~StTableDescriptorI ()
- virtual StTableDescriptorI * getCpy ()=0
- virtual unsigned int getNumElements () const =0
- virtual unsigned int getTotalSizeInBytes () const =0
- virtual char * getElementName (int elementNum) const =0
- virtual unsigned int getElementOffset (int elementNum) const =0
- virtual unsigned int getElementSize (int elementNum) const =0
- virtual int getTrowSize ()=0
- virtual StTypeE getElementType (int elementNum) const =0
- virtual unsigned int * getElementDimensions (int elementNum) const =0
- virtual unsigned int getElementLength (int elementNum) const =0
- virtual unsigned int getElementNumDimensions (int elementNum) const =0
- virtual unsigned int getElementIndexLength (int elementNum, int dimensionNum) const =0

4.46.1 Constructor & Destructor Documentation

4.46.1.1 virtual StTableDescriptorI::~StTableDescriptorI () `[inline, virtual]`

Definition at line 70 of file StTableDescriptorI.h.

```
70 {};
```

4.46.2 Member Function Documentation

4.46.2.1 virtual StTableDescriptorI* StTableDescriptorI::getCpy () `[pure virtual]`

Implemented in StDbTableDescriptor.

4.46.2.2 virtual unsigned int* StTableDescriptorI::getElementDimensions (int *elementNum*) const `[pure virtual]`

Implemented in StDbTableDescriptor.

4.46.2.3 virtual unsigned int StTableDescriptorI::getElementIndexLength (int *elementNum*, int *dimensionNum*) const [pure virtual]

Implemented in StDbTableDescriptor.

4.46.2.4 virtual unsigned int StTableDescriptorI::getElementLength (int *elementNum*) const [pure virtual]

Implemented in StDbTableDescriptor.

4.46.2.5 virtual char* StTableDescriptorI::getElementName (int *elementNum*) const [pure virtual]

Implemented in StDbTableDescriptor.

4.46.2.6 virtual unsigned int StTableDescriptorI::getElementNumDimensions (int *elementNum*) const [pure virtual]

Implemented in StDbTableDescriptor.

4.46.2.7 virtual unsigned int StTableDescriptorI::getElementOffset (int *elementNum*) const [pure virtual]

Implemented in StDbTableDescriptor.

4.46.2.8 virtual unsigned int StTableDescriptorI::getElementSize (int *elementNum*) const [pure virtual]

Implemented in StDbTableDescriptor.

4.46.2.9 virtual StTypeE StTableDescriptorI::getElementType (int *elementNum*) const [pure virtual]

Implemented in StDbTableDescriptor.

4.46.2.10 virtual unsigned int StTableDescriptorI::getNumElements () const [pure virtual]

Implemented in StDbTableDescriptor.

4.46.2.11 virtual unsigned int StTableDescriptorI::getTotalSizeInBytes () const [pure virtual]

Implemented in StDbTableDescriptor.

4.46.2.12 virtual int StTableDescriptorI::getTrowSize () [pure virtual]

Implemented in StDbTableDescriptor.

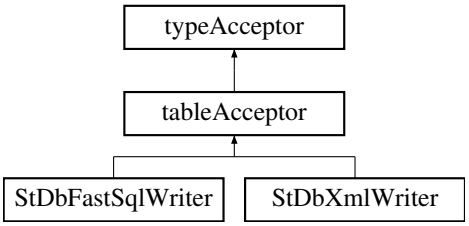
The documentation for this class was generated from the following file:

- StTableDescriptorI.h

4.47 tableAcceptor Class Reference

`#include <tableAcceptor.hh>`

Inheritance diagram for tableAcceptor::



Public Member Functions

- virtual ~tableAcceptor ()
- virtual void streamHeader (const char *name)=0
- virtual void streamTableName (const char *name)=0
- virtual void streamEndTableName ()=0
- virtual void streamAccessor ()=0
- virtual void endAccessor ()=0
- virtual void streamRow (int row)=0
- virtual void streamEndRow ()=0
- virtual void streamTail ()=0
- virtual void ioTable (StDbTable *table)=0

4.47.1 Constructor & Destructor Documentation

4.47.1.1 virtual tableAcceptor::~tableAcceptor () `[inline, virtual]`

Definition at line 46 of file tableAcceptor.hh.

```
46 {};
```

4.47.2 Member Function Documentation

4.47.2.1 virtual void tableAcceptor::endAccessor () `[pure virtual]`

Implemented in StDbFastSqlWriter, and StDbXmlWriter.

4.47.2.2 virtual void tableAcceptor::ioTable (StDbTable * *table*) `[pure virtual]`

Implemented in StDbFastSqlWriter, and StDbXmlWriter.

4.47.2.3 virtual void tableAcceptor::streamAccessor () `[pure virtual]`

Implemented in StDbFastSqlWriter, and StDbXmlWriter.

4.47.2.4 virtual void tableAcceptor::streamEndRow () [pure virtual]

Implemented in StDbFastSqlWriter, and StDbXmlWriter.

4.47.2.5 virtual void tableAcceptor::streamEndTableName () [pure virtual]

Implemented in StDbFastSqlWriter, and StDbXmlWriter.

4.47.2.6 virtual void tableAcceptor::streamHeader (const char * *name*) [pure virtual]

Implemented in StDbFastSqlWriter, and StDbXmlWriter.

4.47.2.7 virtual void tableAcceptor::streamRow (int *row*) [pure virtual]

Implemented in StDbFastSqlWriter, and StDbXmlWriter.

4.47.2.8 virtual void tableAcceptor::streamTableName (const char * *name*) [pure virtual]

Implemented in StDbFastSqlWriter, and StDbXmlWriter.

4.47.2.9 virtual void tableAcceptor::streamTail () [pure virtual]

Implemented in StDbFastSqlWriter, and StDbXmlWriter.

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

- tableAcceptor.hh

4.48 tableDescriptor Struct Reference

```
#include <StDbTableDescriptor.h>
```

Public Attributes

- StTypeE type
- char name [60]
- unsigned int size
- unsigned int offset
- unsigned int dimensionlen [4]

4.48.1 Member Data Documentation

4.48.1.1 unsigned int tableDescriptor::dimensionlen[4]

Definition at line 72 of file StDbTableDescriptor.h.

4.48.1.2 char tableDescriptor::name[60]

Definition at line 69 of file StDbTableDescriptor.h.

4.48.1.3 unsigned int tableDescriptor::offset

Definition at line 71 of file StDbTableDescriptor.h.

4.48.1.4 unsigned int tableDescriptor::size

Definition at line 70 of file StDbTableDescriptor.h.

4.48.1.5 StTypeE tableDescriptor::type

Definition at line 68 of file StDbTableDescriptor.h.

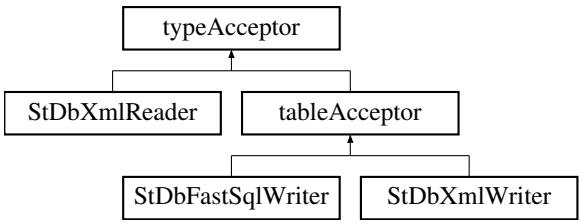
The documentation for this struct was generated from the following file:

- StDbTableDescriptor.h

4.49 typeAcceptor Class Reference

`#include <typeAcceptor.hh>`

Inheritance diagram for typeAcceptor::



Public Member Functions

- virtual ~typeAcceptor ()
- virtual void pass (char *name, short &i, int &len)=0
- virtual void pass (char *name, int &i, int &len)=0
- virtual void pass (char *name, long &i, int &len)=0
- virtual void pass (char *name, unsigned short &i, int &len)=0
- virtual void pass (char *name, unsigned int &i, int &len)=0
- virtual void pass (char *name, unsigned long &i, int &len)=0
- virtual void pass (char *name, long long &i, int &len)=0
- virtual void pass (char *name, float &i, int &len)=0
- virtual void pass (char *name, double &i, int &len)=0
- virtual void pass (char *name, char *&i, int &len)=0
- virtual void pass (char *name, unsigned char &i, int &len)=0
- virtual void pass (char *name, unsigned char *&i, int &len)=0
- virtual void pass (char *name, short *&i, int &len)=0
- virtual void pass (char *name, int *&i, int &len)=0
- virtual void pass (char *name, long *&i, int &len)=0
- virtual void pass (char *name, unsigned short *&i, int &len)=0
- virtual void pass (char *name, unsigned int *&i, int &len)=0
- virtual void pass (char *name, unsigned long *&i, int &len)=0
- virtual void pass (char *name, long long *&i, int &len)=0
- virtual void pass (char *name, float *&i, int &len)=0
- virtual void pass (char *name, double *&i, int &len)=0

4.49.1 Constructor & Destructor Documentation

4.49.1.1 virtual typeAcceptor::~typeAcceptor () `[inline, virtual]`

Definition at line 44 of file typeAcceptor.hh.

```
44 {};
```

4.49.2 Member Function Documentation

4.49.2.1 virtual void typeAcceptor::pass (char * *name*, double *& *i*, int & *len*) [pure virtual]

Implemented in StDbFastSqlWriter, StDbXmlReader, and StDbXmlWriter.

4.49.2.2 virtual void typeAcceptor::pass (char * *name*, float *& *i*, int & *len*) [pure virtual]

Implemented in StDbFastSqlWriter, StDbXmlReader, and StDbXmlWriter.

4.49.2.3 virtual void typeAcceptor::pass (char * *name*, long long *& *i*, int & *len*) [pure virtual]

Implemented in StDbFastSqlWriter, StDbXmlReader, and StDbXmlWriter.

4.49.2.4 virtual void typeAcceptor::pass (char * *name*, unsigned long *& *i*, int & *len*) [pure virtual]

Implemented in StDbFastSqlWriter, StDbXmlReader, and StDbXmlWriter.

4.49.2.5 virtual void typeAcceptor::pass (char * *name*, unsigned int *& *i*, int & *len*) [pure virtual]

Implemented in StDbFastSqlWriter, StDbXmlReader, and StDbXmlWriter.

4.49.2.6 virtual void typeAcceptor::pass (char * *name*, unsigned short *& *i*, int & *len*) [pure virtual]

Implemented in StDbFastSqlWriter, StDbXmlReader, and StDbXmlWriter.

4.49.2.7 virtual void typeAcceptor::pass (char * *name*, long *& *i*, int & *len*) [pure virtual]

Implemented in StDbFastSqlWriter, StDbXmlReader, and StDbXmlWriter.

4.49.2.8 virtual void typeAcceptor::pass (char * *name*, int *& *i*, int & *len*) [pure virtual]

Implemented in StDbFastSqlWriter, StDbXmlReader, and StDbXmlWriter.

4.49.2.9 virtual void typeAcceptor::pass (char * *name*, short *& *i*, int & *len*) [pure virtual]

Implemented in StDbFastSqlWriter, StDbXmlReader, and StDbXmlWriter.

4.49.2.10 virtual void typeAcceptor::pass (char * *name*, unsigned char *& *i*, int & *len*) [pure virtual]

Implemented in StDbFastSqlWriter, StDbXmlReader, and StDbXmlWriter.

4.49.2.11 virtual void typeAcceptor::pass (char * *name*, unsigned char & *i*, int & *len*) [pure virtual]

Implemented in StDbFastSqlWriter, StDbXmlReader, and StDbXmlWriter.

4.49.2.12 virtual void typeAcceptor::pass (char * *name*, char *& *i*, int & *len*) [pure virtual]

Implemented in StDbFastSqlWriter, StDbXmlReader, and StDbXmlWriter.

4.49.2.13 virtual void typeAcceptor::pass (char * *name*, double & *i*, int & *len*) [pure virtual]

Implemented in StDbFastSqlWriter, StDbXmlReader, and StDbXmlWriter.

4.49.2.14 virtual void typeAcceptor::pass (char * *name*, float & *i*, int & *len*) [pure virtual]

Implemented in StDbFastSqlWriter, StDbXmlReader, and StDbXmlWriter.

4.49.2.15 virtual void typeAcceptor::pass (char * *name*, long long & *i*, int & *len*) [pure virtual]

Implemented in StDbFastSqlWriter, StDbXmlReader, and StDbXmlWriter.

4.49.2.16 virtual void typeAcceptor::pass (char * *name*, unsigned long & *i*, int & *len*) [pure virtual]

Implemented in StDbFastSqlWriter, StDbXmlReader, and StDbXmlWriter.

4.49.2.17 virtual void typeAcceptor::pass (char * *name*, unsigned int & *i*, int & *len*) [pure virtual]

Implemented in StDbFastSqlWriter, StDbXmlReader, and StDbXmlWriter.

4.49.2.18 virtual void typeAcceptor::pass (char * *name*, unsigned short & *i*, int & *len*) [pure virtual]

Implemented in StDbFastSqlWriter, StDbXmlReader, and StDbXmlWriter.

4.49.2.19 virtual void typeAcceptor::pass (char * *name*, long & *i*, int & *len*) [pure virtual]

Implemented in StDbFastSqlWriter, StDbXmlReader, and StDbXmlWriter.

4.49.2.20 virtual void typeAcceptor::pass (char * *name*, int & *i*, int & *len*) [pure virtual]

Implemented in StDbFastSqlWriter, StDbXmlReader, and StDbXmlWriter.

4.49.2.21 virtual void typeAcceptor::pass (char * *name*, short & *i*, int & *len*) [pure virtual]

Implemented in StDbFastSqlWriter, StDbXmlReader, and StDbXmlWriter.

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

- typeAcceptor.hh

Chapter 5

Doxygen File Documentation

5.1 dbCollection.h File Reference

Classes

- struct dbCollection

5.2 dbStruct.hh File Reference

```
#include <vector>
```

Classes

- class basic
- class stsize
- class datav
- class elem
- class accessor
- class dbRow
- class dbTable

Typedefs

- typedef vector< elem * > elemVec
- typedef vector< dbRow * > rowVec

5.2.1 Typedef Documentation

5.2.1.1 typedef vector<elem*> elemVec

Definition at line 95 of file dbStruct.hh.

5.2.1.2 typedef vector<dbRow*> rowVec

Definition at line 128 of file dbStruct.hh.

5.3 MysqlDb.cc File Reference

```
#include "MysqlDb.h"
#include "StDbManager.hh"
#include "stdb_streams.h"
#include "StDbDefaults.hh"
#include "StDbManagerImpl.hh"
```

Defines

- #define CR_MIN_ERROR 2000 /∗ For easier client code ∗/
- #define CR_MAX_ERROR 2999
- #define CR_UNKNOWN_ERROR 2000
- #define CR_SOCKET_CREATE_ERROR 2001
- #define CR_CONNECTION_ERROR 2002
- #define CR_CONN_HOST_ERROR 2003
- #define CR_IPSOCK_ERROR 2004
- #define CR_UNKNOWN_HOST 2005
- #define CR_SERVER_GONE_ERROR 2006
- #define CR_VERSION_ERROR 2007
- #define CR_OUT_OF_MEMORY 2008
- #define CR_WRONG_HOST_INFO 2009
- #define CR_LOCALHOST_CONNECTION 2010
- #define CR_TCP_CONNECTION 2011
- #define CR_SERVER_HANDSHAKE_ERR 2012
- #define CR_SERVER_LOST 2013
- #define CR_COMMANDS_OUT_OF_SYNC 2014
- #define CR_NAMEDPIPE_CONNECTION 2015
- #define CR_NAMEDPIPEWAIT_ERROR 2016
- #define CR_NAMEDPIPEOPEN_ERROR 2017
- #define CR_NAMEDPIPESETSTATE_ERROR 2018
- #define __CLASS__ "MysqlDb"
- #define __METHOD__ "reConnect()"
- #define __METHOD__ "Connect(host,user,pw,database,port)"
- #define __METHOD__ "ExecQuery()"

Functions

- const MysqlBin ∗ Binary (const unsigned long int aLen, const float ∗aBin)

Variables

- const char ∗ binaryMessage = {"Cannot Print Query with Binary data"}
- MYSQL ∗ conn

5.3.1 Define Documentation

5.3.1.1 #define __CLASS__ "MysqlDb"

Definition at line 203 of file MysqlDb.cc.

5.3.1.2 #define __METHOD__ "ExecQuery()"

5.3.1.3 #define __METHOD__ "Connect(host,user,pw,database,port)"

5.3.1.4 #define __METHOD__ "reConnect()"

5.3.1.5 #define CR_COMMANDS_OUT_OF_SYNC 2014

Definition at line 197 of file MysqlDb.cc.

5.3.1.6 #define CR_CONN_HOST_ERROR 2003

Definition at line 186 of file MysqlDb.cc.

5.3.1.7 #define CR_CONNECTION_ERROR 2002

Definition at line 185 of file MysqlDb.cc.

5.3.1.8 #define CR_IPSOCK_ERROR 2004

Definition at line 187 of file MysqlDb.cc.

5.3.1.9 #define CR_LOCALHOST_CONNECTION 2010

Definition at line 193 of file MysqlDb.cc.

5.3.1.10 #define CR_MAX_ERROR 2999

Definition at line 182 of file MysqlDb.cc.

5.3.1.11 #define CR_MIN_ERROR 2000 /* For easier client code */

Definition at line 181 of file MysqlDb.cc.

5.3.1.12 #define CR_NAMEDPIPE_CONNECTION 2015

Definition at line 198 of file MysqlDb.cc.

5.3.1.13 #define CR_NAMEDPIPEOPEN_ERROR 2017

Definition at line 200 of file MysqlDb.cc.

5.3.1.14 #define CR_NAMEDPIPESETSTATE_ERROR 2018

Definition at line 201 of file MysqlDb.cc.

5.3.1.15 #define CR_NAMEDPIPEWAIT_ERROR 2016

Definition at line 199 of file MysqlDb.cc.

5.3.1.16 #define CR_OUT_OF_MEMORY 2008

Definition at line 191 of file MysqlDb.cc.

5.3.1.17 #define CR_SERVER_GONE_ERROR 2006

Definition at line 189 of file MysqlDb.cc.

5.3.1.18 #define CR_SERVER_HANDSHAKE_ERR 2012

Definition at line 195 of file MysqlDb.cc.

5.3.1.19 #define CR_SERVER_LOST 2013

Definition at line 196 of file MysqlDb.cc.

5.3.1.20 #define CR_SOCKET_CREATE_ERROR 2001

Definition at line 184 of file MysqlDb.cc.

5.3.1.21 #define CR_TCP_CONNECTION 2011

Definition at line 194 of file MysqlDb.cc.

5.3.1.22 #define CR_UNKNOWN_ERROR 2000

Definition at line 183 of file MysqlDb.cc.

5.3.1.23 #define CR_UNKNOWN_HOST 2005

Definition at line 188 of file MysqlDb.cc.

5.3.1.24 #define CR_VERSION_ERROR 2007

Definition at line 190 of file MysqlDb.cc.

5.3.1.25 #define CR_WRONG_HOST_INFO 2009

Definition at line 192 of file MysqlDb.cc.

5.3.2 Function Documentation

5.3.2.1 const MysqlBin* Binary (const unsigned long int *aLen*, const float * *aBin*)

Definition at line 902 of file MysqlDb.cc.

```
902                                                                    {
903   static MysqlBin   *tBin=0;
904   static char       *tString=0;
905   unsigned long int  tNewLen;
906
907   if (!tBin) tBin=new MysqlBin;
908   if (tString) delete [] tString;
909
910   tString = new char[2*aLen+1];
911   tNewLen = mysql_escape_string(tString,(char*) aBin,aLen);
912   tBin->Input(tNewLen,tString);
913   return tBin;
914 }
```

5.3.3 Variable Documentation

5.3.3.1 const char* binaryMessage = {"Cannot Print Query with Binary data"} [static]

Definition at line 206 of file MysqlDb.cc.

5.3.3.2 MYSQL* conn [static]

Definition at line 207 of file MysqlDb.cc.

5.4 MysqlDb.h File Reference

```
#include "mysql.h"
#include "stdb_streams.h"
#include "StDbBuffer.h"
#include "StDbLogger.hh"
#include "parseXmlString.hh"
#include "StDbManagerImpl.hh"
#include <vector.h>
#include <string.h>
```

Classes

- class MysqlResult
- class MysqlBin
- class MysqlDb

Defines

- #define endsql ";"

Functions

- const MysqlBin ∗ Binary (const unsigned long int aLen, const float ∗aBin)

5.4.1 Define Documentation

5.4.1.1 #define endsql ";"

Definition at line 146 of file MysqlDb.h.

5.4.2 Function Documentation

5.4.2.1 const MysqlBin∗ Binary (const unsigned long int *aLen*, const float ∗ *aBin*)

Definition at line 902 of file MysqlDb.cc.

```
902                                                                    {
903   static MysqlBin   *tBin=0;
904   static char       *tString=0;
905   unsigned long int  tNewLen;
906
907   if (!tBin) tBin=new MysqlBin;
908   if (tString) delete [] tString;
909
910   tString = new char[2*aLen+1];
911   tNewLen = mysql_escape_string(tString,(char*) aBin,aLen);
912   tBin->Input(tNewLen,tString);
```

```
  return tBin;
}
```

5.5 parseXmlString.cc File Reference

```
#include "parseXmlString.hh"
#include "stdb_streams.h"
#include <string.h>
```

5.6 parseXmlString.hh File Reference

Classes

- class parseXmlString

5.7 StDataBaseI.cc File Reference

```
#include "StDataBaseI.hh"
#include "StDbManager.hh"
#include <string.h>
```

5.8 StDataBaseI.hh File Reference

```
#include "StDbDefs.hh"
```

Classes

- class StDataBaseI

5.9 stdb_streams.h File Reference

```
#include "Stiostream.h"
#include "StString.h"
```

5.10 StDbBuffer.cc File Reference

```
#include "StDbBuffer.h"
#include <stdlib.h>
#include <string.h>
#include "stdb_streams.h"
```

Defines

- #define castcase(typelist, casttype, tpe) case typelist: {casttype *tVal=(casttype*)aVal;casttype tValSwap;MemSwapCpy((char*)&tValSwap,(char*)tVal,mycsize[typelist],mycswapl[typelist],Client);*s=(tpe)tValSwap;};break;
- #define genwritemem(tpe)
- #define castcasest(typelist, casttype) case typelist: {casttype tVal; MemSwapCpy((char*)&tVal,(char*)aVal,mycsize[typelist],mycswapl[typelist],Client);StString sStream;sStream.precision(10); sStream << tVal; string s2=sStream.str();char *tStr=new char[s2.length()+1];strcpy(tStr,s2.c_str());s[0]=tStr; };break
- #define Rscal(tpe)
- #define Wscal(tpe, tpelist, len)
- #define Rarray(tpe, tpelist)
- #define Warray(tpe, tpelist)

Functions

- genwritemem (char)
- genwritemem (unsigned char)
- genwritemem (short)
- genwritemem (int)
- genwritemem (long)
- genwritemem (float)
- genwritemem (double)
- Rscal (char &)
- Rscal (unsigned char &)
- Rscal (short &)
- Rscal (unsigned short &)
- Rscal (int &)
- Rscal (unsigned int &)
- Rscal (long &)
- Rscal (unsigned long &)
- Rscal (long long &)
- Rscal (float &)
- Rscal (double &)
- Rscal (char *&)
- Wscal (char, _char, 1)
- Wscal (unsigned char, _uchar, 1)
- Wscal (short, _short, 1)
- Wscal (unsigned short, _ushort, 1)
- Wscal (int, _int, 1)

- Wscal (unsigned int, _uint, 1)
- Wscal (long, _long, 1)
- Wscal (unsigned long, _ulong, 1)
- Wscal (float, _float, 1)
- Wscal (double, _double, 1)
- Rarray (unsigned char, _uchar)
- Rarray (short, _short)
- Rarray (unsigned short, _ushort)
- Rarray (int, _int)
- Rarray (unsigned int, _uint)
- Rarray (long, _long)
- Rarray (unsigned long, _ulong)
- Rarray (float, _float)
- Rarray (double, _double)
- Rarray (char *, _string)
- Warray (char, _char)
- Warray (unsigned char, _uchar)
- Warray (short, _short)
- Warray (unsigned short, _ushort)
- Warray (int, _int)
- Warray (unsigned int, _uint)
- Warray (long, _long)
- Warray (unsigned long, _ulong)
- Warray (float, _float)
- Warray (double, _double)
- Warray (char *, _string)

5.10.1 Define Documentation

5.10.1.1 #define castcase(typelist, casttype, tpe) case typelist: {casttype *tVal=(casttype*)aVal;casttype tValSwap;MemSwapCpy((char*)&tValSwap,(char*)tVal,mycsize[typelist],mycswapl[typelist],Client);*s=(tpe)tValSwap;};break;

Definition at line 308 of file StDbBuffer.cc.

5.10.1.2 #define castcasest(typelist, casttype) case typelist: {casttype tVal; MemSwapCpy((char*)&tVal,(char*)aVal,mycsize[typelist],mycswapl[typelist],Client);StString sStream;sStream.precision(10); sStream << tVal; string s2=sStream.str();char *tStr=new char[s2.length()+1];strcpy(tStr,s2.c_str());s[0]=tStr; };break

Definition at line 342 of file StDbBuffer.cc.

5.10.1.3 #define genwritemem(tpe)

Value:

```
bool StDbBuffer::WriteMem( tpe *s,void* aVal, myctype type)\
{bool tRetVal=true;\
 switch (type) {\
   castcase(_char,char,tpe);\
```

```
   castcase(_uchar,unsigned char,tpe);\
   castcase(_short,short,tpe);\
   castcase(_ushort,unsigned short,tpe);\
   castcase(_int,int,tpe);\
   castcase(_uint,unsigned int,tpe);\
   castcase(_long,long,tpe);\
   castcase(_ulong,unsigned long,tpe);\
   castcase(_longlong,long long,tpe);\
   castcase(_float,float,tpe);\
   castcase(_double,double,tpe);\
 case _string: {char** tVal=(char**)aVal;StrConv(*tVal,*s);};break;\
 default: cout <<"wrong type" << endl;tRetVal=false;\
  };\
  return tRetVal;\
}
```

Definition at line 310 of file StDbBuffer.cc.

5.10.1.4 #define Rarray(tpe, tpelist)

Value:

```
bool  StDbBuffer::ReadArray(tpe* &s, int &len,const char *aName)\
{ bool tRetVal=false; \
  bool newCheck=false;\
 if (Find_Col(aName)) \
  {int  i;\
  if (mCol[mCur].type==_char ) {\
    len=mCol[mCur].length/sizeof(tpe);\
    s=new tpe[len];\
    newCheck=true;\
    MemSwapCpy((char*)s,(char*)mCol[mCur].val,len*sizeof(tpe),mycswapl[tpelist],Auto);\
    tRetVal=true;\
  } else {\
    len=mCol[mCur].length;\
    s=new tpe[len];\
    newCheck=true;\
    for (i=0;i<len;i++)\
      { if (!(WriteMem(&s[i],(void*)(((char*)mCol[mCur].val)+i*mycsize[mCol[mCur].type]),mCol[mCur].type))
      if (i==(int)mCol[mCur].length) tRetVal=true;}}\
 return tRetVal;\
}
```

Definition at line 445 of file StDbBuffer.cc.

5.10.1.5 #define Rscal(tpe)

Value:

```
bool  StDbBuffer::ReadScalar(tpe s,const char *aName) \
{bool tRetVal=false; \
  if(Find_Col(aName) && WriteMem(&s,mCol[mCur].val,mCol[mCur].type)) \
    tRetVal=true;\
  return tRetVal;}
```

Definition at line 369 of file StDbBuffer.cc.

5.10.1.6 #define Warray(tpe, tpelist)

Value:

```
bool StDbBuffer::WriteArray(tpe* s,int len,const char *aName) \
{ if (Find_Col(aName))\
  {ChangeField(tpelist,(void*)s,len);}\
 else\
   {AddField(aName,tpelist,(void*)s,len);};\
return true;\
}
```

Definition at line 496 of file StDbBuffer.cc.

5.10.1.7 #define Wscal(tpe, tpelist, len)

Value:

```
bool StDbBuffer::WriteScalar(const tpe s,const char *aName) \
{ if (Find_Col(aName))\
  {ChangeField(tpelist,(void*)&s,len);}\
 else\
   {AddField(aName,tpelist,(void*)&s,len);};\
return true;\
}
```

Definition at line 396 of file StDbBuffer.cc.

5.10.2 Function Documentation

5.10.2.1 genwritemem (double)

5.10.2.2 genwritemem (float)

5.10.2.3 genwritemem (long)

5.10.2.4 genwritemem (int)

5.10.2.5 genwritemem (short)

5.10.2.6 genwritemem (unsigned *char*)

5.10.2.7 genwritemem (char)

5.10.2.8 Rarray (char *, _string)

5.10.2.9 Rarray (double, _double)

5.10.2.10 Rarray (float, _float)

5.10.2.11 Rarray (unsigned *long*, _ulong)

5.10.2.12 Rarray (long, _long)

5.10.2.13 Rarray (unsigned *int*, _uint)

5.10.2.14 Rarray (int, _int)

5.10.2.15 Rarray (unsigned *short*, _ushort)

5.10.2.16 Rarray (short, _short)

5.10.2.17 Rarray (unsigned *char*, _uchar)

5.10.2.18 Rscal (char *&)

5.10.2.19 Rscal (double &)

5.10.2.20 Rscal (float &)

5.10.2.21 Rscal (long long &)

5.10.2.22 Rscal (unsigned long &)

5.10.2.23 Rscal (long &)

5.10.2.24 Rscal (unsigned int &)

5.10.2.25 Rscal (int &)

5.10.2.26 Rscal (unsigned short &)

5.10.2.27 Rscal (short &)

5.10.2.28 Rscal (unsigned char &)

5.10.2.29 Rscal (char &)

5.10.2.30 Warray (char *, _string)

5.11 StDbBuffer.h File Reference

```
#include "mysql.h"
#include "mysql_com.h"
#include "StDbBufferI.h"
```

Classes

- struct column
- class StDbBuffer

Enumerations

- enum myctype {
  _char, _uchar, _short, _ushort,
  _int, _uint, _long, _ulong,
  _longlong, _float, _double, _ascii,
  _string }
- enum BuffMode { Auto, Client, Storage }

Variables

- const int mycsize [ ] = {sizeof(char),sizeof(unsigned char),sizeof(short),sizeof(unsigned short),sizeof(int),sizeof(unsigned int),sizeof(long),sizeof(unsigned long),sizeof(long long),sizeof(float),sizeof(double),sizeof(char∗),sizeof(char∗)}
- const int mycswapl [ ] = {1,1,1,1,1,1,1,1,1,1,1,1}

5.11.1 Enumeration Type Documentation

5.11.1.1 enum BuffMode

Enumeration values:

Auto

Client

Storage

Definition at line 70 of file StDbBuffer.h.

```
70 {Auto,Client,Storage};
```

5.11.1.2 enum myctype

Enumeration values:

_char

_uchar

_short

_ushort

_int

_uint

_long

_ulong

_longlong

_float

_double

_ascii

_string

Definition at line 54 of file StDbBuffer.h.

```
54 {_char,_uchar,_short,_ushort,_int,_uint,_long,_ulong,_longlong,_float,_double,_ascii,_string};
```

5.11.2 Variable Documentation

5.11.2.1 const int mycsize[ ] = {sizeof(char),sizeof(unsigned char),sizeof(short),sizeof(unsigned short),sizeof(int),sizeof(unsigned int),sizeof(long),sizeof(unsigned long),sizeof(long long),sizeof(float),sizeof(double),sizeof(char*),sizeof(char*)}

Definition at line 56 of file StDbBuffer.h.

5.11.2.2 const int mycswapl[ ] = {1,1,1,1,1,1,1,1,1,1,1,1}

Definition at line 61 of file StDbBuffer.h.

5.12 StDbBufferI.h File Reference

Classes

- class StDbBufferI

5.13 StDbConfigNode.cc File Reference

```
#include "stdb_streams.h"
#include <string.h>
#include "StDbConfigNode.hh"
#include "StDbManager.hh"
#include "StDbTable.h"
```

Defines

- #define __CLASS__ "StDbConfigNode"

5.13.1 Define Documentation

5.13.1.1 #define __CLASS__ "StDbConfigNode"

Definition at line 129 of file StDbConfigNode.cc.

5.14 StDbConfigNode.hh File Reference

```
#include "StDbNode.hh"
#include "StDbTableIter.hh"
```

Classes

- class StDbConfigNode

5.15 StDbConfigNodeImpl.cc File Reference

```
#include "stdb_streams.h"
#include <string.h>
#include "StDbConfigNodeImpl.hh"
#include "StDbManagerImpl.hh"
#include "StDbTableIterImpl.hh"
#include "StDbElementIndex.hh"
```

Defines

- #define __CLASS__ "StDbConfigNodeImpl"
- #define __METHOD__ "updateDbInfo()"
- #define __METHOD__ "buildTree(int opt)"
- #define __METHOD__ "addChildren(dbEnvList*)"
- #define __METHOD__ "printNumberStats()"

5.15.1 Define Documentation

5.15.1.1 #define __CLASS__ "StDbConfigNodeImpl"

Definition at line 61 of file StDbConfigNodeImpl.cc.

5.15.1.2 #define __METHOD__ "printNumberStats()"

5.15.1.3 #define __METHOD__ "addChildren(dbEnvList*)"

5.15.1.4 #define __METHOD__ "buildTree(int opt)"

5.15.1.5 #define __METHOD__ "updateDbInfo()"

5.16 StDbConfigNodeImpl.hh File Reference

```
#include "StDbConfigNode.hh"
#include "StDbElementIndex.hh"
#include <list>
```

Classes

- class StDbConfigNodeImpl

Typedefs

- typedef list< StDbTable * > TableList

5.16.1 Typedef Documentation

5.16.1.1 typedef list<StDbTable*> TableList

Definition at line 43 of file StDbConfigNodeImpl.hh.

5.17 StDbDefaults.cc File Reference

```
#include "StDbDefaults.hh"
#include "stdb_streams.h"
#include <dirent.h>
#include <stdlib.h>
```

5.18 StDbDefaults.hh File Reference

```
#include "StDbDefs.hh"
#include <string.h>
```

Classes

- class StDbDefaults

5.19 StDbDefs.hh File Reference

Enumerations

- enum StDbType {
  dbStDb = 0, dbServer, dbRunLog, dbConfigurations,
  dbConditions, dbCalibrations, dbGeometry, dbRunCatalog,
  dbRunParams, dbScalers, dbTestScheme, dbTUser1,
  dbTUser2, dbTUser3, dbTEnd }
- enum StDbDomain {
  dbDomainUnknown = 0, dbStar, dbTpc, dbEmc,
  dbFtpc, dbSvt, dbCtb, dbTrg,
  dbDaq, dbScaler, dbGlobal, dbL3,
  dbOnl, dbRich, dbMwc, dbRhic,
  dbSsd, dbRts, dbTof, dbFpd,
  dbEemc, dbPmd, dbBbc, dbTracker,
  dbZdc, dbDUser1, dbDUser2, dbDUser3,
  dbDEnd }
- enum StDbStoreType { dbV00 = 0 }
- enum dbFindServerMode { userHome = 0, serverEnvVar, starDefault }

5.19.1 Enumeration Type Documentation

5.19.1.1 enum dbFindServerMode

Enumeration values:

userHome

serverEnvVar

starDefault

Definition at line 122 of file StDbDefs.hh.

```
122 { userHome=0, serverEnvVar, starDefault };
```

5.19.1.2 enum StDbDomain

Enumeration values:

dbDomainUnknown

dbStar

dbTpc

dbEmc

dbFtpc

dbSvt

dbCtb

dbTrg

dbDaq

dbScaler

dbGlobal

dbL3

dbOnl

dbRich

dbMwc

dbRhic

dbSsd

dbRts

dbTof

dbFpd

dbEemc

dbPmd

dbBbc

dbTracker

dbZdc

dbDUser1

dbDUser2

dbDUser3

dbDEnd

Definition at line 117 of file StDbDefs.hh.

```
117 {dbDomainUnknown=0, dbStar, dbTpc, dbEmc, dbFtpc, dbSvt, dbCtb, dbTrg, dbDaq, dbScaler, dbGlobal, dbL3
```

5.19.1.3 enum StDbStoreType

Enumeration values:

dbV00

Definition at line 120 of file StDbDefs.hh.

```
120 { dbV00=0 };
```

5.19.1.4 enum StDbType

Enumeration values:

dbStDb

dbServer

dbRunLog

dbConfigurations

dbConditions

dbCalibrations

dbGeometry

dbRunCatalog

dbRunParams

dbScalers

dbTestScheme

dbTUser1

dbTUser2

dbTUser3

dbTEnd

Definition at line 114 of file StDbDefs.hh.

```
114 { dbStDb=0, dbServer, dbRunLog, dbConfigurations, dbConditions, dbCalibrations, dbGeometry, dbRunCatal
```

5.20 StDbElementIndex.cc File Reference

```
#include "StDbElementIndex.hh"
#include <string.h>
```

5.21 StDbElementIndex.hh File Reference

```
#include <string.h>
```

Classes

- struct indexNameVals
- class StDbElementIndex

Defines

- #define N_MAX_INDEXVALS 10

5.21.1 Define Documentation

5.21.1.1 #define N_MAX_INDEXVALS 10

Definition at line 35 of file StDbElementIndex.hh.

5.22 StDbFastSqlWriter.cc File Reference

```
#include "StDbFastSqlWriter.h"
#include "StDbTable.h"
```

5.23 StDbFastSqlWriter.h File Reference

```
#include <stdlib.h>
#include <string.h>
#include "tableAcceptor.hh"
#include "stdb_streams.h"
```

Classes

- class StDbFastSqlWriter

5.24 StDbLogger.hh File Reference

```
#include <sys/time.h>
```

Classes

- class StDbLogger

5.25 StDbManager.cc File Reference

```
#include "StDbManager.hh"
#include "StDbManagerImpl.hh"
#include "StDbMessenger.hh"
#include "stdb_streams.h"
```

5.26 StDbManager.hh File Reference

```
#include <string.h>
#include "StString.h"
#include "stdb_streams.h"
#include "StDbDefs.hh"
#include "StDbMessService.hh"
```

Classes

- class StDbManager

5.27 StDbManagerImpl.cc File Reference

```
#include "StDbManagerImpl.hh"
#include "StDbDefaults.hh"
#include "StDbTableFactory.hh"
#include "StDbConfigNodeImpl.hh"
#include "StDbServerImpl.hh"
#include "StDbTable.h"
#include "StDbTime.h"
#include "StDbTableIter.hh"
#include "dbCollection.h"
#include "StDbMessenger.hh"
#include "stdb_streams.h"
#include <string.h>
```

Defines

- #define __CLASS__ "StDbManagerImpl"
- #define __METHOD__ "lookUpServer()"
- #define __METHOD__
- #define __METHOD__ "getDbTypeName(StDbType)"
- #define __METHOD__ "getDbDomainName(StDbDomain)"
- #define __METHOD__ "getDbType(typeName)"
- #define __METHOD__ "getDbDomain(domainName)"
- #define __METHOD__ "initConfig(dbName,configName,opt)"
- #define __METHOD__ "initConfig(type,domain,time)"
- #define __METHOD__ "updateDateTime(StDbTime)"
- #define __METHOD__ "storeDbTable(StDbTable*)"
- #define __METHOD__ "storeAllTable(StDbConfigNode*)"
- #define __METHOD__ "storeConfig(node,ID,configID,commitFlag)"
- #define __METHOD__ "rollBack(node)"
- #define __METHOD__ "printTimeStats()"
- #define __METHOD__ "getDataBaseInfo(dbName,type,domain)"

Variables

- char ** environ

5.27.1 Define Documentation

5.27.1.1 #define __CLASS__ "StDbManagerImpl"

Definition at line 218 of file StDbManagerImpl.cc.

5.27.1.2 #define __METHOD__ "getDataBaseInfo(dbName,type,domain)"

5.27.1.3 #define __METHOD__ "printTimeStats()"

5.27.1.4 #define __METHOD__ "rollBack(node)"

5.27.1.5 #define __METHOD__ "storeConfig(node,ID,configID,commitFlag)"

5.27.1.6 #define __METHOD__ "storeAllTable(StDbConfigNode*)"

5.27.1.7 #define __METHOD__ "storeDbTable(StDbTable*)"

5.27.1.8 #define __METHOD__ "updateDateTime(StDbTime)"

5.27.1.9 #define __METHOD__ "initConfig(type,domain,time)"

5.27.1.10 #define __METHOD__ "initConfig(dbName,configName,opt)"

5.27.1.11 #define __METHOD__ "getDbDomain(domainName)"

5.27.1.12 #define __METHOD__ "getDbType(typeName)"

5.27.1.13 #define __METHOD__ "getDbDomainName(StDbDomain)"

5.27.1.14 #define __METHOD__ "getDbTypeName(StDbType)"

5.27.1.15 #define __METHOD__

5.27.1.16 #define __METHOD__ "lookUpServer()"

5.27.2 Variable Documentation

5.27.2.1 char** environ

Definition at line 216 of file StDbManagerImpl.cc.

5.28 StDbManagerImpl.hh File Reference

```
#include "StDbManager.hh"
#include "parseXmlString.hh"
#include "stdb_streams.h"
#include "StDbTime.h"
#include "StDataBaseI.hh"
#include "StDbTableFactory.hh"
#include "StDbMessService.hh"
#include "StDbConfigNode.hh"
#include "StDbTable.h"
#include "StDbLogger.hh"
#include <list>
```

Classes

- class dbType
- class dbDomain
- class dbEnvList
- class StDbManagerImpl

Defines

- #define MAX_ENV_VARS 50

Typedefs

- typedef list< dbType * > dbTypes
- typedef list< dbDomain * > dbDomains
- typedef list< StDbServer * > ServerList

5.28.1 Define Documentation

5.28.1.1 #define MAX_ENV_VARS 50

Definition at line 71 of file StDbManagerImpl.hh.

5.28.2 Typedef Documentation

5.28.2.1 typedef list<dbDomain*> dbDomains

Definition at line 97 of file StDbManagerImpl.hh.

5.28.2.2 typedef list<dbType*> dbTypes

Definition at line 96 of file StDbManagerImpl.hh.

5.28.2.3 typedef list<StDbServer*> ServerList

Definition at line 98 of file StDbManagerImpl.hh.

5.29 StDbMessenger.cc File Reference

```
#include <string.h>
#include "stdb_streams.h"
#include "StDbMessenger.hh"
```

5.30 StDbMessenger.hh File Reference

```
#include "StDbMessService.hh"
#include "stdb_streams.h"
```

Classes

- class StDbMessenger

5.31 StDbMessService.hh File Reference

Classes

- class StDbMessService

Enumerations

- enum StDbMessLevel {
  dbMDebug = 0, dbMWarn, dbMConnect, dbMErr,
  dbMFatal }

5.31.1 Enumeration Type Documentation

5.31.1.1 enum StDbMessLevel

Enumeration values:

dbMDebug

dbMWarn

dbMConnect

dbMErr

dbMFatal

Definition at line 41 of file StDbMessService.hh.

```
41 { dbMDebug=0, dbMWarn, dbMConnect, dbMErr, dbMFatal };
```

5.32 StDbModifier.cxx File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <Stiostream.h>
#include "TROOT.h"
#include "TInterpreter.h"
#include "TTable.h"
#include "StDbLib/StDbManager.hh"
#include "StDbLib/StDbConfigNode.hh"
#include "StDbLib/StDbTable.h"
#include "StDbLib/StDbDefs.hh"
#include "StDbLib/StDbTableDescriptor.h"
#include "StDbModifier.h"
```

Functions

- ClassImp (StDbModifier) StDbModifier

5.32.1 Function Documentation

5.32.1.1 ClassImp (StDbModifier)

Definition at line 49 of file StDbModifier.cxx.

```
                              : funixTime(0), fTimestamp(0)
{
  // constructor of StDbModifier


  //  fTimestamp = new char[30];
  //  strcpy(fTimestamp,"2038-01-01 04:59:59");
  // Timestamp of the data requested.
  // To get recent one, here, it is assigned as end of UNIX time.

  fDebug = 0;        // set No-debug mode.

  fDbName    = 0;    // set Database name on DB server  as brank.
  fTableName = 0;    // set Table name on DB server  as brank.
                     // If either fDbName or fTableName is still 0
                     // in ReadDataFromBD() and WriteDataToDB(),
                     // the program will be terminated.
  fVersionName = 0;

  fFlavorName = 0;

  fOutputFileName = new char[200];
  strcpy(fOutputFileName,"./database_data.C");
  // default name of a file to be filled data from database.

  fInputFileName = new char[200];
  strcpy(fInputFileName,"./database_data.C");
```

```
// default name of a file to be read data from ascii file.
}
```

5.33 StDbModifier.h File Reference

```
#include "StDbLib/StDbManager.hh"
#include "StDbLib/StDbConfigNode.hh"
#include "StDbLib/StDbTable.h"
#include "StDbLib/StDbDefs.hh"
```

Classes

- class StDbModifier

5.34 StDbNode.cc File Reference

```
#include "StDbNode.hh"
#include "StDbDefaults.hh"
#include "stdb_streams.h"
#include <stdlib.h>
```

5.35 StDbNode.hh File Reference

```
#include "StDbDefs.hh"
#include <string.h>
```

Classes

- class StDbNode

5.36 StDbServer.cc File Reference

```
#include "StDbServer.hh"
#include "StDbTable.h"
#include "StDbConfigNode.hh"
#include <string.h>
```

Defines

- #define __CLASS__ "StDbServer"

5.36.1 Define Documentation

5.36.1.1 #define __CLASS__ "StDbServer"

Definition at line 86 of file StDbServer.cc.

5.37 StDbServer.hh File Reference

```
#include "StDbDefs.hh"
#include "StDataBaseI.hh"
```

Classes

- class StDbServer

5.38 StDbServerImpl.cc File Reference

```
#include "StDbServerImpl.hh"
#include "StDbSql.hh"
```

Defines

- #define __CLASS__ "StDbServerImpl"

5.38.1 Define Documentation

5.38.1.1 #define __CLASS__ "StDbServerImpl"

Definition at line 39 of file StDbServerImpl.cc.

5.39 StDbServerImpl.hh File Reference

```
#include "StDbServer.hh"
#include "MysqlDb.h"
#include "StDbBuffer.h"
#include <list>
```

Classes

- class StDbServerImpl

Typedefs

- typedef list< StDataBaseI * > DbList

5.39.1 Typedef Documentation

5.39.1.1 typedef list<StDataBaseI*> DbList

Definition at line 54 of file StDbServerImpl.hh.

5.40 StDbSql.cc File Reference

```
#include "StDbSql.hh"
#include "StDbManager.hh"
#include "StDbDefaults.hh"
#include "StDbConfigNodeImpl.hh"
#include "StDbTableIter.hh"
#include "StDbTable.h"
#include "StDbFastSqlWriter.h"
#include "stdb_streams.h"
```

Defines

- #define __CLASS__ "StDbSql"
- #define __METHOD__ "QueryDb(StDbConfigNode∗)"
- #define __METHOD__ "QueryDb(StDbNode∗)"
- #define __METHOD__ "QueryDb(table,time)"
- #define __METHOD__ "QueryDb(StDbTable∗, const char∗ where)"
- #define __METHOD__ "QueryDb(StDbTable∗, const char∗ where)"
- #define __METHOD__ "QueryDbFunction(table,whereClause,functionName)"
- #define __METHOD__ "WriteDb(StDbTable∗,uint storeTime)"
- #define __METHOD__ "WriteDbNoIndex(table,storeTime)"
- #define __METHOD__ "QueryDescriptor(StDbTable∗)"
- #define __METHOD__ "WriteDb(node,parentID,configID)"

Variables

- const char ∗ DbQFailed = "Query Failed = "
- const char ∗ DbQInfo = " Query = "
- const char ∗ DbQResult = " Query Result = "

5.40.1 Define Documentation

5.40.1.1 #define __CLASS__ "StDbSql"

Definition at line 140 of file StDbSql.cc.

5.40.1.2 #define __METHOD__ "WriteDb(node,parentID,configID)"

5.40.1.3 #define __METHOD__ "QueryDescriptor(StDbTable*)"

5.40.1.4 #define __METHOD__ "WriteDbNoIndex(table,storeTime)"

5.40.1.5 #define __METHOD__ "WriteDb(StDbTable*,uint storeTime)"

5.40.1.6 #define __METHOD__ "QueryDbFunction(table,whereClause,functionName)"

5.40.1.7 #define __METHOD__ "QueryDb(StDbTable*, const char* where)"

5.40.1.8 #define __METHOD__ "QueryDb(StDbTable*, const char* where)"

5.40.1.9 #define __METHOD__ "QueryDb(table,time)"

5.40.1.10 #define __METHOD__ "QueryDb(StDbNode*)"

5.40.1.11 #define __METHOD__ "QueryDb(StDbConfigNode*)"

5.40.2 Variable Documentation

5.40.2.1 const char* DbQFailed = "Query Failed = " [static]

Definition at line 142 of file StDbSql.cc.

5.40.2.2 const char* DbQInfo = " Query = " [static]

Definition at line 143 of file StDbSql.cc.

5.40.2.3 const char* DbQResult = " Query Result = " [static]

Definition at line 144 of file StDbSql.cc.

5.41 StDbSql.hh File Reference

```
#include "StDbTableDescriptor.h"
#include "StDbManager.hh"
#include "StDataBaseI.hh"
#include "MysqlDb.h"
#include "StDbBuffer.h"
#include <list>
```

Classes

- class StDbSql

Defines

- #define MAX_EID_INDECES 10

Typedefs

- typedef list< StDbTableDescriptor * > DescList

5.41.1 Define Documentation

5.41.1.1 #define MAX_EID_INDECES 10

Definition at line 90 of file StDbSql.hh.

5.41.2 Typedef Documentation

5.41.2.1 typedef list<StDbTableDescriptor*> DescList

Definition at line 87 of file StDbSql.hh.

5.42 StDbStoreInfo.cc File Reference

```
#include "StDbStoreInfo.hh"
#include <string.h>
```

Defines

- #define N_ROWS_INCREMENT 500

5.42.1 Define Documentation

5.42.1.1 #define N_ROWS_INCREMENT 500

Definition at line 36 of file StDbStoreInfo.cc.

5.43 StDbStoreInfo.hh File Reference

Classes

- class StDbStoreInfo

5.44 StDbTable.cc File Reference

```
#include "StDbTable.h"
#include "StDbBuffer.h"
#include "typeAcceptor.hh"
#include "StTableDescriptorI.h"
#include "StDbDefaults.hh"
#include "StDbManager.hh"
#include <string.h>
#include "stdb_streams.h"
#include <malloc.h>
```

Defines

- #define __CLASS__ "StDbTable"

5.44.1 Define Documentation

5.44.1.1 #define __CLASS__ "StDbTable"

Definition at line 343 of file StDbTable.cc.

5.45 StDbTable.h File Reference

```
#include "StDbNode.hh"
#include "StDbTime.h"
#include "StTableDescriptorI.h"
#include "StDbStoreInfo.hh"
#include <string.h>
```

Classes

- class StDbTable

5.46 StDbTableDescriptor.cc File Reference

```
#include "StDbTableDescriptor.h"
#include <stdlib.h>
#include <math.h>
#include "stdb_streams.h"
```

5.47 StDbTableDescriptor.h File Reference

```
#include <string.h>
#include "StTableDescriptorI.h"
#include "StDbBuffer.h"
```

Classes

- struct tableDescriptor
- class StDbTableDescriptor

5.48 StDbTableFactory.cc File Reference

```
#include "StDbTableFactory.hh"
#include "StDbTable.h"
```

5.49 StDbTableFactory.hh File Reference

Classes

- class StDbTableFactory

5.50 StDbTableIter.hh File Reference

```
#include "StDbTable.h"
```

Classes

- class StDbTableIter

5.51 StDbTableIterImpl.cc File Reference

```
#include "StDbTableIterImpl.hh"
```

5.52 StDbTableIterImpl.hh File Reference

```
#include "StDbConfigNodeImpl.hh"
```

Classes

- class StDbTableIterImpl

Typedefs

- typedef TableList::iterator ListIter

5.52.1 Typedef Documentation

5.52.1.1 typedef TableList::iterator ListIter

Definition at line 34 of file StDbTableIterImpl.hh.

5.53 StDbTime.h File Reference

```
#include <string.h>
```

Classes

- class StDbTime

5.54 StDbXmlReader.cc File Reference

```
#include "StDbXmlReader.h"
#include "stdb_streams.h"
#include <string.h>
#include "dbStruct.hh"
```

Functions

- template<class T> void passAux (elem *e, T *&i, int &len)

5.54.1 Function Documentation

5.54.1.1 template<class T> void passAux (elem * *e*, T *& *i*, int & *len*) [static]

Definition at line 70 of file StDbXmlReader.cc.

```
{
  len = e->size.isize;
  i = new T[len];
  memset(i,0,sizeof(T)*len);
  const char* p1 = e->val.data;
  const char* p2=p1;
  int iloop = 0;
  for(;1;p2++) {
     if (p2[0] && p2[0]!=',')           continue;
     i[iloop++] = (T)atof(p1);
     p1=p2+1;
     if (iloop==len || p2[0]==0)        break;
  }
}
```

5.55 StDbXmlReader.h File Reference

```
#include <stdlib.h>
#include <string.h>
#include "typeAcceptor.hh"
#include "stdb_streams.h"
```

Classes

- class StDbXmlReader

5.56 StDbXmlWriter.cc File Reference

```
#include "StDbXmlWriter.h"
#include "StDbTable.h"
```

5.57 StDbXmlWriter.h File Reference

```
#include <stdlib.h>
#include <string.h>
#include "stdb_streams.h"
#include "tableAcceptor.hh"
```

Classes

- class StDbXmlWriter

5.58 StString.cc File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include "StString.h"
```

5.59 StString.h File Reference

```
#include <string>
```

Classes

- class StString

Defines

- #define stendl ("\n")

5.59.1 Define Documentation

5.59.1.1 #define stendl ("\n")

Definition at line 28 of file StString.h.

5.60 StTableDescriptorI.h File Reference

Classes

- class StTableDescriptorI

Enumerations

- enum StTypeE {
  Stchar, Stuchar, Stshort, Stushort,
  Stint, Stuint, Stlong, Stulong,
  Stlonglong, Stfloat, Stdouble }

Variables

- const int StTypeSize [ ] = {sizeof(char),sizeof(unsigned char),sizeof(short),sizeof(unsigned short), sizeof(int),sizeof(unsigned int),sizeof(long),sizeof(unsigned long), sizeof(long long),sizeof(float),sizeof(double)}

5.60.1 Enumeration Type Documentation

5.60.1.1 enum StTypeE

Enumeration values:

Stchar

Stuchar

Stshort

Stushort

Stint

Stuint

Stlong

Stulong

Stlonglong

Stfloat

Stdouble

Definition at line 60 of file StTableDescriptorI.h.

```
60 {Stchar,Stuchar,Stshort,Stushort,Stint,Stuint,Stlong,Stulong,Stlonglong,Stfloat,Stdouble};
```

5.60.2 Variable Documentation

5.60.2.1 const int StTypeSize[ ] = {sizeof(char),sizeof(unsigned char),sizeof(short),sizeof(unsigned short), sizeof(int),sizeof(unsigned int),sizeof(long),sizeof(unsigned long), sizeof(long long),sizeof(float),sizeof(double)}

Definition at line 63 of file StTableDescriptorI.h.

5.61 tableAcceptor.hh File Reference

```
#include "typeAcceptor.hh"
```

Classes

- class tableAcceptor

5.62 typeAcceptor.hh File Reference

Classes

- class typeAcceptor

Index

~MysqlDb
MysqlDb, 27
~StDataBaseI
StDataBaseI, 51
~StDbBuffer
StDbBuffer, 61
~StDbBufferI
StDbBufferI, 76
~StDbConfigNode
StDbConfigNode, 84
~StDbConfigNodeImpl
StDbConfigNodeImpl, 94
~StDbDefaults
StDbDefaults, 105
~StDbElementIndex
StDbElementIndex, 109
~StDbFastSqlWriter
StDbFastSqlWriter, 114
~StDbLogger
StDbLogger, 121
~StDbManager
StDbManager, 125
~StDbManagerImpl
StDbManagerImpl, 138
~StDbMessService
StDbMessService, 170
~StDbMessenger
StDbMessenger, 167
~StDbModifier
StDbModifier, 172
~StDbNode
StDbNode, 181
~StDbServer
StDbServer, 190
~StDbServerImpl
StDbServerImpl, 197
~StDbSql
StDbSql, 204
~StDbStoreInfo
StDbStoreInfo, 233
~StDbTable
StDbTable, 239
~StDbTableDescriptor
StDbTableDescriptor, 270
~StDbTableFactory
StDbTableFactory, 280
~StDbTableIter
StDbTableIter, 281
~StDbTableIterImpl
StDbTableIterImpl, 283
~StDbTime
StDbTime, 287
~StDbXmlReader
StDbXmlReader, 290
~StDbXmlWriter
StDbXmlWriter, 302
~StTableDescriptorI
StTableDescriptorI, 317
~accessor
accessor, 7
~basic
basic, 9
~dbEnvList
dbEnvList, 15
~dbRow
dbRow, 17
~dbTable
dbTable, 19
~parseXmlString
parseXmlString, 46
~tableAcceptor
tableAcceptor, 320
~typeAcceptor
typeAcceptor, 323
__CLASS__
MysqlDb.cc, 330
StDbConfigNode.cc, 349
StDbConfigNodeImpl.cc, 351
StDbManagerImpl.cc, 365
StDbServer.cc, 377
StDbServerImpl.cc, 379
StDbSql.cc, 381
StDbTable.cc, 386
__METHOD__
MysqlDb.cc, 330
StDbConfigNodeImpl.cc, 351
StDbManagerImpl.cc, 365, 366
StDbSql.cc, 381, 382
_ascii
StDbBuffer.h, 347

_char
StDbBuffer.h, 346
_double
StDbBuffer.h, 347
_float
StDbBuffer.h, 347
_int
StDbBuffer.h, 347
_long
StDbBuffer.h, 347
_longlong
StDbBuffer.h, 347
_short
StDbBuffer.h, 346
_string
StDbBuffer.h, 347
_uchar
StDbBuffer.h, 346
_uint
StDbBuffer.h, 347
_ulong
StDbBuffer.h, 347
_ushort
StDbBuffer.h, 347

a
dbTable, 19
accessor, 7
~accessor, 7
accessor, 7
e, 8
nelems, 8
addChildren
StDbConfigNode, 84
StDbConfigNodeImpl, 95
addDataBase
StDbServer, 191
StDbServerImpl, 197
addDbDomain
StDbManagerImpl, 138
addDbTable
StDbConfigNode, 84
StDbConfigNodeImpl, 95
addDbType
StDbManagerImpl, 138
addDescriptor
StDbSql, 204
addElementIndex
StDbElementIndex, 110
AddField
StDbBuffer, 62
addNameValuePair
StDbElementIndex, 110
addNElements
StDbTable, 240
addNRows
StDbTable, 240
AddRows
StDbTable, 240
addTable
StDbConfigNode, 84
StDbConfigNodeImpl, 95
addWrittenNode
StDbNode, 181
addWrittenRow
StDbStoreInfo, 233
addWrittenRows
StDbTable, 241
appendNode
StDbConfigNode, 84
Auto
StDbBuffer.h, 346

basic, 9
~basic, 9
endKey, 10
iend, 10
istart, 10
setEndKey, 9
setStartKey, 9
startKey, 10
Binary
MysqlDb.cc, 332
MysqlDb.h, 333
binaryMessage
MysqlDb.cc, 332
buff
StDbServerImpl, 200
StDbSql, 232
BuffMode
StDbBuffer.h, 346
buildDbTable
StDbXmlReader, 290
buildStruct
StDbXmlReader, 292
buildTree
StDbConfigNode, 84
StDbConfigNodeImpl, 96

canRollBack
StDbNode, 181
StDbStoreInfo, 234
castcase
StDbBuffer.cc, 341
castcasest
StDbBuffer.cc, 341
ChangeField
StDbBuffer, 62

checkColumn
StDbSql, 204
checkDbDomain
StDataBaseI, 52
checkDbName
StDataBaseI, 52
checkDbType
StDataBaseI, 52
checkDescriptor
StDbTable, 241
checkForNull
StDbSql, 204
checkForTable
MysqlDb, 28
checkName
StDbNode, 181
checkNode
StDbNode, 181
checkTableCatalog
StDbSql, 205
checkTablePrepForQuery
StDbSql, 205
checkValue
StDbSql, 205
checkVersion
StDbNode, 182
ClassImp
StDbModifier.cxx, 372
clear
StDbSql, 206
clearIndex
StDbElementIndex, 110
clearStoreInfo
StDbTable, 241
Client
StDbBuffer.h, 346
Clone
StDbTable, 242
Close
MysqlDb, 28
close
StDataBaseI, 52
StDbSql, 206
closeAllConnections
StDbManager, 125
StDbManagerImpl, 138, 139
closeConnection
StDbManager, 125
StDbManagerImpl, 139
StDbServer, 191
StDbServerImpl, 197
cnames
MysqlDb, 41
CodeStrArray
MysqlDb, 28
column, 11
length, 11
name, 11
type, 11
val, 11
commit
StDbNode, 182
StDbStoreInfo, 234
commitAllNodes
StDbManager, 126
StDbManagerImpl, 139
commitAllTables
StDbManager, 126
StDbManagerImpl, 139
commitData
StDbTable, 242
compareTables
StDbConfigNode, 85
StDbConfigNodeImpl, 96
conn
MysqlDb.cc, 332
Connect
MysqlDb, 29
CR_COMMANDS_OUT_OF_SYNC
MysqlDb.cc, 330
CR_CONN_HOST_ERROR
MysqlDb.cc, 330
CR_CONNECTION_ERROR
MysqlDb.cc, 330
CR_IPSOCK_ERROR
MysqlDb.cc, 330
CR_LOCALHOST_CONNECTION
MysqlDb.cc, 330
CR_MAX_ERROR
MysqlDb.cc, 330
CR_MIN_ERROR
MysqlDb.cc, 330
CR_NAMEDPIPE_CONNECTION
MysqlDb.cc, 330
CR_NAMEDPIPEOPEN_ERROR
MysqlDb.cc, 330
CR_NAMEDPIPESETSTATE_ERROR
MysqlDb.cc, 330
CR_NAMEDPIPEWAIT_ERROR
MysqlDb.cc, 331
CR_OUT_OF_MEMORY
MysqlDb.cc, 331
CR_SERVER_GONE_ERROR
MysqlDb.cc, 331
CR_SERVER_HANDSHAKE_ERR
MysqlDb.cc, 331
CR_SERVER_LOST
MysqlDb.cc, 331

CR_SOCKET_CREATE_ERROR
MysqlDb.cc, 331
CR_TCP_CONNECTION
MysqlDb.cc, 331
CR_UNKNOWN_ERROR
MysqlDb.cc, 331
CR_UNKNOWN_HOST
MysqlDb.cc, 331
CR_VERSION_ERROR
MysqlDb.cc, 331
CR_WRONG_HOST_INFO
MysqlDb.cc, 331
createMemory
StDbTable, 242, 243
curRow
dbTable, 19

data
datav, 12
datav, 12
data, 12
datav, 12
Db
StDbServerImpl, 200
StDbSql, 232
dbBbc
StDbDefs.hh, 356
dbCalibrations
StDbDefs.hh, 357
dbCollection, 13
dbCollection
name, 13
dbCollection.h, 327
dbConditions
StDbDefs.hh, 356
dbConfigurations
StDbDefs.hh, 356
dbCtb
StDbDefs.hh, 355
dbDaq
StDbDefs.hh, 356
dbDEnd
StDbDefs.hh, 356
dbDomain, 14
dbDomain, 14
dbDomain
dbDomain, 14
domain, 14
name, 14
dbDomainFree
StDbManagerImpl, 164
dbDomains
StDbManagerImpl.hh, 367
dbDomainUnknown
StDbDefs.hh, 355
dbDUser1
StDbDefs.hh, 356
dbDUser2
StDbDefs.hh, 356
dbDUser3
StDbDefs.hh, 356
dbEemc
StDbDefs.hh, 356
dbEmc
StDbDefs.hh, 355
dbEnvList, 15
dbEnvList, 15
dbEnvList
~dbEnvList, 15
dbEnvList, 15
envDef, 15
envVar, 15
num, 15
dbFindServerMode
StDbDefs.hh, 355
dbFpd
StDbDefs.hh, 356
dbFtpc
StDbDefs.hh, 355
dbGeometry
StDbDefs.hh, 357
dbGlobal
StDbDefs.hh, 356
dbL3
StDbDefs.hh, 356
DbList
StDbServerImpl.hh, 380
dbMConnect
StDbMessService.hh, 371
dbMDebug
StDbMessService.hh, 371
dbMErr
StDbMessService.hh, 371
dbMFatal
StDbMessService.hh, 371
dbMWarn
StDbMessService.hh, 371
dbMwc
StDbDefs.hh, 356
dbOnl
StDbDefs.hh, 356
dbPmd
StDbDefs.hh, 356
DbQFailed
StDbSql.cc, 382
DbQInfo
StDbSql.cc, 382
DbQResult

StDbSql.cc, 382
dbRhic
StDbDefs.hh, 356
dbRich
StDbDefs.hh, 356
dbRow, 17
dbRow, 17
dbRow
~dbRow, 17
dbRow, 17
rowID, 17
rowNumber, 17
dbRts
StDbDefs.hh, 356
dbRunCatalog
StDbDefs.hh, 357
dbRunLog
StDbDefs.hh, 356
dbRunParams
StDbDefs.hh, 357
dbScaler
StDbDefs.hh, 356
dbScalers
StDbDefs.hh, 357
dbServer
StDbDefs.hh, 356
dbSsd
StDbDefs.hh, 356
dbStar
StDbDefs.hh, 355
dbStDb
StDbDefs.hh, 356
dbStreamer
StDbTable, 243
dbStreamerWrite
StDbTable, 244
dbStruct.hh, 328
dbStruct.hh
elemVec, 328
rowVec, 328
dbSvt
StDbDefs.hh, 355
dbTable, 19
dbTable, 19
dbTable
~dbTable, 19
a, 19
curRow, 19
dbTable, 19
name, 20
numRows, 20
row, 20
dbTableStreamer
StDbTable, 244
dbTEnd
StDbDefs.hh, 357
dbTestScheme
StDbDefs.hh, 357
dbTof
StDbDefs.hh, 356
dbTpc
StDbDefs.hh, 355
dbTracker
StDbDefs.hh, 356
dbTrg
StDbDefs.hh, 355
dbTUser1
StDbDefs.hh, 357
dbTUser2
StDbDefs.hh, 357
dbTUser3
StDbDefs.hh, 357
dbType, 21
dbType, 21
dbType
dbType, 21
name, 21
type, 21
dbTypeFree
StDbManagerImpl, 164
dbTypes
StDbManagerImpl.hh, 367
dbV00
StDbDefs.hh, 356
dbZdc
StDbDefs.hh, 356
decodeElementID
StDbNode, 182
DecodeStrArray
MysqlDb, 30
defaultFlavor
StDbTable, 245
deleteChildren
StDbConfigNode, 85
deleteDataBases
StDbServerImpl, 198
deleteDescriptor
StDbSql, 206
deleteDescriptors
StDbSql, 206
deleteDomains
StDbManagerImpl, 140
deleteRows
StDbSql, 206
deleteServers
StDbManagerImpl, 140
deleteTables
StDbConfigNodeImpl, 97

deleteTree
StDbConfigNode, 85
deleteTypes
StDbManagerImpl, 141
DescList
StDbSql.hh, 383
dimensionlen
tableDescriptor, 322
domain
dbDomain, 14
done
StDbTableIter, 281
StDbTableIterImpl, 284
duplicateData
StDbTable, 245

e
accessor, 8
elem, 22
elem, 22
name, 22
size, 22
type, 22
val, 22
elemVec
dbStruct.hh, 328
end
StDbLogger, 121
endAccessor
StDbFastSqlWriter, 114
StDbXmlWriter, 302
tableAcceptor, 320
endKey
basic, 10
endRowPadding
StDbTableDescriptor, 270
endsql
MysqlDb.h, 333
envDef
dbEnvList, 15
environ
StDbManagerImpl.cc, 366
envVar
dbEnvList, 15
ExecQuery
MysqlDb, 31

fDbName
StDbModifier, 177
fDebug
StDbModifier, 177
fetchAllTables
StDbManager, 126
StDbManagerImpl, 141
fetchDbTable
StDbManager, 126
StDbManagerImpl, 141, 142
fFlavorName
StDbModifier, 177
fillElement
StDbTableDescriptor, 270
fillElements
StDbXmlReader, 292
fillLengths
StDbTableDescriptor, 271
fillSizeAndOffset
StDbTableDescriptor, 271
Find_Col
StDbBuffer, 63
findChildConfigNode
StDbConfigNode, 85
findConfigNode
StDbConfigNode, 85–87
findDb
StDbManager, 126
StDbManagerImpl, 142
findDefaultServer
StDbManager, 126
StDbManagerImpl, 143
findDescriptor
StDbSql, 207
findElement
StDbXmlReader, 294
findLocalTable
StDbConfigNode, 87
StDbConfigNodeImpl, 97
findServer
StDbManager, 126, 127
StDbManagerImpl, 143, 144
findServerString
StDbManagerImpl, 144
findServersXml
StDbManagerImpl, 145
findTable
StDbConfigNode, 87
StDbConfigNodeImpl, 97
fInputFileName
StDbModifier, 177
fOutputFileName
StDbModifier, 178
fPrec
StString, 316
fTableName
StDbModifier, 178
fTimestamp
StDbModifier, 178
funixTime
StDbModifier, 178

fVersionName
StDbModifier, 178

genwritemem
StDbBuffer.cc, 341, 345
getBeginDateTime
StDbTable, 245
getBeginTime
StDbTable, 246
getBranchID
StDbConfigNode, 87
getColumnList
StDbSql, 207
getConfigNodeName
StDbManager, 127
StDbManagerImpl, 147
getConnectTimes
StDbServer, 191
StDbServerImpl, 198
getCpy
StDbTableDescriptor, 273
StTableDescriptorI, 317
getCstructName
StDbTable, 246
getCurrentInternalSize
StDbTableDescriptor, 273
getDataBaseInfo
StDbManager, 127
StDbManagerImpl, 147, 148
getDataIDs
StDbStoreInfo, 234
getDataTable
StDbSql, 207
StDbTable, 246
getDataTables
StDbSql, 208
getDataValue
StDbTable, 246
getDateCheckTime
StDbManager, 127
StDbManagerImpl, 148
getDateRequestTime
StDbManager, 127
StDbManagerImpl, 148
getDateStoreTime
StDbManager, 127
StDbManagerImpl, 149
getDateTime
StDataBaseI, 52
StDbSql, 208
StDbTime, 287
getDbDomain
StDataBaseI, 52
StDbManager, 127
StDbManagerImpl, 149
StDbNode, 183
getDbDomainName
StDbManager, 128
StDbManagerImpl, 149
getDbName
StDataBaseI, 52
StDbManager, 128
StDbManagerImpl, 150
StDbNode, 184
getDbStoreType
StDataBaseI, 53
getDbType
StDataBaseI, 53
StDbManager, 128
StDbManagerImpl, 150
StDbNode, 184
getDbTypeName
StDbManager, 128
StDbManagerImpl, 150
getDescriptor
StDbSql, 209
StDbTable, 246
getDescriptorCpy
StDbTable, 247
getDomainName
StDataBaseI, 53
getElementDimensions
StDbTableDescriptor, 273
StTableDescriptorI, 317
getElementID
StDbTable, 247
getElementIndex
StDbConfigNode, 87
StDbConfigNodeImpl, 98
getElementIndexInfo
StDbConfigNode, 87
StDbConfigNodeImpl, 98
getElementIndexLength
StDbTableDescriptor, 273
StTableDescriptorI, 317
getElementLength
StDbTableDescriptor, 273
StTableDescriptorI, 318
getElementList
StDbSql, 209
getElementListIN
StDbSql, 210
getElementName
StDbTable, 247
StDbTableDescriptor, 274
StTableDescriptorI, 318
getElementNumDimensions
StDbTableDescriptor, 274

StTableDescriptorI, 318
getElementOffset
StDbTableDescriptor, 274
StTableDescriptorI, 318
getElementSize
StDbTableDescriptor, 274
StTableDescriptorI, 318
getElementSpecs
StDbTable, 247
getElementType
StDbTableDescriptor, 275
StTableDescriptorI, 318
getEmptyString
StDbSql, 210
getEndDateTime
StDbTable, 247
getEndStoreDateTime
StDbTable, 248
getEndStoreTime
StDbTable, 248
getEndTime
StDbDefaults, 105
StDbTable, 248
getEnvList
StDbManager, 128
StDbManagerImpl, 151
getExternalVersion
StDbManager, 128
StDbManagerImpl, 151
getFileName
StDbDefaults, 105
getFirstChildNode
StDbConfigNode, 87
getFlavor
StDbTable, 248
getFlavorQuery
StDbSql, 210
getHostName
StDbServer, 191
getIndexAfter
parseXmlString, 46
getIndexBefore
parseXmlString, 46
getIndexName
StDbElementIndex, 110
getIndexNames
StDbSql, 211
getIndexVal
StDbElementIndex, 111
GetLastInsertID
MysqlDb, 32
getMaxTime
StDbTable, 248
getMessenger
StDbManager, 128
getMessLevel
StDbMessService, 170
getMyName
StDbNode, 184
getName
StDbNode, 184
getNextID
StDbNode, 184
getNextIndex
StDbElementIndex, 111
getNextName
StDbManagerImpl, 152
getNextNode
StDbConfigNode, 87
getNodeID
StDbNode, 184
getNodeType
StDbNode, 185
GetNRows
StDbTable, 248
getNumberStats
StDbConfigNode, 88
StDbConfigNodeImpl, 98
getNumCalls
StDbLogger, 121
getNumElements
StDbTableDescriptor, 275
StTableDescriptorI, 318
getNumIndeces
StDbConfigNode, 88
StDbConfigNodeImpl, 98
StDbElementIndex, 111
getParentNode
StDbConfigNode, 88
getPortNumber
StDbServer, 191
getProdTime
StDbDefaults, 105
StDbTable, 248
getProdTimeQuery
StDbSql, 211
getQueryTimes
StDbServer, 191
StDbServerImpl, 198
getRowID
StDbTable, 249
getRowLimit
StDbTable, 249
getSchemaID
StDbTable, 249
StDbTableDescriptor, 275
getServerFileName
StDbDefaults, 106

getServerName
StDbServer, 191
getSize
StDbTableDescriptor, 275
getSocketTimes
StDbServer, 191
StDbServerImpl, 198
getStDbTableIter
StDbConfigNode, 88
StDbConfigNodeImpl, 99
getString
parseXmlString, 47
getStringAfter
parseXmlString, 47
getStringBefore
parseXmlString, 48
getStructID
StDbTableDescriptor, 275
GetTable
StDbTable, 249
GetTableCpy
StDbTable, 249
getTableDescriptor
StDbTableDescriptor, 275
getTableSize
StDbTable, 249
getTimeValues
StDbTable, 250
getTotalSizeInBytes
StDbTableDescriptor, 276
StTableDescriptorI, 318
getTotalTimes
StDbLogger, 122
getTrowSize
StDbTableDescriptor, 276
StTableDescriptorI, 318
getType
StDbTableDescriptor, 276
getTypeName
StDataBaseI, 53
getUnixCheckTime
StDbManager, 128
StDbManagerImpl, 152
getUnixRequestTime
StDbManager, 128
StDbManagerImpl, 152
getUnixSocket
StDbServer, 191
getUnixStoreTime
StDbManager, 128
StDbManagerImpl, 152
getUnixTime
StDataBaseI, 53
StDbSql, 211
StDbTime, 287
getVersion
StDbNode, 185
getWrittenRows
StDbTable, 250

hasChildren
StDbConfigNode, 88
hasData
StDbConfigNode, 88
StDbStoreInfo, 234
StDbTable, 250
hasDescriptor
StDbTable, 250
hasInstance
StDbSql, 212

iend
basic, 10
iname
indexNameVals, 24
indexNameVals, 24
indexNameVals
iname, 24
ival, 24
Init
StDbBuffer, 63
init
StDbSql, 212
StDbTable, 250
StDbTableDescriptor, 276
StDbTableIterImpl, 284
initConfig
StDbManager, 129
StDbManagerImpl, 152–154
initDomains
StDbManagerImpl, 154
initEndTime
StDbSql, 212
initTypes
StDbManagerImpl, 155
Input
MysqlBin, 25
MysqlDb, 32
InputEnd
MysqlDb, 33
InputRow
MysqlDb, 33
InputStart
MysqlDb, 34
insertNodeRelation
StDataBaseI, 53
StDbSql, 212
insertNodeString

StDbSql, 213
Instance
StDbDefaults, 106
StDbManager, 129
ioTable
StDbFastSqlWriter, 114
StDbXmlWriter, 302
tableAcceptor, 320
IsBaseLine
StDbTable, 251
IsBinary
StDbTable, 251
isBinary
MysqlDb, 41
isBlob
MysqlDb, 41
IsClientMode
StDbBuffer, 63
StDbBufferI, 76
IsConfigured
StDbNode, 185
IsConnected
MysqlDb, 35
isConnected
StDbServer, 192
StDbServerImpl, 198
isDbNode
StDbConfigNode, 88
isDefault
StDbServer, 192
IsDefaultFlavor
StDbDefaults, 106
IsDefaultVersion
StDbDefaults, 106
IsIndexed
StDbTable, 251
isize
stsize, 312
isNode
StDbNode, 185
IsQuiet
StDbManager, 129
isSpecialType
MysqlDb, 41
IsStorageMode
StDbBuffer, 63
StDbBufferI, 76
IsTable
StDbNode, 185
StDbTable, 251
istart
basic, 10
IsValid
StDbManager, 130
StDbManagerImpl, 155
StDbTableDescriptor, 277
IsVerbose
StDbManager, 130
itr
StDbTableIterImpl, 284
ival
indexNameVals, 24

jfields
MysqlDb, 41

lastType
StDbTableDescriptor, 278
length
column, 11
ListIter
StDbTableIterImpl.hh, 394
loca
StDbXmlReader, 300
lookUpServers
StDbManagerImpl, 155

MAX_EID_INDECES
StDbSql.hh, 383
MAX_ENV_VARS
StDbManagerImpl.hh, 367
maxlines
StDbXmlReader, 300
mbeginTime
StDbTable, 265
mBinData
MysqlBin, 25
mbranchID
StDbConfigNode, 91
mcanRollBack
StDbNode, 188
StDbStoreInfo, 235
mcheckTime
StDbManagerImpl, 164
mCol
StDbBuffer, 74
mcols
StDbTableDescriptor, 278
mconnectLog
MysqlDb, 41
mCur
StDbBuffer, 74
StDbTableDescriptor, 278
mcurrent
StDbElementIndex, 112
mcurrentDb
StDbServerImpl, 200
mData

MysqlDb, 41
mdata
StDbTable, 265
mdataBases
StDbServerImpl, 200
mdataIDs
StDbStoreInfo, 235
mdataLog
StDbManagerImpl, 164
mdataTable
StDbTable, 265
mdateTime
StDbTime, 288
mdbDomain
StDataBaseI, 58
StDbNode, 188
mdbhost
MysqlDb, 41
mdbLevel
StDbMessService, 171
mdbName
MysqlDb, 41
StDataBaseI, 58
StDbNode, 188
mdbPort
MysqlDb, 42
mdbpw
MysqlDb, 42
mdbServerFile
StDbDefaults, 107
mdbServerVar
StDbDefaults, 107
mdbServerVersion
MysqlDb, 42
mdbStore
StDataBaseI, 58
mdbType
StDataBaseI, 58
StDbNode, 188
mdbuser
MysqlDb, 42
mdefaultEndDateTime
StDbSql, 232
mdefaultEndTime
StDbSql, 232
mdefaultFlavor
StDbTable, 265
mdescriptor
StDbTable, 265
mdescriptors
StDbSql, 232
mdomainName
StDataBaseI, 58
mDomains
StDbManagerImpl, 164
melementID
StDbTable, 265
melementIndex
StDbConfigNodeImpl, 103
melementName
StDbTable, 265
MemSwapCpy
StDbBuffer, 63
mendStoreTime
StDbTable, 265
mendTime
StDbDefaults, 107
StDbTable, 265
Messenger
StDbManager, 134
mfactory
StDbManagerImpl, 165
mfirstChildNode
StDbConfigNode, 91
mflavor
StDbDefaults, 107
StDbTable, 265
mgr
StDbSql, 232
mhasBinaryQuery
MysqlDb, 42
mhasConnected
MysqlDb, 42
mhasData
StDbConfigNode, 91
StDbTable, 265
mhasDefaultServer
StDbManagerImpl, 165
mhasDescriptor
StDbTable, 266
mhasDouble
StDbTableDescriptor, 278
mhasServerList
StDbManagerImpl, 165
mhostName
StDbServer, 194
mindexRef
StDbConfigNodeImpl, 103
mInstance
StDbDefaults, 107
StDbManager, 134
mIsBaseLine
StDbTable, 266
mIsBinary
StDbTable, 266
misConfigured
StDbNode, 188
misDbNode

StDbConfigNode, 92
misDefault
StDbServer, 194
mIsIndexed
StDbTable, 266
misQuiet
StDbManager, 134
misTimeLogged
StDbManager, 134
misValid
StDbTableDescriptor, 278
misVerbose
StDbManager, 134
mLast
StDbBuffer, 74
mLen
MysqlBin, 25
mlogTime
MysqlDb, 42
mMax
StDbBuffer, 74
StDbTableDescriptor, 278
mMaxRows
StDbStoreInfo, 235
mMode
StDbBuffer, 74
mname
StDbNode, 188
mnextNode
StDbConfigNode, 92
mnode
StDbTableIterImpl, 284
mnodeID
StDbNode, 188
mnodeLog
StDbManagerImpl, 165
mnodeType
StDbNode, 188
mnTotal
StDbLogger, 122
mnumElements
StDbTableDescriptor, 278
mnumIndeces
StDbElementIndex, 112
mnumRows
StDbStoreInfo, 235
mnvals
StDbElementIndex, 112
mos
StDbMessenger, 169
mparentNode
StDbConfigNode, 92
mparser
StDbManagerImpl, 165
mportNumber
StDbServer, 194
mprodTime
StDbDefaults, 107
StDbTable, 266
mpWord
StDbManager, 134
mpword
StDbServer, 194
mQuery
MysqlDb, 42
mQueryLast
MysqlDb, 42
mQueryLen
MysqlDb, 42
mqueryLog
MysqlDb, 42
mqueryState
MysqlDb, 43
mRes
MysqlDb, 43
MysqlResult, 45
mRetString
StDbSql, 213
mretString
StDbSql, 232
mRow
MysqlResult, 45
mrowNumber
StDbTable, 266
mrows
StDbTable, 266
mrowsRequested
StDbTable, 266
mschemaID
StDbTable, 266
StDbTableDescriptor, 278
mSep
MysqlResult, 45
mserverName
StDbServer, 194
mservers
StDbManagerImpl, 165
msocketLog
MysqlDb, 43
mstoredData
StDbTable, 266
mstoreTime
StDbManagerImpl, 165
mstrDup
StDbNode, 185
mstringDup
StDbManager, 130
StDbServer, 192

mstructID
StDbTableDescriptor, 278
mstructName
StDbTable, 266
mt0
StDbLogger, 122
mtableCatalog
StDbSql, 232
mTables
StDbConfigNodeImpl, 103
mtableSize
StDbTableDescriptor, 279
mtimeout
MysqlDb, 43
mtimeVals
StDbTable, 267
mtotalTimes
StDbLogger, 122
mtypeName
StDataBaseI, 58
mTypes
StDbManagerImpl, 165
munixSocket
StDbServer, 194
munixTime
StDbTime, 288
muserName
StDbManager, 134
StDbServer, 194
mversion
StDbDefaults, 107
StDbNode, 188
my_manager
MysqlDb, 43
mycsize
StDbBuffer.h, 347
mycswapl
StDbBuffer.h, 347
myctype
StDbBuffer.h, 346
MysqlBin, 25
MysqlBin, 25
MysqlBin
Input, 25
mBinData, 25
mLen, 25
MysqlBin, 25
MysqlDb, 26
MysqlDb, 27
MysqlDb
~MysqlDb, 27
checkForTable, 28
Close, 28
cnames, 41
CodeStrArray, 28
Connect, 29
DecodeStrArray, 30
ExecQuery, 31
GetLastInsertID, 32
Input, 32
InputEnd, 33
InputRow, 33
InputStart, 34
isBinary, 41
isBlob, 41
IsConnected, 35
isSpecialType, 41
jfields, 41
mconnectLog, 41
mData, 41
mdbhost, 41
mdbName, 41
mdbPort, 42
mdbpw, 42
mdbServerVersion, 42
mdbuser, 42
mhasBinaryQuery, 42
mhasConnected, 42
mlogTime, 42
mQuery, 42
mQueryLast, 42
mQueryLen, 42
mqueryLog, 42
mqueryState, 43
mRes, 43
msocketLog, 43
mtimeout, 43
my_manager, 43
MysqlDb, 27
NbFields, 35
NbRows, 35
operator<<, 35–37
Output, 37
printQuery, 38
QueryStatus, 39
RazQuery, 39
RecommendedServer, 39
reConnect, 39
Release, 40
setDefaultDb, 40
MysqlDb.cc, 329
MysqlDb.cc
__CLASS__, 330
__METHOD__, 330
Binary, 332
binaryMessage, 332
conn, 332
CR_COMMANDS_OUT_OF_SYNC, 330

CR_CONN_HOST_ERROR, 330
CR_CONNECTION_ERROR, 330
CR_IPSOCK_ERROR, 330
CR_LOCALHOST_CONNECTION, 330
CR_MAX_ERROR, 330
CR_MIN_ERROR, 330
CR_NAMEDPIPE_CONNECTION, 330
CR_NAMEDPIPEOPEN_ERROR, 330
CR_NAMEDPIPESETSTATE_ERROR, 330
CR_NAMEDPIPEWAIT_ERROR, 331
CR_OUT_OF_MEMORY, 331
CR_SERVER_GONE_ERROR, 331
CR_SERVER_HANDSHAKE_ERR, 331
CR_SERVER_LOST, 331
CR_SOCKET_CREATE_ERROR, 331
CR_TCP_CONNECTION, 331
CR_UNKNOWN_ERROR, 331
CR_UNKNOWN_HOST, 331
CR_VERSION_ERROR, 331
CR_WRONG_HOST_INFO, 331
MysqlDb.h, 333
MysqlDb.h
Binary, 333
endsql, 333
MysqlResult, 44
MysqlResult, 44
MysqlResult
mRes, 45
mRow, 45
mSep, 45
MysqlResult, 44
NbFields, 44
NbRows, 44
operator++, 44
Release, 44

N_MAX_INDEXVALS
StDbElementIndex.hh, 359
N_ROWS_INCREMENT
StDbStoreInfo.cc, 384
name
column, 11
dbCollection, 13
dbDomain, 14
dbTable, 20
dbType, 21
elem, 22
tableDescriptor, 322
NbFields
MysqlDb, 35
MysqlResult, 44
NbRows
MysqlDb, 35
MysqlResult, 44
nelems
accessor, 8
newDbDomain
StDbManagerImpl, 156
newDbTable
StDbManager, 130
StDbManagerImpl, 157
StDbTableFactory, 280
newDbType
StDbManagerImpl, 157
next
StDbTableIter, 281
StDbTableIterImpl, 284
num
dbEnvList, 15
numRows
dbTable, 20

offset
tableDescriptor, 322
offsetToLast4Bytes
StDbTableDescriptor, 279
offsetToNextEmptyByte
StDbTableDescriptor, 279
operator++
MysqlResult, 44
StDbTableIter, 281
StDbTableIterImpl, 284
operator<<
MysqlDb, 35–37
StString, 313–315
os
StDbFastSqlWriter, 120
StDbXmlWriter, 311
Output
MysqlDb, 37

padsize
StDbTableDescriptor, 279
parseXmlString, 46
parseXmlString, 46
parseXmlString
~parseXmlString, 46
getIndexAfter, 46
getIndexBefore, 46
getString, 47
getStringAfter, 47
getStringBefore, 48
parseXmlString, 46
removeBlankEnds, 48
parseXmlString.cc, 335
parseXmlString.hh, 336
pass
StDbFastSqlWriter, 114–118

StDbXmlReader, 294–299
StDbXmlWriter, 303–309
typeAcceptor, 324, 325
passAux
StDbXmlReader.cc, 396
PassInElement
StDbTable, 251
PassOutElement
StDbTable, 253
PointMysqlDb
StDbServerImpl, 199
precision
StString, 315
prepareNode
StDbSql, 213
Print
StDbBuffer, 64
printCstructName
StDbTable, 255
printDataTable
StDbTable, 255
printDbName
StDataBaseI, 54
StDbManager, 130
StDbManagerImpl, 158
StDbNode, 186
printDbServerFile
StDbDefaults, 106
printDbServerVar
StDbDefaults, 106
printDomainName
StDataBaseI, 54
printElementName
StDbTable, 255
printFlavor
StDbDefaults, 107
StDbTable, 255
printHostName
StDbServer, 192
printIndexName
StDbElementIndex, 111
printInfo
StDbManager, 130, 131
printMessage
StDbMessenger, 168
StDbMessService, 170, 171
printName
StDbNode, 186
printNextIndex
StDbElementIndex, 111
printNoDataReturned
StDbTable, 255
printNodeType
StDbNode, 186
printNumberStats
StDbConfigNode, 88
StDbConfigNodeImpl, 99
printPword
StDbServer, 192
printQuery
MysqlDb, 38
printServerName
StDbServer, 192
printTables
StDbConfigNode, 89
StDbConfigNodeImpl, 99
printTimeStats
StDbManager, 131
StDbManagerImpl, 158
printTree
StDbConfigNode, 89
printTypeName
StDataBaseI, 54
printUnixSocket
StDbServer, 192
printUser
StDbServer, 193
printVersion
StDbDefaults, 107
StDbNode, 186
pWord
StDbManager, 131

QueryDb
StDataBaseI, 54
StDbSql, 214, 217
QueryDbFunction
StDataBaseI, 54
StDbSql, 218
QueryDbTimes
StDataBaseI, 54
StDbSql, 219
QueryDescriptor
StDataBaseI, 55
StDbSql, 221
queryNode
StDbSql, 222
QueryStatus
MysqlDb, 39

Rarray
StDbBuffer.cc, 342, 345
Raz
StDbBuffer, 64
RazQuery
MysqlDb, 39
ReadArray
StDbBuffer, 65, 66

StDbBufferI, 76, 77
readConfigNodeInfo
StDbSql, 222
ReadDataFromDB
StDbModifier, 173
ReadElement
StDbTable, 255
readNodeInfo
StDbSql, 223
ReadScalar
StDbBuffer, 66, 67
StDbBufferI, 78, 79
readTable
StDbXmlReader, 299
readTableInfo
StDbSql, 223
RecommendedServer
MysqlDb, 39
reConnect
MysqlDb, 39
Release
MysqlDb, 40
MysqlResult, 44
removeBlankEnds
parseXmlString, 48
removeTable
StDbConfigNode, 89
StDbConfigNodeImpl, 100
resetConfig
StDbConfigNode, 89
StDbConfigNodeImpl, 100
resetCounter
StDbElementIndex, 112
resetStoreInfo
StDbStoreInfo, 234
reSize
StDbTableDescriptor, 277
resizeElementID
StDbTable, 257
reSizeInternalStore
StDbStoreInfo, 234
resizeNumRows
StDbTable, 258
rollBack
StDataBaseI, 55
StDbManager, 131
StDbManagerImpl, 159
StDbSql, 224
rollBackAllNodes
StDbManager, 131
StDbManagerImpl, 160
rollBackAllTables
StDbManager, 131
StDbManagerImpl, 160
row
dbTable, 20
rowID
dbRow, 17
rowNumber
dbRow, 17
rowSizeTT
StDbTableDescriptor, 279
rowVec
dbStruct.hh, 328
Rscal
StDbBuffer.cc, 342, 345

selectElements
StDataBaseI, 55
StDbSql, 225
sendMess
StDbSql, 225
serverEnvVar
StDbDefs.hh, 355
ServerList
StDbManagerImpl.hh, 368
setBaseLine
StDbTable, 258
setBeginTime
StDbTable, 258
setBinary
StDbTable, 259
setBranchID
StDbConfigNode, 89
setChildNode
StDbConfigNode, 89
SetClientMode
StDbBuffer, 67
StDbBufferI, 79
setConfigured
StDbNode, 186
setCstructName
StDbTable, 259
setDataBase
StDataBaseI, 55
setDataTable
StDbTable, 259
SetDateTime
StDbModifier, 174
setDateTime
StDbTime, 287
setDbDomain
StDataBaseI, 56
StDbNode, 186
SetDbName
StDbModifier, 174
setDbName
StDataBaseI, 56

StDbNode, 186
setDbStoreType
StDataBaseI, 56
setDbType
StDataBaseI, 56
StDbNode, 187
setDbUtils
StDbSql, 226
SetDebug
StDbModifier, 174
setDefaultBeginTime
StDbSql, 226
setDefaultDb
MysqlDb, 40
setDefaultEndTime
StDbSql, 226
setDefaultFlavor
StDbTable, 259
setDefaultReturnValues
StDbSql, 226
setDescriptor
StDbTable, 259
setDomainName
StDataBaseI, 56
setElementID
StDbTable, 259
setElementIndexInfo
StDbConfigNode, 90
StDbConfigNodeImpl, 100
setElementName
StDbTable, 260
setEndKey
basic, 9
setEndStoreTime
StDbTable, 260
setEndTime
StDbTable, 260
setFirstChildNode
StDbConfigNode, 90
SetFlavor
StDbModifier, 174
setFlavor
StDbConfigNode, 90
StDbTable, 260
setHostName
StDbServer, 193
setIndexed
StDbTable, 261
SetInputFileName
StDbModifier, 174
setIsDbNode
StDbConfigNode, 90
setIsDefaultServer
StDbServer, 193
setMessenger
StDbManager, 132
setMessLevel
StDbMessService, 171
setName
StDbNode, 187
setNextNode
StDbConfigNode, 90
setNodeID
StDbNode, 187
setNodeInfo
StDbTable, 261
setNodeType
StDbNode, 187
SetOutputFileName
StDbModifier, 175
setParentNode
StDbConfigNode, 90
setPortNumber
StDbServer, 193
setProdTime
StDbConfigNode, 91
StDbTable, 261
setQuiet
StDbManager, 132
setRequestTime
StDbManager, 132
StDbManagerImpl, 161
setRowLimit
StDbTable, 261
setRowNumber
StDbTable, 261
setSchemaID
StDbTable, 261
StDbTableDescriptor, 277
setServerName
StDbServer, 193
setStartKey
basic, 9
SetStorageMode
StDbBuffer, 67
StDbBufferI, 79
setStoreTime
StDbManager, 132
StDbManagerImpl, 161
setStructID
StDbTableDescriptor, 277
SetTable
StDbTable, 262
SetTableName
StDbModifier, 175
setTablesFlavor
StDbConfigNode, 91
StDbConfigNodeImpl, 101

setTablesProdTime
StDbConfigNode, 91
StDbConfigNodeImpl, 101
SetTime
StDbModifier, 175
setTime
StDbTime, 287
setTimeLogging
StDbServer, 193
StDbServerImpl, 199
setTimeValues
StDbTable, 262
setTypeName
StDataBaseI, 57
setUnixSocket
StDbServer, 193
setUnixTime
StDbTime, 287
setUser
StDbManager, 132
StDbServer, 193
setVerbose
StDbManager, 133
setVersion
StDbNode, 187
SetVersionName
StDbModifier, 175
size
elem, 22
tableDescriptor, 322
starDefault
StDbDefs.hh, 355
start
StDbLogger, 122
startKey
basic, 10
Stchar
StTableDescriptorI.h, 402
StDataBaseI, 50
StDataBaseI, 51
StDataBaseI
~StDataBaseI, 51
checkDbDomain, 52
checkDbName, 52
checkDbType, 52
close, 52
getDateTime, 52
getDbDomain, 52
getDbName, 52
getDbStoreType, 53
getDbType, 53
getDomainName, 53
getTypeName, 53
getUnixTime, 53
insertNodeRelation, 53
mdbDomain, 58
mdbName, 58
mdbStore, 58
mdbType, 58
mdomainName, 58
mtypeName, 58
printDbName, 54
printDomainName, 54
printTypeName, 54
QueryDb, 54
QueryDbFunction, 54
QueryDbTimes, 54
QueryDescriptor, 55
rollBack, 55
selectElements, 55
setDataBase, 55
setDbDomain, 56
setDbName, 56
setDbStoreType, 56
setDbType, 56
setDomainName, 56
setTypeName, 57
StDataBaseI, 51
storeConfigNode, 57
storeTableNode, 57
use, 57
WriteDb, 57
StDataBaseI.cc, 337
StDataBaseI.hh, 338
stdb_streams.h, 339
StDbBuffer, 59
StDbBuffer, 61
StDbBuffer
~StDbBuffer, 61
AddField, 62
ChangeField, 62
Find_Col, 63
Init, 63
IsClientMode, 63
IsStorageMode, 63
mCol, 74
mCur, 74
MemSwapCpy, 63
mLast, 74
mMax, 74
mMode, 74
Print, 64
Raz, 64
ReadArray, 65, 66
ReadScalar, 66, 67
SetClientMode, 67
SetStorageMode, 67
StDbBuffer, 61

StrConv, 68, 69
WhatsIn, 69
WriteArray, 69, 70
WriteMem, 71, 72
WriteScalar, 72, 73
zeroColumn, 73
StDbBuffer.cc, 340
StDbBuffer.cc
castcase, 341
castcasest, 341
genwritemem, 341, 345
Rarray, 342, 345
Rscal, 342, 345
Warray, 342, 345
Wscal, 343, 345
StDbBuffer.h, 346
_ascii, 347
_char, 346
_double, 347
_float, 347
_int, 347
_long, 347
_longlong, 347
_short, 346
_string, 347
_uchar, 346
_uint, 347
_ulong, 347
_ushort, 347
Auto, 346
Client, 346
Storage, 346
StDbBuffer.h
BuffMode, 346
mycsize, 347
mycswapl, 347
myctype, 346
StDbBufferI, 75
StDbBufferI
~StDbBufferI, 76
IsClientMode, 76
IsStorageMode, 76
ReadArray, 76, 77
ReadScalar, 78, 79
SetClientMode, 79
SetStorageMode, 79
WriteArray, 79, 80
WriteScalar, 80, 81
StDbBufferI.h, 348
StDbConfigNode, 82
StDbConfigNode, 83
StDbConfigNode
~StDbConfigNode, 84
addChildren, 84
addDbTable, 84
addTable, 84
appendNode, 84
buildTree, 84
compareTables, 85
deleteChildren, 85
deleteTree, 85
findChildConfigNode, 85
findConfigNode, 85–87
findLocalTable, 87
findTable, 87
getBranchID, 87
getElementIndex, 87
getElementIndexInfo, 87
getFirstChildNode, 87
getNextNode, 87
getNumberStats, 88
getNumIndeces, 88
getParentNode, 88
getStDbTableIter, 88
hasChildren, 88
hasData, 88
isDbNode, 88
mbranchID, 91
mfirstChildNode, 91
mhasData, 91
misDbNode, 92
mnextNode, 92
mparentNode, 92
printNumberStats, 88
printTables, 89
printTree, 89
removeTable, 89
resetConfig, 89
setBranchID, 89
setChildNode, 89
setElementIndexInfo, 90
setFirstChildNode, 90
setFlavor, 90
setIsDbNode, 90
setNextNode, 90
setParentNode, 90
setProdTime, 91
setTablesFlavor, 91
setTablesProdTime, 91
StDbConfigNode, 83
zeroNodes, 91
StDbConfigNode.cc, 349
StDbConfigNode.cc
__CLASS__, 349
StDbConfigNode.hh, 350
StDbConfigNodeImpl, 93
StDbConfigNodeImpl, 94
StDbConfigNodeImpl

~StDbConfigNodeImpl, 94
addChildren, 95
addDbTable, 95
addTable, 95
buildTree, 96
compareTables, 96
deleteTables, 97
findLocalTable, 97
findTable, 97
getElementIndex, 98
getElementIndexInfo, 98
getNumberStats, 98
getNumIndeces, 98
getStDbTableIter, 99
melementIndex, 103
mindexRef, 103
mTables, 103
printNumberStats, 99
printTables, 99
removeTable, 100
resetConfig, 100
setElementIndexInfo, 100
setTablesFlavor, 101
setTablesProdTime, 101
StDbConfigNodeImpl, 94
updateDbInfo, 101
updateDbTable, 102
updateDbTables, 102
StDbConfigNodeImpl.cc, 351
StDbConfigNodeImpl.cc
__CLASS__, 351
__METHOD__, 351
StDbConfigNodeImpl.hh, 352
StDbConfigNodeImpl.hh
TableList, 352
StDbDefaults, 104
StDbDefaults, 104
StDbDefaults
~StDbDefaults, 105
getEndTime, 105
getFileName, 105
getProdTime, 105
getServerFileName, 106
Instance, 106
IsDefaultFlavor, 106
IsDefaultVersion, 106
mdbServerFile, 107
mdbServerVar, 107
mendTime, 107
mflavor, 107
mInstance, 107
mprodTime, 107
mversion, 107
printDbServerFile, 106
printDbServerVar, 106
printFlavor, 107
printVersion, 107
StDbDefaults, 104
StDbDefaults.cc, 353
StDbDefaults.hh, 354
StDbDefs.hh, 355
dbBbc, 356
dbCalibrations, 357
dbConditions, 356
dbConfigurations, 356
dbCtb, 355
dbDaq, 356
dbDEnd, 356
dbDomainUnknown, 355
dbDUser1, 356
dbDUser2, 356
dbDUser3, 356
dbEemc, 356
dbEmc, 355
dbFpd, 356
dbFtpc, 355
dbGeometry, 357
dbGlobal, 356
dbL3, 356
dbMwc, 356
dbOnl, 356
dbPmd, 356
dbRhic, 356
dbRich, 356
dbRts, 356
dbRunCatalog, 357
dbRunLog, 356
dbRunParams, 357
dbScaler, 356
dbScalers, 357
dbServer, 356
dbSsd, 356
dbStar, 355
dbStDb, 356
dbSvt, 355
dbTEnd, 357
dbTestScheme, 357
dbTof, 356
dbTpc, 355
dbTracker, 356
dbTrg, 355
dbTUser1, 357
dbTUser2, 357
dbTUser3, 357
dbV00, 356
dbZdc, 356
serverEnvVar, 355
starDefault, 355

userHome, 355
StDbDefs.hh
dbFindServerMode, 355
StDbDomain, 355
StDbStoreType, 356
StDbType, 356
StDbDomain
StDbDefs.hh, 355
StDbElementIndex, 109
StDbElementIndex, 109
StDbElementIndex
~StDbElementIndex, 109
addElementIndex, 110
addNameValuePair, 110
clearIndex, 110
getIndexName, 110
getIndexVal, 111
getNextIndex, 111
getNumIndeces, 111
mcurrent, 112
mnumIndeces, 112
mnvals, 112
printIndexName, 111
printNextIndex, 111
resetCounter, 112
StDbElementIndex, 109
StDbElementIndex.cc, 358
StDbElementIndex.hh, 359
StDbElementIndex.hh
N_MAX_INDEXVALS, 359
StDbFastSqlWriter, 113
StDbFastSqlWriter, 114
StDbFastSqlWriter
~StDbFastSqlWriter, 114
endAccessor, 114
ioTable, 114
os, 120
pass, 114–118
StDbFastSqlWriter, 114
streamAccessor, 119
streamEndRow, 119
streamEndTableName, 119
streamHeader, 119
streamRow, 119
streamTableName, 119
streamTail, 120
StDbFastSqlWriter.cc, 360
StDbFastSqlWriter.h, 361
StDbLogger, 121
StDbLogger, 121
StDbLogger
~StDbLogger, 121
end, 121
getNumCalls, 121
getTotalTimes, 122
mnTotal, 122
mt0, 122
mtotalTimes, 122
start, 122
StDbLogger, 121
wallTime, 122
StDbLogger.hh, 362
StDbManager, 123
StDbManager, 125
StDbManager
~StDbManager, 125
closeAllConnections, 125
closeConnection, 125
commitAllNodes, 126
commitAllTables, 126
fetchAllTables, 126
fetchDbTable, 126
findDb, 126
findDefaultServer, 126
findServer, 126, 127
getConfigNodeName, 127
getDataBaseInfo, 127
getDateCheckTime, 127
getDateRequestTime, 127
getDateStoreTime, 127
getDbDomain, 127
getDbDomainName, 128
getDbName, 128
getDbType, 128
getDbTypeName, 128
getEnvList, 128
getExternalVersion, 128
getMessenger, 128
getUnixCheckTime, 128
getUnixRequestTime, 128
getUnixStoreTime, 128
initConfig, 129
Instance, 129
IsQuiet, 129
IsValid, 130
IsVerbose, 130
Messenger, 134
mInstance, 134
misQuiet, 134
misTimeLogged, 134
misVerbose, 134
mpWord, 134
mstringDup, 130
muserName, 134
newDbTable, 130
printDbName, 130
printInfo, 130, 131
printTimeStats, 131

pWord, 131
rollBack, 131
rollBackAllNodes, 131
rollBackAllTables, 131
setMessenger, 132
setQuiet, 132
setRequestTime, 132
setStoreTime, 132
setUser, 132
setVerbose, 133
StDbManager, 125
storeAllTables, 133
storeConfig, 133
storeDbTable, 133
turnOffTimeLogging, 133
updateMessLevel, 133
userName, 133
StDbManager.cc, 363
StDbManager.hh, 364
StDbManagerImpl, 135
StDbManagerImpl, 137
StDbManagerImpl
~StDbManagerImpl, 138
addDbDomain, 138
addDbType, 138
closeAllConnections, 138, 139
closeConnection, 139
commitAllNodes, 139
commitAllTables, 139
dbDomainFree, 164
dbTypeFree, 164
deleteDomains, 140
deleteServers, 140
deleteTypes, 141
fetchAllTables, 141
fetchDbTable, 141, 142
findDb, 142
findDefaultServer, 143
findServer, 143, 144
findServerString, 144
findServersXml, 145
getConfigNodeName, 147
getDataBaseInfo, 147, 148
getDateCheckTime, 148
getDateRequestTime, 148
getDateStoreTime, 149
getDbDomain, 149
getDbDomainName, 149
getDbName, 150
getDbType, 150
getDbTypeName, 150
getEnvList, 151
getExternalVersion, 151
getNextName, 152
getUnixCheckTime, 152
getUnixRequestTime, 152
getUnixStoreTime, 152
initConfig, 152–154
initDomains, 154
initTypes, 155
IsValid, 155
lookUpServers, 155
mcheckTime, 164
mdataLog, 164
mDomains, 164
mfactory, 165
mhasDefaultServer, 165
mhasServerList, 165
mnodeLog, 165
mparser, 165
mservers, 165
mstoreTime, 165
mTypes, 165
newDbDomain, 156
newDbTable, 157
newDbType, 157
printDbName, 158
printTimeStats, 158
rollBack, 159
rollBackAllNodes, 160
rollBackAllTables, 160
setRequestTime, 161
setStoreTime, 161
StDbManagerImpl, 137
storeAllTables, 161
storeConfig, 162
storeDbTable, 163
turnOffTimeLogging, 163
updateDateTime, 164
updateUnixTime, 164
xmlInputSource, 165
xmlServerList, 165
StDbManagerImpl.cc, 365
StDbManagerImpl.cc
__CLASS__, 365
__METHOD__, 365, 366
environ, 366
StDbManagerImpl.hh, 367
StDbManagerImpl.hh
dbDomains, 367
dbTypes, 367
MAX_ENV_VARS, 367
ServerList, 368
StDbMessenger, 167
StDbMessenger, 167
StDbMessenger
~StDbMessenger, 167
mos, 169

printMessage, 168
StDbMessenger, 167
StDbMessenger.cc, 369
StDbMessenger.hh, 370
StDbMessLevel
StDbMessService.hh, 371
StDbMessService, 170
StDbMessService, 170
StDbMessService
~StDbMessService, 170
getMessLevel, 170
mdbLevel, 171
printMessage, 170, 171
setMessLevel, 171
StDbMessService, 170
StDbMessService.hh, 371
dbMConnect, 371
dbMDebug, 371
dbMErr, 371
dbMFatal, 371
dbMWarn, 371
StDbMessService.hh
StDbMessLevel, 371
StDbModifier, 172
StDbModifier, 172
StDbModifier
~StDbModifier, 172
fDbName, 177
fDebug, 177
fFlavorName, 177
fInputFileName, 177
fOutputFileName, 178
fTableName, 178
fTimestamp, 178
funixTime, 178
fVersionName, 178
ReadDataFromDB, 173
SetDateTime, 174
SetDbName, 174
SetDebug, 174
SetFlavor, 174
SetInputFileName, 174
SetOutputFileName, 175
SetTableName, 175
SetTime, 175
SetVersionName, 175
StDbModifier, 172
WriteDataToDB, 175
StDbModifier.cxx, 372
StDbModifier.cxx
ClassImp, 372
StDbModifier.h, 374
StDbNode, 179
StDbNode, 180
StDbNode
~StDbNode, 181
addWrittenNode, 181
canRollBack, 181
checkName, 181
checkNode, 181
checkVersion, 182
commit, 182
decodeElementID, 182
getDbDomain, 183
getDbName, 184
getDbType, 184
getMyName, 184
getName, 184
getNextID, 184
getNodeID, 184
getNodeType, 185
getVersion, 185
IsConfigured, 185
isNode, 185
IsTable, 185
mcanRollBack, 188
mdbDomain, 188
mdbName, 188
mdbType, 188
misConfigured, 188
mname, 188
mnodeID, 188
mnodeType, 188
mstrDup, 185
mversion, 188
printDbName, 186
printName, 186
printNodeType, 186
printVersion, 186
setConfigured, 186
setDbDomain, 186
setDbName, 186
setDbType, 187
setName, 187
setNodeID, 187
setNodeType, 187
setVersion, 187
StDbNode, 180
StDbNode.cc, 375
StDbNode.hh, 376
StDbServer, 189
StDbServer, 190
StDbServer
~StDbServer, 190
addDataBase, 191
closeConnection, 191
getConnectTimes, 191
getHostName, 191

getPortNumber, 191
getQueryTimes, 191
getServerName, 191
getSocketTimes, 191
getUnixSocket, 191
isConnected, 192
isDefault, 192
mhostName, 194
misDefault, 194
mportNumber, 194
mpword, 194
mserverName, 194
mstringDup, 192
munixSocket, 194
muserName, 194
printHostName, 192
printPword, 192
printServerName, 192
printUnixSocket, 192
printUser, 193
setHostName, 193
setIsDefaultServer, 193
setPortNumber, 193
setServerName, 193
setTimeLogging, 193
setUnixSocket, 193
setUser, 193
StDbServer, 190
useDb, 194
StDbServer.cc, 377
StDbServer.cc
__CLASS__, 377
StDbServer.hh, 378
StDbServerImpl, 196
StDbServerImpl, 196, 197
StDbServerImpl
~StDbServerImpl, 197
addDataBase, 197
buff, 200
closeConnection, 197
Db, 200
deleteDataBases, 198
getConnectTimes, 198
getQueryTimes, 198
getSocketTimes, 198
isConnected, 198
mcurrentDb, 200
mdataBases, 200
PointMysqlDb, 199
setTimeLogging, 199
StDbServerImpl, 196, 197
useDb, 199, 200
StDbServerImpl.cc, 379
StDbServerImpl.cc
__CLASS__, 379
StDbServerImpl.hh, 380
StDbServerImpl.hh
DbList, 380
StDbSql, 202
StDbSql, 203, 204
StDbSql
~StDbSql, 204
addDescriptor, 204
buff, 232
checkColumn, 204
checkForNull, 204
checkTableCatalog, 205
checkTablePrepForQuery, 205
checkValue, 205
clear, 206
close, 206
Db, 232
deleteDescriptor, 206
deleteDescriptors, 206
deleteRows, 206
findDescriptor, 207
getColumnList, 207
getDataTable, 207
getDataTables, 208
getDateTime, 208
getDescriptor, 209
getElementList, 209
getElementListIN, 210
getEmptyString, 210
getFlavorQuery, 210
getIndexNames, 211
getProdTimeQuery, 211
getUnixTime, 211
hasInstance, 212
init, 212
initEndTime, 212
insertNodeRelation, 212
insertNodeString, 213
mdefaultEndDateTime, 232
mdefaultEndTime, 232
mdescriptors, 232
mgr, 232
mRetString, 213
mretString, 232
mtableCatalog, 232
prepareNode, 213
QueryDb, 214, 217
QueryDbFunction, 218
QueryDbTimes, 219
QueryDescriptor, 221
queryNode, 222
readConfigNodeInfo, 222
readNodeInfo, 223

readTableInfo, 223
rollBack, 224
selectElements, 225
sendMess, 225
setDbUtils, 226
setDefaultBeginTime, 226
setDefaultEndTime, 226
setDefaultReturnValues, 226
StDbSql, 203, 204
storeConfigNode, 226
storeTableNode, 227
updateEndTime, 227
use, 228
WriteDb, 228, 229
WriteDbNoIndex, 230
StDbSql.cc, 381
StDbSql.cc
__CLASS__, 381
__METHOD__, 381, 382
DbQFailed, 382
DbQInfo, 382
DbQResult, 382
StDbSql.hh, 383
StDbSql.hh
DescList, 383
MAX_EID_INDECES, 383
StDbStoreInfo, 233
StDbStoreInfo, 233
StDbStoreInfo
~StDbStoreInfo, 233
addWrittenRow, 233
canRollBack, 234
commit, 234
getDataIDs, 234
hasData, 234
mcanRollBack, 235
mdataIDs, 235
mMaxRows, 235
mnumRows, 235
resetStoreInfo, 234
reSizeInternalStore, 234
StDbStoreInfo, 233
StDbStoreInfo.cc, 384
StDbStoreInfo.cc
N_ROWS_INCREMENT, 384
StDbStoreInfo.hh, 385
StDbStoreType
StDbDefs.hh, 356
StDbTable, 236
StDbTable, 239
StDbTable
~StDbTable, 239
addNElements, 240
addNRows, 240
AddRows, 240
addWrittenRows, 241
checkDescriptor, 241
clearStoreInfo, 241
Clone, 242
commitData, 242
createMemory, 242, 243
dbStreamer, 243
dbStreamerWrite, 244
dbTableStreamer, 244
defaultFlavor, 245
duplicateData, 245
getBeginDateTime, 245
getBeginTime, 246
getCstructName, 246
getDataTable, 246
getDataValue, 246
getDescriptor, 246
getDescriptorCpy, 247
getElementID, 247
getElementName, 247
getElementSpecs, 247
getEndDateTime, 247
getEndStoreDateTime, 248
getEndStoreTime, 248
getEndTime, 248
getFlavor, 248
getMaxTime, 248
GetNRows, 248
getProdTime, 248
getRowID, 249
getRowLimit, 249
getSchemaID, 249
GetTable, 249
GetTableCpy, 249
getTableSize, 249
getTimeValues, 250
getWrittenRows, 250
hasData, 250
hasDescriptor, 250
init, 250
IsBaseLine, 251
IsBinary, 251
IsIndexed, 251
IsTable, 251
mbeginTime, 265
mdata, 265
mdataTable, 265
mdefaultFlavor, 265
mdescriptor, 265
melementID, 265
melementName, 265
mendStoreTime, 265
mendTime, 265

mflavor, 265
mhasData, 265
mhasDescriptor, 266
mIsBaseLine, 266
mIsBinary, 266
mIsIndexed, 266
mprodTime, 266
mrowNumber, 266
mrows, 266
mrowsRequested, 266
mschemaID, 266
mstoredData, 266
mstructName, 266
mtimeVals, 267
PassInElement, 251
PassOutElement, 253
printCstructName, 255
printDataTable, 255
printElementName, 255
printFlavor, 255
printNoDataReturned, 255
ReadElement, 255
resizeElementID, 257
resizeNumRows, 258
setBaseLine, 258
setBeginTime, 258
setBinary, 259
setCstructName, 259
setDataTable, 259
setDefaultFlavor, 259
setDescriptor, 259
setElementID, 259
setElementName, 260
setEndStoreTime, 260
setEndTime, 260
setFlavor, 260
setIndexed, 261
setNodeInfo, 261
setProdTime, 261
setRowLimit, 261
setRowNumber, 261
setSchemaID, 261
SetTable, 262
setTimeValues, 262
StDbTable, 239
StreamAccessor, 262, 263
WriteElement, 263
StDbTable.cc, 386
StDbTable.cc
__CLASS__, 386
StDbTable.h, 387
StDbTableDescriptor, 268
StDbTableDescriptor, 269
StDbTableDescriptor
~StDbTableDescriptor, 270
endRowPadding, 270
fillElement, 270
fillLengths, 271
fillSizeAndOffset, 271
getCpy, 273
getCurrentInternalSize, 273
getElementDimensions, 273
getElementIndexLength, 273
getElementLength, 273
getElementName, 274
getElementNumDimensions, 274
getElementOffset, 274
getElementSize, 274
getElementType, 275
getNumElements, 275
getSchemaID, 275
getSize, 275
getStructID, 275
getTableDescriptor, 275
getTotalSizeInBytes, 276
getTrowSize, 276
getType, 276
init, 276
IsValid, 277
lastType, 278
mcols, 278
mCur, 278
mhasDouble, 278
misValid, 278
mMax, 278
mnumElements, 278
mschemaID, 278
mstructID, 278
mtableSize, 279
offsetToLast4Bytes, 279
offsetToNextEmptyByte, 279
padsize, 279
reSize, 277
rowSizeTT, 279
setSchemaID, 277
setStructID, 277
StDbTableDescriptor, 269
storeRowSize, 278
StDbTableDescriptor.cc, 388
StDbTableDescriptor.h, 389
StDbTableFactory, 280
StDbTableFactory
~StDbTableFactory, 280
newDbTable, 280
StDbTableFactory.cc, 390
StDbTableFactory.hh, 391
StDbTableIter, 281
StDbTableIter, 281

StDbTableIter
~StDbTableIter, 281
done, 281
next, 281
operator++, 281
StDbTableIter, 281
StDbTableIter.hh, 392
StDbTableIterImpl, 283
StDbTableIterImpl, 283
StDbTableIterImpl
~StDbTableIterImpl, 283
done, 284
init, 284
itr, 284
mnode, 284
next, 284
operator++, 284
StDbTableIterImpl, 283
StDbTableIterImpl.cc, 393
StDbTableIterImpl.hh, 394
StDbTableIterImpl.hh
ListIter, 394
StDbTime, 286
StDbTime, 286
StDbTime
~StDbTime, 287
getDateTime, 287
getUnixTime, 287
mdateTime, 288
munixTime, 288
setDateTime, 287
setTime, 287
setUnixTime, 287
StDbTime, 286
StDbTime.h, 395
StDbType
StDbDefs.hh, 356
StDbXmlReader, 289
StDbXmlReader, 290
StDbXmlReader
~StDbXmlReader, 290
buildDbTable, 290
buildStruct, 292
fillElements, 292
findElement, 294
loca, 300
maxlines, 300
pass, 294–299
readTable, 299
StDbXmlReader, 290
tab, 300
StDbXmlReader.cc, 396
StDbXmlReader.cc
passAux, 396
StDbXmlReader.h, 397
StDbXmlWriter, 301
StDbXmlWriter, 302
StDbXmlWriter
~StDbXmlWriter, 302
endAccessor, 302
ioTable, 302
os, 311
pass, 303–309
StDbXmlWriter, 302
streamAccessor, 309
streamEndRow, 309
streamEndTableName, 310
streamHeader, 310
streamRow, 310
streamTableName, 310
streamTail, 310
StDbXmlWriter.cc, 398
StDbXmlWriter.h, 399
Stdouble
StTableDescriptorI.h, 402
stendl
StString.h, 401
Stfloat
StTableDescriptorI.h, 402
Stint
StTableDescriptorI.h, 402
Stlong
StTableDescriptorI.h, 402
Stlonglong
StTableDescriptorI.h, 402
Storage
StDbBuffer.h, 346
storeAllTables
StDbManager, 133
StDbManagerImpl, 161
storeConfig
StDbManager, 133
StDbManagerImpl, 162
storeConfigNode
StDataBaseI, 57
StDbSql, 226
storeDbTable
StDbManager, 133
StDbManagerImpl, 163
storeRowSize
StDbTableDescriptor, 278
storeTableNode
StDataBaseI, 57
StDbSql, 227
str
StString, 315
StrConv
StDbBuffer, 68, 69

StreamAccessor
StDbTable, 262, 263
streamAccessor
StDbFastSqlWriter, 119
StDbXmlWriter, 309
tableAcceptor, 320
streamEndRow
StDbFastSqlWriter, 119
StDbXmlWriter, 309
tableAcceptor, 320
streamEndTableName
StDbFastSqlWriter, 119
StDbXmlWriter, 310
tableAcceptor, 321
streamHeader
StDbFastSqlWriter, 119
StDbXmlWriter, 310
tableAcceptor, 321
streamRow
StDbFastSqlWriter, 119
StDbXmlWriter, 310
tableAcceptor, 321
streamTableName
StDbFastSqlWriter, 119
StDbXmlWriter, 310
tableAcceptor, 321
streamTail
StDbFastSqlWriter, 120
StDbXmlWriter, 310
tableAcceptor, 321
Stshort
StTableDescriptorI.h, 402
stsize, 312
isize, 312
stsize, 312
StString, 313
StString, 313
StString
fPrec, 316
operator<<, 313–315
precision, 315
str, 315
StString, 313
StString.cc, 400
StString.h, 401
StString.h
stendl, 401
StTableDescriptorI, 317
StTableDescriptorI
~StTableDescriptorI, 317
getCpy, 317
getElementDimensions, 317
getElementIndexLength, 317
getElementLength, 318
getElementName, 318
getElementNumDimensions, 318
getElementOffset, 318
getElementSize, 318
getElementType, 318
getNumElements, 318
getTotalSizeInBytes, 318
getTrowSize, 318
StTableDescriptorI.h, 402
Stchar, 402
Stdouble, 402
Stfloat, 402
Stint, 402
Stlong, 402
Stlonglong, 402
Stshort, 402
Stuchar, 402
Stuint, 402
Stulong, 402
Stushort, 402
StTableDescriptorI.h
StTypeE, 402
StTypeSize, 402
StTypeE
StTableDescriptorI.h, 402
StTypeSize
StTableDescriptorI.h, 402
Stuchar
StTableDescriptorI.h, 402
Stuint
StTableDescriptorI.h, 402
Stulong
StTableDescriptorI.h, 402
Stushort
StTableDescriptorI.h, 402

tab
StDbXmlReader, 300
tableAcceptor, 320
tableAcceptor
~tableAcceptor, 320
endAccessor, 320
ioTable, 320
streamAccessor, 320
streamEndRow, 320
streamEndTableName, 321
streamHeader, 321
streamRow, 321
streamTableName, 321
streamTail, 321
tableAcceptor.hh, 403
tableDescriptor, 322
tableDescriptor
dimensionlen, 322

name, 322
offset, 322
size, 322
type, 322
TableList
StDbConfigNodeImpl.hh, 352
turnOffTimeLogging
StDbManager, 133
StDbManagerImpl, 163
type
column, 11
dbType, 21
elem, 22
tableDescriptor, 322
typeAcceptor, 323
typeAcceptor
~typeAcceptor, 323
pass, 324, 325
typeAcceptor.hh, 404

updateDateTime
StDbManagerImpl, 164
updateDbInfo
StDbConfigNodeImpl, 101
updateDbTable
StDbConfigNodeImpl, 102
updateDbTables
StDbConfigNodeImpl, 102
updateEndTime
StDbSql, 227
updateMessLevel
StDbManager, 133
updateUnixTime
StDbManagerImpl, 164
use
StDataBaseI, 57
StDbSql, 228
useDb
StDbServer, 194
StDbServerImpl, 199, 200
userHome
StDbDefs.hh, 355
userName
StDbManager, 133

val
column, 11
elem, 22

wallTime
StDbLogger, 122
Warray
StDbBuffer.cc, 342, 345
WhatsIn
StDbBuffer, 69
WriteArray
StDbBuffer, 69, 70
StDbBufferI, 79, 80
WriteDataToDB
StDbModifier, 175
WriteDb
StDataBaseI, 57
StDbSql, 228, 229
WriteDbNoIndex
StDbSql, 230
WriteElement
StDbTable, 263
WriteMem
StDbBuffer, 71, 72
WriteScalar
StDbBuffer, 72, 73
StDbBufferI, 80, 81
Wscal
StDbBuffer.cc, 343, 345

xmlInputSource
StDbManagerImpl, 165
xmlServerList
StDbManagerImpl, 165

zeroColumn
StDbBuffer, 73
zeroNodes
StDbConfigNode, 91