

SysToMath DB C++ Libraries Reference Manual
Version 1.04-r364

Generated by Doxygen 1.5.4-20071203

Sat Jan 5 11:25:33 2008

Contents

1	SysToMath DB C++ Libraries Main Page	1
2	SysToMath DB C++ Libraries Module Index	1
3	SysToMath DB C++ Libraries Directory Hierarchy	2
4	SysToMath DB C++ Libraries Namespace Index	2
5	SysToMath DB C++ Libraries Class Index	2
6	SysToMath DB C++ Libraries Class Index	4
7	SysToMath DB C++ Libraries File Index	5
8	SysToMath DB C++ Libraries Module Documentation	6
9	SysToMath DB C++ Libraries Directory Documentation	22
10	SysToMath DB C++ Libraries Namespace Documentation	25
11	SysToMath DB C++ Libraries Class Documentation	30
12	SysToMath DB C++ Libraries File Documentation	73
13	SysToMath DB C++ Libraries Example Documentation	93
14	SysToMath DB C++ Libraries Page Documentation	97

1 SysToMath DB C++ Libraries Main Page

1.1 Introduction

This documentation describes the C++ libraries contained in the SysToMath DB C++ Libraries package:

- [SysToMath DbMap C++ Library](#)
- [SysToMath DbSql C++ Library](#)

They depend on the

- **SysToMath Base C Library**
- **SysToMath Aids C++ Library**

belonging to the SysToMath C and C++ Libraries packages, respectively.

1.2 Supported Tool Families

The C++ libraries contained in the SysToMath DB C++ Libraries package are designed to support the tool family:

- [Microsoft Visual Studio Tool Family](#)

2 SysToMath DB C++ Libraries Module Index

2.1 SysToMath DB C++ Libraries Modules

Here is a list of all modules:

SysToMath DbMap C++ Library	6
DbMap: Named Persistent Associative Containers	6
Class template db::map	10
db::map comparison operators	11
db::map specialized algorithms	11
Class template db::set	12
db::set comparison operators	12
db::set specialized algorithms	13
DbMap Implementation	13
SysToMath DbMap C++ Library Test	21
SysToMath DbSql C++ Library	13
DbSql: SQL Interface of Named Persistent Associative Containers	14
DbSql Implementation	19

3 SysToMath DB C++ Libraries Directory Hierarchy

3.1 SysToMath DB C++ Libraries Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

stmdbmap	24
stm	23
impl	22
test	25

stmdbsql	24
stm	23
impl	22

4 SysToMath DB C++ Libraries Namespace Index

4.1 SysToMath DB C++ Libraries Namespace List

Here is a list of all documented namespaces with brief descriptions:

stm::db (Namespace db contains the associative container class templates db::map and db::set supporting unique keys and the associative container class templates db::multimap and db::multiset supporting equivalent keys)	25
--	-----------

5 SysToMath DB C++ Libraries Class Index

5.1 SysToMath DB C++ Libraries Class Hierarchy

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

stm::db::buffer	30
stm::db::local::xmap< ProxyT >::Item< U >	63
stm::db::sql::connection	31
stm::db::map< Key, T, Compare, Allocator, Policy, Traits >	34
stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::index< SKey, SCompare >	39
stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::locker	41
stm::db::multimap< Key, T, Compare, Allocator, Policy, Traits >	42
stm::db::multimap< Key, T, Compare, Allocator, Policy, Traits >::locker	44
stm::db::multiset< Key, Compare, Allocator, Policy, Traits >	45
stm::db::multiset< Key, Compare, Allocator, Policy, Traits >::locker	47
stm::db::null_type	47
stm::db::optional< T >	48
stm::db::policy< FlagsT, Flags >	48
stm::db::default_policy< Flags >	33
stm::db::policy< dword, Flags >	48
stm::db::local::pxmap< Key, T, Compare, Allocator, Policy, Traits, multi >	

stm::db::map < std::string, stm::db::sql::table::descr >	34
stm::db::set < Key, Compare, Allocator, Policy, Traits >	49
stm::db::set < Key, Compare, Allocator, Policy, Traits >::locker	54
stm::db::stream < T, Enable >	54
stm::db::stream < std::pair< T1, T2 > >	55
stm::db::sql::table	56
stm::db::sql::local::tableImpl < DefColSequence >::openIndex< indexId, IndexT >	59
stm::db::sql::local::tableImpl < DefColSequence >::openIndexes< indexCnt, Indexes, IndexFlags >	59
stm::db::sql::local::tableImpl < DefColSequence >::registerResultCreator< indexId, IndexT >	60
stm::db::sql::local::tableImpl < DefColSequence >::registerResultCreators< indexCnt, Indexes >	60
stm::db::traits < SizeT, DiffT, CharT, Allocator >	61
stm::db::default_traits < Allocator >	33
stm::db::traits < dword, int32, char, Allocator >	61
stm::db::local::xmap < ProxyT >::descr_type	61
stm::db::local::xmap < ProxyT >::secDb	68
stm::db::local::xmap < ProxyT >::xiter< BaseIt, DbAccessT >	69

6 SysToMath DB C++ Libraries Class Index

6.1 SysToMath DB C++ Libraries Class List

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

stm::db::buffer (The interface class buffer serves as a buffer abstraction for the conversion of application key and data objects into their byte serialized form needed to store them in Berkeley databases and vice versa)	30
stm::db::sql::connection (Class representing a SQL database connection)	31
stm::db::default_policy < Flags > (Default policy template parameter for class templates db::map , db::set , db::multimap and db::multiset)	33
stm::db::default_traits < Allocator > (Default traits template parameter for class templates db::map , db::set , db::multimap and db::multiset)	33

stm::db::map< Key, T, Compare, Allocator, Policy, Traits > (The db::map class template is a kind of associative container that supports unique keys (contains at most one of each key value) and provides for fast retrieval of values of another type T based on the keys)	34
stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::index< SKey, SCompare > (Member class template db::map::index)	39
stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::locker (Xxx)	41
stm::db::multimap< Key, T, Compare, Allocator, Policy, Traits > (Class template db::multimap)	42
stm::db::multimap< Key, T, Compare, Allocator, Policy, Traits >::locker (Member class db::multimap::locker)	44
stm::db::multiset< Key, Compare, Allocator, Policy, Traits > (Class template db::multiset)	45
stm::db::multiset< Key, Compare, Allocator, Policy, Traits >::locker (Member class db::multiset::locker)	47
stm::db::null_type (Empty type with default constructor doing nothing)	47
stm::db::optional< T > (The class template optional<T> is a boost::variant type consisting of a db::null_type member, if the instance holds no T value and a T member, if it holds a T value)	48
stm::db::policy< FlagsT, Flags > (Class template serving as Policy template parameter for class templates db::map , db::set , db::multimap and db::multiset)	48
stm::db::set< Key, Compare, Allocator, Policy, Traits > (The db::set class template is a kind of associative container that supports unique keys (contains at most one of each key value) and provides for fast retrieval based on these keys)	49
stm::db::set< Key, Compare, Allocator, Policy, Traits >::locker (Xxx)	54
stm::db::stream< T, Enable > (The class template stream<T> on the first hand serves as stream type to append values of type T in a byte serialized form to the buffers used in the access imlementation of Berkeley databases and on the other hand to extract values of type T from those buffers)	54
stm::db::stream< std::pair< T1, T2 > > (Partial specialization of class template stream<T, Enable> for std::pair)	55
stm::db::sql::table (Abstract base class representing a SQL database table)	56
stm::db::sql::local::tableImpl< DefColSequence >::openIndex< indexId, IndexT > (Helper functor for constructor which opens an index)	59
stm::db::sql::local::tableImpl< DefColSequence >::openIndexes< indexCnt, Indexes, IndexFlags > (Helper functor for constructor which opens indexes)	59
stm::db::sql::local::tableImpl< DefColSequence >::registerResultCreator< indexId, IndexT > (Helper functor for constructor which registers a result creation function)	60
stm::db::sql::local::tableImpl< DefColSequence >::registerResultCreators< indexCnt, Indexes > (Helper functor for constructor which registers result creation functions)	60

- [stm::db::traits< SizeT, DiffT, CharT, Allocator >](#) (Class template serving as Traits template parameter for class templates [db::map](#), [db::set](#), [db::multimap](#) and [db::multiset](#)) [61](#)
- [stm::db::local::xmap< ProxyT >::descr_type](#) (Type to describe the observable xmap values) [61](#)
- [stm::db::local::xmap< ProxyT >::Item< U >](#) (The class template [Item<U>](#) serves as proxy for key and data values of type [U](#) to realize the access to Berkeley databases) [63](#)
- [stm::db::local::xmap< ProxyT >::secDb](#) (Type of secondary database) [68](#)
- [stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >](#) (The class template [xiter<BaseIt, DbAccessT>](#) implements class template [iter<BaseIt, DbAccessT>](#), which in turn is the base of all [db::map](#), [db::map::index](#), [db::set](#), [db::multimap](#) and [db::multiset](#) iterators) [69](#)

7 SysToMath DB C++ Libraries File Index

7.1 SysToMath DB C++ Libraries File List

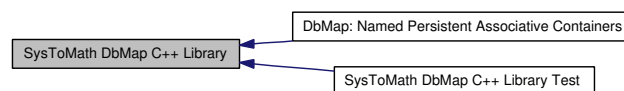
Here is a list of all documented files with brief descriptions:

- [dbmap.hpp](#) (Declarations for named persistent associative containers with unique and equivalent keys) [73](#)
- [dbmaptest.cpp](#) (Implementation of a console program demonstrating the usage and testing the implementation of the SysToMath DbMap C++ Library [stmdbmap](#)) [78](#)
- [dbsql.cpp](#) (Implementation of the SQL interface of named persistent associative containers) [80](#)
- [dbsql.hpp](#) (Declarations for the SQL interface of named persistent associative containers with unique and equivalent keys) [80](#)
- [dbxmap.hpp](#) (Implementation of named persistent associative containers) [82](#)
- [dbxsql.hpp](#) (Implementation of the SQL interface of named persistent associative containers) [88](#)

8 SysToMath DB C++ Libraries Module Documentation

8.1 SysToMath DbMap C++ Library

Collaboration diagram for SysToMath DbMap C++ Library:



8.1.1 Detailed Description

SysToMath DbMap C++ Library ([stmdbmap](#)).

The SysToMath DbMap C++ Library consists of several library objects providing various macros, functions, function templates, classes and class templates offering named persistent associative containers.

Modules

- [DbMap: Named Persistent Associative Containers](#)

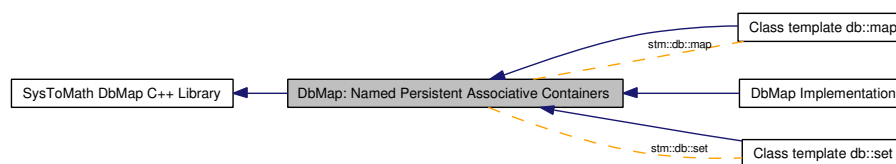
Named persistent associative containers with unique and equivalent keys.

- [SysToMath DbMap C++ Library Test](#)

Demonstration of the usage and tests of the implementation of the SysToMath DbMap C++ Library `stmdbmap`.

8.2 DbMap: Named Persistent Associative Containers

Collaboration diagram for DbMap: Named Persistent Associative Containers:



8.2.1 Detailed Description

Named persistent associative containers with unique and equivalent keys.

The declarations of these utilities are all contained in the namespace `stm::db`.

Files

- file [dbmap.hpp](#)

Declarations for named persistent associative containers with unique and equivalent keys.

Modules

- [Class template db::map](#)
- [Class template db::set](#)
- [DbMap Implementation](#)

Implementation of named persistent associative containers with unique and equivalent keys.

Classes

- struct [stm::db::policy< FlagsT, Flags >](#)

Class template serving as Policy template parameter for class templates `db::map`, `db::set`, `db::multimap` and `db::multiset`.

- struct `stm::db::default_policy< Flags >`
Default `policy` template parameter for class templates `db::map`, `db::set`, `db::multimap` and `db::multiset`.
- struct `stm::db::traits< SizeT, DiffT, CharT, Allocator >`
Class template serving as Traits template parameter for class templates `db::map`, `db::set`, `db::multimap` and `db::multiset`.
- struct `stm::db::default_traits< Allocator >`
Default `traits` template parameter for class templates `db::map`, `db::set`, `db::multimap` and `db::multiset`.
- struct `stm::db::null_type`
Empty type with default constructor doing nothing.
- struct `stm::db::optional< T >`
The class template `optional<T>` is a `boost::variant` type consisting of a `db::null_type` member, if the instance holds no `T` value and a `T` member, if it holds a `T` value.
- class `stm::db::buffer`
The interface class `buffer` serves as a `buffer` abstraction for the conversion of application key and data objects into their byte serialized form needed to store them in Berkeley databases and vice versa.
- struct `stm::db::stream< T, Enable >`
The class template `stream<T>` on the first hand serves as `stream` type to append values of type `T` in a byte serialized form to the buffers used in the access implementation of Berkeley databases and on the other hand to extract values of type `T` from those buffers.
- class `stm::db::map< Key, T, Compare, Allocator, Policy, Traits >`
The `db::map` class template is a kind of associative container that supports unique keys (contains at most one of each key value) and provides for fast retrieval of values of another type `T` based on the keys.
- class `stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::locker`
xxx
- class `stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::index< SKey, SCompare >`
Member class template `db::map::index`.
- class `stm::db::set< Key, Compare, Allocator, Policy, Traits >`
The `db::set` class template is a kind of associative container that supports unique keys (contains at most one of each key value) and provides for fast retrieval based on these keys.
- class `stm::db::set< Key, Compare, Allocator, Policy, Traits >::locker`
xxx
- class `stm::db::multimap< Key, T, Compare, Allocator, Policy, Traits >`
Class template `db::multimap`.
- class `stm::db::multimap< Key, T, Compare, Allocator, Policy, Traits >::locker`
Member class `db::multimap::locker`.

- class `stm::db::multiset< Key, Compare, Allocator, Policy, Traits >`
Class template `db::multiset`.
- class `stm::db::multiset< Key, Compare, Allocator, Policy, Traits >::locker`
Member class `db::multiset::locker`.

Typedefs

- typedef `dword` `stm::db::size_type`
For now the Berkeley database engine supports only 32 bit sizes.
- typedef `int32` `stm::db::difference_type`
For now the Berkeley database engine supports only 32 bit sizes.
- typedef char `stm::db::char_type`
For now only plain char characters are supported.

Functions

- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits, class SKey, class SCompare>
bool `stm::db::operator==` (const typename map< Key, T, Compare, Allocator, Policy, Traits >::template index< SKey, SCompare > &x, const typename map< Key, T, Compare, Allocator, Policy, Traits >::template index< SKey, SCompare > &y)
- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits, class SKey, class SCompare>
bool `stm::db::operator<` (const typename map< Key, T, Compare, Allocator, Policy, Traits >::template index< SKey, SCompare > &x, const typename map< Key, T, Compare, Allocator, Policy, Traits >::template index< SKey, SCompare > &y)
- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits, class SKey, class SCompare>
bool `stm::db::operator!=` (const typename map< Key, T, Compare, Allocator, Policy, Traits >::template index< SKey, SCompare > &x, const typename map< Key, T, Compare, Allocator, Policy, Traits >::template index< SKey, SCompare > &y)
- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits, class SKey, class SCompare>
bool `stm::db::operator>` (const typename map< Key, T, Compare, Allocator, Policy, Traits >::template index< SKey, SCompare > &x, const typename map< Key, T, Compare, Allocator, Policy, Traits >::template index< SKey, SCompare > &y)
- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits, class SKey, class SCompare>
bool `stm::db::operator>=` (const typename map< Key, T, Compare, Allocator, Policy, Traits >::template index< SKey, SCompare > &x, const typename map< Key, T, Compare, Allocator, Policy, Traits >::template index< SKey, SCompare > &y)
- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits, class SKey, class SCompare>
bool `stm::db::operator<=` (const typename map< Key, T, Compare, Allocator, Policy, Traits >::template index< SKey, SCompare > &x, const typename map< Key, T, Compare, Allocator, Policy, Traits >::template index< SKey, SCompare > &y)
- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits, class SKey, class SCompare>
void `stm::db::swap` (typename map< Key, T, Compare, Allocator, Policy, Traits >::template index< SKey, SCompare > &x, typename map< Key, T, Compare, Allocator, Policy, Traits >::template index< SKey, SCompare > &y)

- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator== (const multimap< Key, T, Compare, Allocator, Policy, Traits > &x,`
`const multimap< Key, T, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator< (const multimap< Key, T, Compare, Allocator, Policy, Traits > &x, const`
`multimap< Key, T, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator!= (const multimap< Key, T, Compare, Allocator, Policy, Traits > &x, const`
`multimap< Key, T, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator> (const multimap< Key, T, Compare, Allocator, Policy, Traits > &x, const`
`multimap< Key, T, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator>= (const multimap< Key, T, Compare, Allocator, Policy, Traits > &x,`
`const multimap< Key, T, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator<= (const multimap< Key, T, Compare, Allocator, Policy, Traits > &x,`
`const multimap< Key, T, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
`void stm::db::swap (multimap< Key, T, Compare, Allocator, Policy, Traits > &x, multimap< Key,`
`T, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator== (const multiset< Key, Compare, Allocator, Policy, Traits > &x, const`
`multiset< Key, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator< (const multiset< Key, Compare, Allocator, Policy, Traits > &x, const`
`multiset< Key, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator!= (const multiset< Key, Compare, Allocator, Policy, Traits > &x, const`
`multiset< Key, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator> (const multiset< Key, Compare, Allocator, Policy, Traits > &x, const`
`multiset< Key, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator>= (const multiset< Key, Compare, Allocator, Policy, Traits > &x, const`
`multiset< Key, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator<= (const multiset< Key, Compare, Allocator, Policy, Traits > &x, const`
`multiset< Key, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
`void stm::db::swap (multiset< Key, Compare, Allocator, Policy, Traits > &x, multiset< Key, Com-`
`pare, Allocator, Policy, Traits > &y)`

Variables

- `const size_type stm::db::locking = 1`
Policy flags (bitwise orable).

8.2.2 Typedef Documentation

8.2.2.1 `typedef dword stm::db::size_type`

For now the Berkeley database engine supports only 32 bit sizes.

Definition at line 288 of file `dbmap.hpp`.

8.2.2.2 `typedef int32 stm::db::difference_type`

For now the Berkeley database engine supports only 32 bit sizes.

Definition at line 293 of file `dbmap.hpp`.

8.2.2.3 `typedef char stm::db::char_type`

For now only plain char characters are supported.

Definition at line 298 of file `dbmap.hpp`.

8.2.3 Variable Documentation

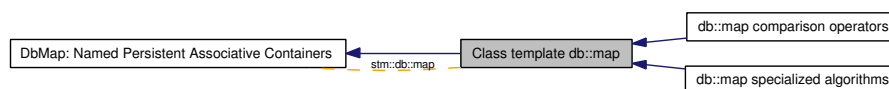
8.2.3.1 `const size_type stm::db::locking = 1`

Policy flags (bitwise orable).

Definition at line 303 of file `dbmap.hpp`.

8.3 Class template `db::map`

Collaboration diagram for Class template `db::map`:



Modules

- [db::map comparison operators](#)
- [db::map specialized algorithms](#)

Classes

- class `stm::db::map< Key, T, Compare, Allocator, Policy, Traits >`

The `db::map` class template is a kind of associative container that supports unique keys (contains at most one of each key value) and provides for fast retrieval of values of another type `T` based on the keys.

8.4 db::map comparison operators

Collaboration diagram for db::map comparison operators:



Functions

- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator== (const map< Key, T, Compare, Allocator, Policy, Traits > &x, const map< Key, T, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator< (const map< Key, T, Compare, Allocator, Policy, Traits > &x, const map< Key, T, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator!= (const map< Key, T, Compare, Allocator, Policy, Traits > &x, const map< Key, T, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator> (const map< Key, T, Compare, Allocator, Policy, Traits > &x, const map< Key, T, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator>= (const map< Key, T, Compare, Allocator, Policy, Traits > &x, const map< Key, T, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator<= (const map< Key, T, Compare, Allocator, Policy, Traits > &x, const map< Key, T, Compare, Allocator, Policy, Traits > &y)`

8.5 db::map specialized algorithms

Collaboration diagram for db::map specialized algorithms:

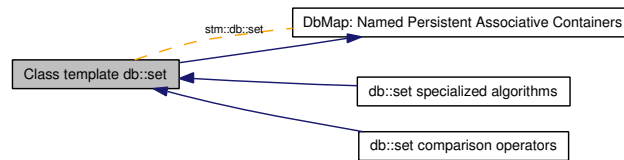


Functions

- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
`void stm::db::swap (map< Key, T, Compare, Allocator, Policy, Traits > &x, map< Key, T, Compare, Allocator, Policy, Traits > &y)`

8.6 Class template db::set

Collaboration diagram for Class template db::set:



Modules

- [db::set comparison operators](#)
- [db::set specialized algorithms](#)

Classes

- class [stm::db::set< Key, Compare, Allocator, Policy, Traits >](#)

The `db::set` class template is a kind of associative container that supports unique keys (contains at most one of each key value) and provides for fast retrieval based on these keys.

8.7 db::set comparison operators

Collaboration diagram for db::set comparison operators:



Functions

- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator== (const set< Key, Compare, Allocator, Policy, Traits > &x, const set< Key, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator< (const set< Key, Compare, Allocator, Policy, Traits > &x, const set< Key, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator!= (const set< Key, Compare, Allocator, Policy, Traits > &x, const set< Key, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator> (const set< Key, Compare, Allocator, Policy, Traits > &x, const set< Key, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator>= (const set< Key, Compare, Allocator, Policy, Traits > &x, const set< Key, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator<= (const set< Key, Compare, Allocator, Policy, Traits > &x, const set< Key, Compare, Allocator, Policy, Traits > &y)`

8.8 db::set specialized algorithms

Collaboration diagram for db::set specialized algorithms:

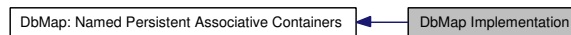


Functions

- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
`void stm::db::swap (set< Key, Compare, Allocator, Policy, Traits > &x, set< Key, Compare, Allocator, Policy, Traits > &y)`

8.9 DbMap Implementation

Collaboration diagram for DbMap Implementation:



8.9.1 Detailed Description

Implementation of named persistent associative containers with unique and equivalent keys.

Files

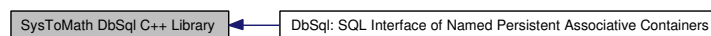
- file [dbxmap.hpp](#)
Implementation of named persistent associative containers.

Classes

- class `stm::db::local::xmap< ProxyT >`
- class `stm::db::local::pxmap< Key, T, Compare, Allocator, Policy, Traits, multi >`
- class `stm::db::local::plocker< Key, T, Compare, Allocator, Policy, Traits, multi, active >`
- class `stm::db::local::plocker< Key, T, Compare, Allocator, Policy, Traits, multi, boost::mpl::true_ >`
- class `stm::db::local::pindex< Key, T, Compare, Allocator, Policy, Traits, SKey, SCompare >`

8.10 SysToMath DbSql C++ Library

Collaboration diagram for SysToMath DbSql C++ Library:



8.10.1 Detailed Description

SysToMath DbSql C++ Library (stmdbsql).

The SysToMath DbSql C++ Library consists of several library objects providing various macros, functions, function templates, classes and class templates offering a rudimentary SQL interface of named persistent associative containers.

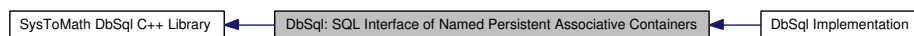
Modules

- [DbSql: SQL Interface of Named Persistent Associative Containers](#)

Rudimentary SQL interface of named persistent associative containers with unique and equivalent keys.

8.11 DbSql: SQL Interface of Named Persistent Associative Containers

Collaboration diagram for DbSql: SQL Interface of Named Persistent Associative Containers:



8.11.1 Detailed Description

Rudimentary SQL interface of named persistent associative containers with unique and equivalent keys.

The declarations of these utilities are all contained in the namespace `stm::db::sql`.

Files

- file [dbsql.hpp](#)

Declarations for the SQL interface of named persistent associative containers with unique and equivalent keys.

Modules

- [DbSql Implementation](#)

Implementation of the rudimentary SQL interface of named persistent associative containers with unique and equivalent keys.

Classes

- struct `stm::db::sql::noconversion< T >`
- class [stm::db::sql::table](#)
Abstract base class representing a SQL database [table](#).
- class `stm::db::sql::table::field`
- struct `stm::db::sql::table::average< T >`
- struct `stm::db::sql::table::average< db::null_type >`

- struct `stm::db::sql::table::average`< `bool` >
- struct `stm::db::sql::table::average`< `std::string` >
- struct `stm::db::sql::table::average`< `std::vector`< `byte` > >
- struct `stm::db::sql::table::defcol`< `T`, `Flags`, `Convert` >
- struct `stm::db::sql::table::types`< `DefColSequence` >
- class `stm::db::sql::table::column`
- class `stm::db::sql::table::layout`
- class `stm::db::sql::table::descr`
- class `stm::db::sql::table::descmap`
- class `stm::db::sql::table::locker`
- class `stm::db::sql::result`
- class `stm::db::sql::connection`

Class representing a SQL database connection.

Functions

- `const std::string & stm::db::sql::connection::path () const`
Returns the database connection path name..
- `bool stm::db::sql::connection::existsTable (const std::string &name) const throw ()`
Returns true, if the table name exists for this connection, else false.
- `void stm::db::sql::connection::createTable (const std::string &name, const std::string &type, const table::layout &columns) throw (std::runtime_error)`
If a table type is defined and a table name does not already exist, create a table named name of type type with layout columns for this connection, else throw.
- `template<class InputIterator> void stm::db::sql::connection::createTable (const std::string &name, const std::string &type, InputIterator firstColumn, InputIterator lastColumn) throw (std::runtime_error)`
*If a table type is defined and a table name does not already exist, create a table named name of type type with layout determined by the range [*firstColumn, *lastColumns) for this connection, else throw.*
- `bool stm::db::sql::connection::deleteTable (const std::string &name, bool force=false)`
If a table named name exists for this connection, delete it, if it is not active or force is true (default: false).
- `bool stm::db::sql::connection::activateTable (const std::string &name, size_type cacheSizeKb=0) throw (std::runtime_error)`
If a table named name exists for this connection, activate it, if it is not already active.
- `bool stm::db::sql::connection::deactivateTable (const std::string &name)`
If a table named name exists for this connection, deactivate it, if it is active.
- `table & stm::db::sql::connection::getTable (const std::string &name) const throw (std::runtime_error)`
- `const std::string & stm::db::sql::connection::getTableType (const std::string &name) const throw (std::runtime_error)`
- `const table::descmap & stm::db::sql::connection::tables () const`
Returns a constant reference to the table map which describes all tables of this connection.

- `table::descr * stm::db::sql::connection::getTableDescr (const std::string &name)`
Returns the address of the `table` descriptor of `table` name, if it exists, else the `NULL` pointer.
- `result * stm::db::sql::connection::execsql (const std::string &cmd) throw (std::runtime_error)`
Interpretes the SQL command `cmd` and returns a pointer to an according result object.
- `int stm::db::sql::connection::errorno () const`
Returns the numerical error status of the last SQL command execution.
- `const std::string & stm::db::sql::connection::errormsg () const`
Returns the textual error status of the last SQL command execution.

Variables

- `const size_type stm::db::sql::delayedsort = local::DelayedSortFlag`
Flag value for class template `table::defcol` indicating a sortable column whose index shall be constructed delayed, if applicable.
- `const size_type stm::db::sql::recnosort = local::DelayedSortFlag | local::RecnoFlag`
Flag value for class template `table::defcol` indicating a sortable column whose index shall be constructed delayed and support fast record based positioning, if applicable.
- `const size_type stm::db::sql::notsortable = local::NotSortableValue`
Flag value for class template `table::defcol` overwrites all other flags and indicates a not sortable column.

8.11.2 Function Documentation

8.11.2.1 `const std::string & stm::db::sql::connection::path () const` [inline, inherited]

Returns the database `connection path` name..

Definition at line 2471 of file `dbxsql.hpp`.

8.11.2.2 `bool stm::db::sql::connection::existsTable (const std::string & name) const throw ()` [inherited]

Returns true, if the `table` name exists for this `connection`, else false.

Definition at line 323 of file `dbsql.cpp`.

Referenced by `stm::db::sql::connection::getTableDescr()`.

8.11.2.3 `void stm::db::sql::connection::createTable (const std::string & name, const std::string & type, const table::layout & columns) throw (std::runtime_error)` [inherited]

If a `table` type is defined and a `table` name does not already exist, create a `table` named name of type type with layout columns for this `connection`, else throw.

Creation of a `table` does not mean the creation or opening of any database or index. This is done by `activateTable()`.

Definition at line 336 of file dbsql.cpp.

References StmDebugAidsErrmsg.

Referenced by stm::db::sql::connection::createTable().

8.11.2.4 `template<class InputIterator> void stm::db::sql::connection::createTable (const std::string & name, const std::string & type, InputIterator firstColumn, InputIterator lastColumn) throw (std::runtime_error) [inline, inherited]`

If a `table` type is defined and a `table` name does not already exist, create a `table` named name of type type with layout determined by the range [`*firstColumn`, `*lastColumns`) for this `connection`, else throw.

Creation of a `table` does not mean the creation or opening of any database or index. This is done by `activateTable()`.

Definition at line 2475 of file dbxsql.hpp.

References stm::db::sql::connection::createTable().

8.11.2.5 `bool stm::db::sql::connection::deleteTable (const std::string & name, bool force = false) [inherited]`

If a `table` named name exists for this `connection`, delete it, if it is not active or force is true (default: false).

If force is true and the `table` is active, it is deactivated before deletion. Deletion of a `table` does not mean the physical deletion of any database or index. This has to be done by external means. Return true on success.

Definition at line 377 of file dbsql.cpp.

References stm::db::sql::connection::deactivateTable().

8.11.2.6 `bool stm::db::sql::connection::activateTable (const std::string & name, size_type cacheSizeKb = 0) throw (std::runtime_error) [inherited]`

If a `table` named name exists for this `connection`, activate it, if it is not already active.

Throw, if the `table` does not exist, or if its activation failed. Activation of a `table` means opening the corresponding database and its indexes, creating them, if they do not already exist. return true, if the `table` was activated, false, if it already had been activated.

Definition at line 387 of file dbsql.cpp.

References StmDebugAidsErrmsg.

Referenced by stm::db::sql::connection::execsql().

8.11.2.7 `bool stm::db::sql::connection::deactivateTable (const std::string & name) [inherited]`

If a `table` named name exists for this `connection`, deactivate it, if it is active.

Deactivation of a `table` means closing of the corresponding database and its indexes. Return true on success.

Definition at line 453 of file dbsql.cpp.

Referenced by stm::db::sql::connection::deleteTable().

8.11.2.8 `const table::descmap & stm::db::sql::connection::tables () const` [inline, inherited]

Returns a constant reference to the [table map](#) which describes all tables of this [connection](#).

Definition at line 2486 of file `dbxsql.hpp`.

8.11.2.9 `table::descr * stm::db::sql::connection::getTableDescr (const std::string & name)` [inherited]

Returns the address of the [table](#) descriptor of [table](#) name, if it exists, else the NULL pointer.

Definition at line 329 of file `dbsql.cpp`.

References `stm::db::sql::connection::existsTable()`.

8.11.2.10 `result * stm::db::sql::connection::execsql (const std::string & cmd) throw (std::runtime_error)` [inherited]

Interpretes the SQL command `cmd` and returns a pointer to an according result object.

On return the error status has been updated.

Definition at line 531 of file `dbsql.cpp`.

References `stm::db::sql::connection::activateTable()`, `stm::db::sql::table::name()`, and `StmDebugAidsErrMsg`.

8.11.2.11 `int stm::db::sql::connection::errno () const` [inline, inherited]

Returns the numerical error status of the last SQL command execution.

Definition at line 2488 of file `dbxsql.hpp`.

8.11.2.12 `const std::string & stm::db::sql::connection::errmsg () const` [inline, inherited]

Returns the textual error status of the last SQL command execution.

Definition at line 2490 of file `dbxsql.hpp`.

8.11.3 Variable Documentation

8.11.3.1 `const size_type stm::db::sql::delayedsort = local::DelayedSortFlag`

Flag value for class template `table::defcol` indicating a sortable column whose index shall be constructed delayed, if applicable.

Definition at line 160 of file `dbsql.hpp`.

8.11.3.2 `const size_type stm::db::sql::recnosort = local::DelayedSortFlag | local::RecnoFlag`

Flag value for class template `table::defcol` indicating a sortable column whose index shall be constructed delayed and support fast record based positioning, if applicable.

Definition at line 167 of file `dbsql.hpp`.

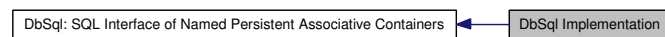
8.11.3.3 `const size_type stm::db::sql::notsortable = local::NotSortableValue`

Flag value for class template `table::defcol` overwrites all other flags and indicates a not sortable column.

Definition at line 173 of file `dbsql.hpp`.

8.12 DbSql Implementation

Collaboration diagram for DbSql Implementation:



8.12.1 Detailed Description

Implementation of the rudimentary SQL interface of named persistent associative containers with unique and equivalent keys.

Files

- file [dbxsql.hpp](#)
Implementation of the SQL interface of named persistent associative containers.
- file [dbsql.cpp](#)
Implementation of the SQL interface of named persistent associative containers.

Classes

- class `stm::db::sql::local::xtable`
- class `stm::db::sql::local::xtypes< DefColSequence >`
- class `stm::db::sql::local::xcolumn`
- class `stm::db::sql::local::xdescr`
- class `stm::db::sql::local::xlocker`
- class `stm::db::sql::local::xconnection`
- class `stm::db::sql::local::tableImpl< DefColSequence >`
- struct `stm::db::sql::local::updateResult< ValueT, col >`
- struct `stm::db::sql::local::updateResult< ValueT, 0 >`
- class `stm::db::sql::local::resultImpl< TableImplT, sortColNr, descending >`
- struct `stm::db::sql::select::type`
- struct `stm::db::sql::select::push_back_impl`
- struct `stm::db::sql::select::env`
- struct `stm::db::sql::select::syntax`

Enumerations

- enum {
`FlagBits = sizeof(size_type) * CHAR_BIT,`
`MSFlagBit = 1 << (FlagBits - 1),`

```

NotSortableValue = ~MSFlagBit,
DelayedSortFlag = MSFlagBit >> 1,
RecnoFlag = DelayedSortFlag >> 1 }

```

Functions

- `template<typename Skeys>`
`static void stm::db::sql::local::checkSecondaryKeys (table::layout::const_iterator cit)`
Helper for [table](#) layout consistency check.
- `template<>`
`static void stm::db::sql::local::checkSecondaryKeys< boost::tuples::null_type >`
`(table::layout::const_iterator)`

Variables

- `std::map< std::string, boost::function< table *(const std::string &, table::layout &, db::size_type)>> stm::db::sql::local::tableTypes`
- `const phoenix::function< type::setmember_impl< std::string,&type::tablename >> stm::db::sql::select::setTablename = type::setmember_impl<std::string, &type::tablename> ()`
- `const phoenix::function< type::setmember_impl< std::string,&type::indexname >> stm::db::sql::select::setIndexname = type::setmember_impl<std::string, &type::indexname> ()`
- `const phoenix::function< type::setmember_impl< bool,&type::descending >> stm::db::sql::select::setDescending = type::setmember_impl<bool, &type::descending> ()`
- `const phoenix::function< type::setmember_impl< bool,&type::all >> stm::db::sql::select::setAll = type::setmember_impl<bool, &type::all> ()`
- `const phoenix::function< type::getmember_impl< std::string,&type::tablename >> stm::db::sql::select::getTablename = type::getmember_impl<std::string, &type::tablename> ()`
- `const phoenix::function< type::getmember_impl< std::vector< std::string >,&type::columns >> stm::db::sql::select::getColumns = type::getmember_impl<std::vector<std::string>, &type::columns> ()`
- `const phoenix::function< push_back_impl > stm::db::sql::select::push_back = push_back_impl ()`

8.12.2 Function Documentation

8.12.2.1 `template<typename Skeys> static void stm::db::sql::local::checkSecondaryKeys (table::layout::const_iterator cit) [inline, static]`

Helper for [table](#) layout consistency check.

Definition at line 687 of file `dbxsql.hpp`.

References `get()`, and `StmDebugAidsErrMsg`.

8.13 SysToMath DbMap C++ Library Test

Collaboration diagram for SysToMath DbMap C++ Library Test:



8.13.1 Detailed Description

Demonstration of the usage and tests of the implementation of the SysToMath DbMap C++ Library `stmdbmap`.

Files

- file [dbmaptest.cpp](#)

Implementation of a console program demonstrating the usage and testing the implementation of the SysToMath DbMap C++ Library `stmdbmap`.

Namespaces

- namespace `stm`

Classes

- struct `Data`

Defines

- `#define _MAX_PATH 1024`

Typedefs

- typedef `stm::db::map< unsigned int, Data >` `DataMap`
- typedef `DataMap::index< std::string >` `CompIndex`
- typedef `DataMap::index< unsigned int >` `SalIndex`

Functions

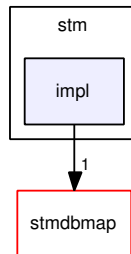
- static void `test ()`
- static bool `getCompany (const DataMap::value_type &value, CompIndex::skey_type &skey)`
- static bool `getSalary (const DataMap::value_type &value, SalIndex::skey_type &skey)`
- int `main (int argc, char *argv[])`

Variables

- const char * `Copyright` = "(C) 2003-2008 Tom Michaelis, SysToMath"
- const char * `Version` = "1.04-r364"

9 SysToMath DB C++ Libraries Directory Documentation

9.1 stmdbsql/stm/impl/ Directory Reference

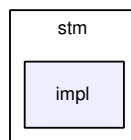


Files

- file [dbxsql.hpp](#)

Implementation of the SQL interface of named persistent associative containers.

9.2 stmdbmap/stm/impl/ Directory Reference

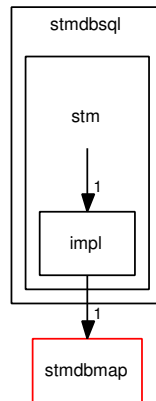


Files

- file [dbxmap.hpp](#)

Implementation of named persistent associative containers.

9.3 stndbmap/stm/ Directory Reference



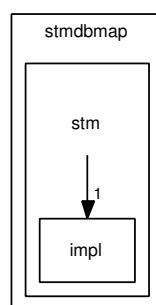
Directories

- directory [impl](#)

Files

- file [dbsql.cpp](#)
Implementation of the SQL interface of named persistent associative containers.
- file [dbsql.hpp](#)
Declarations for the SQL interface of named persistent associative containers with unique and equivalent keys.

9.4 stndbmap/stm/ Directory Reference



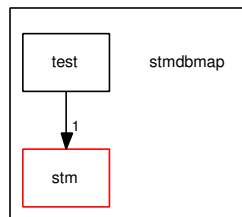
Directories

- directory [impl](#)

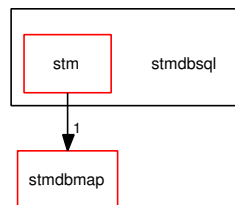
Files

- file [dbmap.hpp](#)

Declarations for named persistent associative containers with unique and equivalent keys.

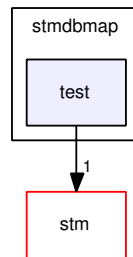
9.5 stmdbmap/ Directory Reference**Directories**

- directory [stm](#)
- directory [test](#)

9.6 stmdbsql/ Directory Reference**Directories**

- directory [stm](#)

9.7 stmdbmap/test/ Directory Reference



Files

- file [dbmaptest.cpp](#)

Implementation of a console program demonstrating the usage and testing the implementation of the SysToMath DbMap C++ Library stmdbmap.

10 SysToMath DB C++ Libraries Namespace Documentation

10.1 stm::db Namespace Reference

10.1.1 Detailed Description

Namespace [db](#) contains the associative container class templates [db::map](#) and [db::set](#) supporting unique keys and the associative container class templates [db::multimap](#) and [db::multiset](#) supporting equivalent keys.

They are modelled after the containers [std::map](#), [std::set](#), [std::multimap](#) and [std::multiset](#) as defined in the international standard C++-ISO/IEC 14882:2003(E). Citations of this international standard used in this header file are parenthesized expressions with the same syntax and semantics as inside the international standard itself. The class templates [db::map](#), [db::set](#), [db::multimap](#) and [db::multiset](#) meet all associative container requirements (23.1.2) and additionally permit named persistent storage of their contents as their implementation is based on the Berkeley database engine. So by means of [db::map](#), [db::set](#), [db::multimap](#) and [db::multiset](#) iterators which meet all bidirectional iterator requirements (24.1.4), arbitrary big containers are effectively manageable. Moreover, the class template [db::map](#) provides a member class template [index](#) permitting quick sorted database access according to user definable secondary keys and sort criteria.

Classes

- struct [policy](#)

Class template serving as Policy template parameter for class templates [db::map](#), [db::set](#), [db::multimap](#) and [db::multiset](#).

- struct [default_policy](#)

Default [policy](#) template parameter for class templates [db::map](#), [db::set](#), [db::multimap](#) and [db::multiset](#).

- struct [traits](#)

Class template serving as Traits template parameter for class templates `db::map`, `db::set`, `db::multimap` and `db::multiset`.

- struct `default_traits`

Default `traits` template parameter for class templates `db::map`, `db::set`, `db::multimap` and `db::multiset`.

- struct `null_type`

Empty type with default constructor doing nothing.

- struct `optional`

The class template `optional<T>` is a `boost::variant` type consisting of a `db::null_type` member, if the instance holds no `T` value and a `T` member, if it holds a `T` value.

- class `buffer`

The interface class `buffer` serves as a `buffer` abstraction for the conversion of application key and data objects into their byte serialized form needed to store them in Berkeley databases and vice versa.

- struct `stream`

The class template `stream<T>` on the first hand serves as `stream` type to append values of type `T` in a byte serialized form to the buffers used in the access implementation of Berkeley databases and on the other hand to extract values of type `T` from those buffers.

- class `map`

The `db::map` class template is a kind of associative container that supports unique keys (contains at most one of each key value) and provides for fast retrieval of values of another type `T` based on the keys.

- class `set`

The `db::set` class template is a kind of associative container that supports unique keys (contains at most one of each key value) and provides for fast retrieval based on these keys.

- class `multimap`

Class template `db::multimap`.

- class `multiset`

Class template `db::multiset`.

- struct `is_variant`

- struct `is_variant< boost::variant< BOOST_VARIANT_ENUM_PARAMS(T)> >`

- struct `is_optional`

- struct `is_optional< optional< T > >`

- struct `is_bytelike`

- struct `may_endianvt`

- struct `may_endianvt< T, typename boost::enable_if< boost::mpl::and_< boost::is_arithmetic< T >, boost::mpl::not_< boost::mpl::bool_< is_bytelike< T >::value > > >::type >`

- struct `is_conslist`

- struct `is_conslist< boost::tuples::null_type >`

- struct `is_conslist< boost::tuples::cons< HT, TT > >`

- struct `is_conslist< conslist< Sequence > >`

- struct `is_conslist< T, typename boost::enable_if< boost::is_same< boost::tuples::cons< typename T::head_type, typename T::tail_type >, typename T::inherited > >::type >`

- struct `conslist`< Sequence, typename boost::enable_if< boost::mpl::and_< boost::mpl::is_sequence< Sequence >, boost::mpl::empty< Sequence > > >::type >
- struct `conslist`< Sequence, typename boost::enable_if< boost::mpl::and_< boost::mpl::is_sequence< Sequence >, boost::mpl::not_< boost::mpl::empty< Sequence > > > >::type >
- struct `stream`< T, typename boost::enable_if< boost::is_empty< T > >::type >
- struct `stream`< T, typename boost::enable_if< boost::is_pod< T > >::type >
- struct `stream`< std::vector< T, Allocator > >
- struct `stream`< std::basic_string< charT, traits, Allocator > >
- struct `stream`< T, typename boost::enable_if< is_variant< T > >::type >
- struct `stream`< T, typename boost::enable_if< is_optional< T > >::type >
- struct `stream`< T, typename boost::enable_if< is_conslist< T > >::type >
- struct `stream`< std::pair< T1, T2 > >

Partial specialization of class template `stream`<T, Enable> for `std::pair`.

- struct `stream`< db::null_type >
- struct `stream`< sql::table::column >
- struct `stream`< sql::table::layout >
- struct `stream`< sql::table::descr >
- struct `stream`< Data >

Typedefs

- typedef `dword` `size_type`
For now the Berkeley database engine supports only 32 bit sizes.
- typedef `int32` `difference_type`
For now the Berkeley database engine supports only 32 bit sizes.
- typedef char `char_type`
For now only plain char characters are supported.

Functions

- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>
bool `operator==` (const `map`< Key, T, Compare, Allocator, Policy, Traits > &x, const `map`< Key, T, Compare, Allocator, Policy, Traits > &y)
- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>
bool `operator<` (const `map`< Key, T, Compare, Allocator, Policy, Traits > &x, const `map`< Key, T, Compare, Allocator, Policy, Traits > &y)
- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>
bool `operator!=` (const `map`< Key, T, Compare, Allocator, Policy, Traits > &x, const `map`< Key, T, Compare, Allocator, Policy, Traits > &y)
- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>
bool `operator>` (const `map`< Key, T, Compare, Allocator, Policy, Traits > &x, const `map`< Key, T, Compare, Allocator, Policy, Traits > &y)
- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>
bool `operator>=` (const `map`< Key, T, Compare, Allocator, Policy, Traits > &x, const `map`< Key, T, Compare, Allocator, Policy, Traits > &y)

- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
void **swap** (`set< Key, Compare, Allocator, Policy, Traits > &x`, `set< Key, Compare, Allocator, Policy, Traits > &y`)
- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
bool **operator==** (`const multimap< Key, T, Compare, Allocator, Policy, Traits > &x`, `const multimap< Key, T, Compare, Allocator, Policy, Traits > &y`)
- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
bool **operator<** (`const multimap< Key, T, Compare, Allocator, Policy, Traits > &x`, `const multimap< Key, T, Compare, Allocator, Policy, Traits > &y`)
- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
bool **operator!=** (`const multimap< Key, T, Compare, Allocator, Policy, Traits > &x`, `const multimap< Key, T, Compare, Allocator, Policy, Traits > &y`)
- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
bool **operator>** (`const multimap< Key, T, Compare, Allocator, Policy, Traits > &x`, `const multimap< Key, T, Compare, Allocator, Policy, Traits > &y`)
- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
bool **operator>=** (`const multimap< Key, T, Compare, Allocator, Policy, Traits > &x`, `const multimap< Key, T, Compare, Allocator, Policy, Traits > &y`)
- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
bool **operator<=** (`const multimap< Key, T, Compare, Allocator, Policy, Traits > &x`, `const multimap< Key, T, Compare, Allocator, Policy, Traits > &y`)
- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
void **swap** (`multimap< Key, T, Compare, Allocator, Policy, Traits > &x`, `multimap< Key, T, Compare, Allocator, Policy, Traits > &y`)
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
bool **operator==** (`const multiset< Key, Compare, Allocator, Policy, Traits > &x`, `const multiset< Key, Compare, Allocator, Policy, Traits > &y`)
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
bool **operator<** (`const multiset< Key, Compare, Allocator, Policy, Traits > &x`, `const multiset< Key, Compare, Allocator, Policy, Traits > &y`)
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
bool **operator!=** (`const multiset< Key, Compare, Allocator, Policy, Traits > &x`, `const multiset< Key, Compare, Allocator, Policy, Traits > &y`)
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
bool **operator>** (`const multiset< Key, Compare, Allocator, Policy, Traits > &x`, `const multiset< Key, Compare, Allocator, Policy, Traits > &y`)
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
bool **operator>=** (`const multiset< Key, Compare, Allocator, Policy, Traits > &x`, `const multiset< Key, Compare, Allocator, Policy, Traits > &y`)
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
bool **operator<=** (`const multiset< Key, Compare, Allocator, Policy, Traits > &x`, `const multiset< Key, Compare, Allocator, Policy, Traits > &y`)
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
void **swap** (`multiset< Key, Compare, Allocator, Policy, Traits > &x`, `multiset< Key, Compare, Allocator, Policy, Traits > &y`)

Variables

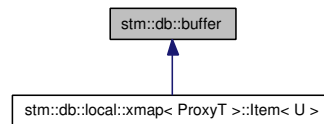
- const `size_type locking` = 1
Policy flags (bitwise orable).

11 SysToMath DB C++ Libraries Class Documentation

11.1 stm::db::buffer Class Reference

```
#include <dbmap.hpp>
```

Inheritance diagram for stm::db::buffer:



11.1.1 Detailed Description

The interface class `buffer` serves as a `buffer` abstraction for the conversion of application key and data objects into their byte serialized form needed to store them in Berkeley databases and vice versa.

Its methods shall be used in the user supplied specializations of class template `db::stream<T>`.

Definition at line 417 of file `dbmap.hpp`.

Public Member Functions

- virtual void `append` (const `byte` *`src`, `size_type` `size`)=0
Append size bytes beginning at src to the buffer.
- virtual const `byte` * `extract` (`size_type` `size`)=0
Return a pointer to the next size bytes of the buffer.
- virtual bool `endiancvt` () const =0
Return true, if arithmetic types have to be endian converted.
- template<typename T>
`buffer` & `operator<<` (const T &value)
Insert operator implementing its operation by means of `stream<T>::get()`.
- template<typename T>
`buffer` & `operator>>` (T &value)
Extractor operator implementing its operation by means of `stream<T>::put()`.

11.1.2 Member Function Documentation

11.1.2.1 virtual void stm::db::buffer::append (const byte * src, size_type size) [pure virtual]

Append size bytes beginning at `src` to the `buffer`.

This method shall be used by the implementations of the user supplied partial specializations of `stream<T, Enable>::put()`.

11.1.2.2 virtual const byte* stm::db::buffer::extract (size_type size) [pure virtual]

Return a pointer to the next size bytes of the [buffer](#).

This method shall be used by the implementations of the user supplied partial specializations of [stream<T, Enable>::get\(\)](#).

11.1.2.3 virtual bool stm::db::buffer::endiancvt () const [pure virtual]

Return true, if arithmetic types have to be endian converted.

This method is intended to be used by the implementations of the user supplied partial specializations of [stream<T, Enable>::put\(\)](#) and specializations of [stream<T, Enable>::get\(\)](#).

Implemented in [stm::db::local::xmap< ProxyT >::Item< U >](#).

11.1.2.4 template<typename T> buffer & stm::db::buffer::operator<< (const T & value) [inline]

Insert operator implementing its operation by means of [stream<T>::get\(\)](#).

Definition at line 3130 of file [dbxmap.hpp](#).

References [get\(\)](#).

11.1.2.5 template<typename T> buffer & stm::db::buffer::operator>> (T & value) [inline]

Extractor operator implementing its operation by means of [stream<T>::put\(\)](#).

Definition at line 3138 of file [dbxmap.hpp](#).

References [put\(\)](#).

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

- [dbmap.hpp](#)
- [dbxmap.hpp](#)

11.2 stm::db::sql::connection Class Reference

```
#include <dbsql.hpp>
```

11.2.1 Detailed Description

Class representing a SQL database [connection](#).

Definition at line 701 of file [dbsql.hpp](#).

Public Member Functions

- **connection** (const std::string &[path](#)) throw (std::runtime_error)
- const std::string & [path](#) () const
Returns the database [connection path](#) name..
- bool [existsTable](#) (const std::string &name) const throw ()

Returns true, if the *table* name exists for this *connection*, else false.

- void `createTable` (const std::string &name, const std::string &type, const table::layout &columns) throw (std::runtime_error)

If a *table* type is defined and a *table* name does not already exist, create a *table* named name of type type with layout columns for this *connection*, else throw.
- template<class InputIterator>
 void `createTable` (const std::string &name, const std::string &type, InputIterator firstColumn, InputIterator lastColumn) throw (std::runtime_error)

If a *table* type is defined and a *table* name does not already exist, create a *table* named name of type type with layout determined by the range [**firstColumn*, **lastColumn*) for this *connection*, else throw.
- bool `deleteTable` (const std::string &name, bool force=false)

If a *table* named name exists for this *connection*, delete it, if it is not active or force is true (default: false).
- bool `activateTable` (const std::string &name, size_type cacheSizeKb=0) throw (std::runtime_error)

If a *table* named name exists for this *connection*, activate it, if it is not already active.
- bool `deactivateTable` (const std::string &name)

If a *table* named name exists for this *connection*, deactivate it, if it is active.
- `table` & `getTable` (const std::string &name) const throw (std::runtime_error)
- const std::string & `getTableType` (const std::string &name) const throw (std::runtime_error)
- const table::descmap & `tables` () const

Returns a constant reference to the *table map* which describes all tables of this *connection*.
- table::desc * `getTableDescr` (const std::string &name)

Returns the address of the *table* descriptor of *table* name, if it exists, else the NULL pointer.
- result * `execsql` (const std::string &cmd) throw (std::runtime_error)

Interpretes the SQL command cmd and returns a pointer to an according result object.
- int `errno` () const

Returns the numerical error status of the last SQL command execution.
- const std::string & `errmsg` () const

Returns the textual error status of the last SQL command execution.

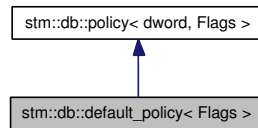
The documentation for this class was generated from the following files:

- [dbsql.hpp](#)
- [dbxsql.hpp](#)
- [dbsql.cpp](#)

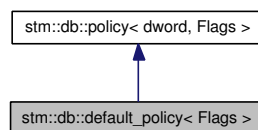
11.3 stm::db::default_policy< Flags > Struct Template Reference

```
#include <dbmap.hpp>
```

Inheritance diagram for stm::db::default_policy< Flags >:



Collaboration diagram for stm::db::default_policy< Flags >:



11.3.1 Detailed Description

template<size_type Flags> struct stm::db::default_policy< Flags >

Default [policy](#) template parameter for class templates [db::map](#), [db::set](#), [db::multimap](#) and [db::multiset](#).

Definition at line 321 of file [dbmap.hpp](#).

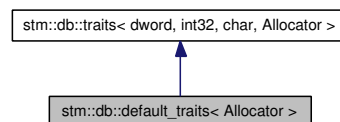
The documentation for this struct was generated from the following file:

- [dbmap.hpp](#)

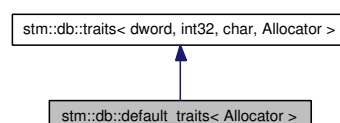
11.4 stm::db::default_traits< Allocator > Struct Template Reference

```
#include <dbmap.hpp>
```

Inheritance diagram for stm::db::default_traits< Allocator >:



Collaboration diagram for stm::db::default_traits< Allocator >:



11.4.1 Detailed Description

`template<class Allocator> struct stm::db::default_traits< Allocator >`

Default `traits` template parameter for class templates `db::map`, `db::set`, `db::multimap` and `db::multiset`.

Definition at line 349 of file `dbmap.hpp`.

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

- [dbmap.hpp](#)

11.5 `stm::db::map< Key, T, Compare, Allocator, Policy, Traits >` Class Template Reference

```
#include <dbmap.hpp>
```

Inheritance diagram for `stm::db::map< Key, T, Compare, Allocator, Policy, Traits >`:



11.5.1 Detailed Description

```
template<class Key, class T, class Compare = std::less<Key>, class Allocator =
std::allocator<std::pair<const Key, T> >, class Policy = default_policy<0>, class Traits =
default_traits<Allocator>>> class stm::db::map< Key, T, Compare, Allocator, Policy, Traits >
```

The `db::map` class template is a kind of associative container that supports unique keys (contains at most one of each key value) and provides for fast retrieval of values of another type `T` based on the keys.

The `db::map` class template supports bidirectional iterators (24.1.4). A `db::map` satisfies all of the requirements of a container and of a reversible container (23.1) and of an associative container (23.1.2). A `db::map` also provides most operations described in (23.1.2) for unique keys. This means that a `db::map` supports the `a_uniq` operations in (23.1.2) but not the `a_eq` operations. For a `db::map<Key,T>` the `key_type` is `Key` and the `value_type` is `std::pair<const Key,T>`. Class template `db::map` is modelled after the container `std::map` (23.3.1) and additionally permits named persistent storage of its contents as its implementation is based on the Berkeley database engine. So by means of `db::map` iterators arbitrary big containers are effectively manageable. Moreover, the class template `db::map` provides a member class template `index` permitting quick sorted database access according to user definable secondary keys and sort criteria. Descriptions are provided here only for operations on `db::map` that are not described in (23.3.1) for `std::map` and for operations where there is additional semantic information.

Examples:

[dbmaptest.cpp](#).

Definition at line 580 of file `dbmap.hpp`.

Type definitions

- typedef `map< Key, T, Compare, Allocator, Policy, Traits >` `db_type`

- typedef Key **key_type**
- typedef T **mapped_type**
- typedef std::pair< const Key, T > **value_type**
- typedef Compare **key_compare**
- typedef proxy::value_compare **value_compare**
- typedef Allocator **allocator_type**
- typedef Policy **policy_type**
- typedef Traits **traits_type**
- typedef Allocator::reference **reference**
- typedef Allocator::const_reference **const_reference**
- typedef Allocator::pointer **pointer**
- typedef Allocator::const_pointer **const_pointer**
- typedef proxy::iterator **iterator**
- typedef proxy::const_iterator **const_iterator**
- typedef proxy::reverse_iterator **reverse_iterator**
- typedef proxy::const_reverse_iterator **const_reverse_iterator**
- typedef traits_type::size_type **size_type**
- typedef traits_type::difference_type **difference_type**
- typedef **traits_type::char_type** **char_type**
- typedef **traits_type::string_type** **string_type**
- typedef policy_type::flags_type **flags_type**

Bitmask type.

Flag value definitions

////////////////////////////////////

- static const **flags_type** **noflags** = proxy::noflags
- static const **flags_type** **truncate** = proxy::truncate
- static const **flags_type** **forceendiancvt** = proxy::forceendiancvt
- static const **flags_type** **multiprocessing** = proxy::multiprocessing

Construction and destruction

////////////////////////////////////

- **map** ()
Default constructor.
- **map** (const **string_type** &name, **flags_type** flags=noflags, const key_compare &comp=key_compare(), const allocator_type &alloc=allocator_type())
If name is empty, the `db::map` is not stored persistently on disk.
- template<class InputIterator>
map (InputIterator first, InputIterator last, const **string_type** &name=**string_type**(), **flags_type** flags=noflags, const key_compare &comp=key_compare(), const allocator_type &alloc=allocator_type())
If name is empty, the `db::map` is not stored persistently on disk.

- `map` (const `map< Key, T, Compare, Allocator, Policy, Traits >` &rhs, const `string_type` &name=`string_type`())
If name is empty, the `db::map` is not stored persistently on disk.
- `map< Key, T, Compare, Allocator, Policy, Traits >` & `operator=` (const `map< Key, T, Compare, Allocator, Policy, Traits >` &rhs)
Assignment is only allowed for a non persistent right hand side.
- void `swap` (`map< Key, T, Compare, Allocator, Policy, Traits >` &other) throw ()
Swap this `db::map` object with the `db::map` object other.
- `~map` ()

Iterators

////////////////////////////////////

- iterator `begin` ()
- const_iterator `begin` () const
- iterator `end` ()
- const_iterator `end` () const
- reverse_iterator `rbegin` ()
- const_reverse_iterator `rbegin` () const
- reverse_iterator `rend` ()
- const_reverse_iterator `rend` () const

Capacity

////////////////////////////////////

- bool `empty` () const
- size_type `size` () const
- size_type `max_size` () const

Element access

////////////////////////////////////

- mapped_type & `operator[]` (const key_type &key)

Modifiers

////////////////////////////////////

- iterator `insert` (iterator hint, const value_type &value)
- std::pair< typename db_type::iterator, bool > `insert` (const value_type &value)
- template<typename InputIterator>
void `insert` (InputIterator first, InputIterator last)
- void `erase` (iterator position)

- `size_type erase` (const key_type &key)
- void `erase` (iterator first, iterator last)
- void `clear` ()

Observers

//

- key_compare `key_comp` () const
- value_compare `value_comp` () const
- allocator_type `get_allocator` () const
- bool `endiancv` () const
- `flags_type` `get_flags` () const
- const `string_type` & `get_name` () const

`db::map` key operations

//

- iterator `find` (const key_type &key)
- const_iterator `find` (const key_type &key) const
- reverse_iterator `rfind` (const key_type &key)
- const_reverse_iterator `rfind` (const key_type &key) const
- size_type `count` (const key_type &key) const
- iterator `lower_bound` (const key_type &key)
- const_iterator `lower_bound` (const key_type &key) const
- iterator `upper_bound` (const key_type &key)
- const_iterator `upper_bound` (const key_type &key) const
- std::pair< iterator, iterator > `equal_range` (const key_type &key)
- std::pair< const_iterator, const_iterator > `equal_range` (const key_type &key) const

Private Types

- typedef local::pxmap< Key, T, Compare, Allocator, Policy, Traits, false > `proxy`

Classes

- class `index`
Member class template `db::map::index`.
- class `locker`
xxx

11.5.2 Member Typedef Documentation

11.5.2.1 `template<class Key, class T, class Compare = std::less<Key>, class Allocator = std::allocator<std::pair<const Key, T> >, class Policy = default_policy<0>, class Traits = default_traits<Allocator>> typedef policy_type::flags_type stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::flags_type`

Bitmask type.

Definition at line 635 of file `dbmap.hpp`.

11.5.3 Constructor & Destructor Documentation

11.5.3.1 `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits> stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::map () [inline]`

Default constructor.

Definition at line 3558 of file `dbxmap.hpp`.

11.5.3.2 `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits> stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::map (const string_type & name, flags_type flags = noflags, const key_compare & comp = key_compare (), const allocator_type & alloc = allocator_type ()) [inline, explicit]`

If name is empty, the `db::map` is not stored persistently on disk.

Definition at line 3563 of file `dbxmap.hpp`.

11.5.3.3 `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits> template<class InputIterator> stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::map (InputIterator first, InputIterator last, const string_type & name = string_type (), flags_type flags = noflags, const key_compare & comp = key_compare (), const allocator_type & alloc = allocator_type ()) [inline]`

If name is empty, the `db::map` is not stored persistently on disk.

Definition at line 3576 of file `dbxmap.hpp`.

11.5.3.4 `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits> stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::map (const map< Key, T, Compare, Allocator, Policy, Traits > & rhs, const string_type & name = string_type ()) [inline]`

If name is empty, the `db::map` is not stored persistently on disk.

Only allowed, if the constructed `db::map` is not persistent, or if it has a name different from that of the right hand side.

Definition at line 3592 of file `dbxmap.hpp`.

References `stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::begin()`, `stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::end()`, `stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::get_name()`, and `StmDebugAidsErrmsg`.

11.5.4 Member Function Documentation

11.5.4.1 `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits> map< Key, T, Compare, Allocator, Policy, Traits > & stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::operator= (const map< Key, T, Compare, Allocator, Policy, Traits > & rhs)`
[inline]

Assignment is only allowed for a non persistent right hand side.

Definition at line 3624 of file `dbxmap.hpp`.

References `stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::get_name()`, and `stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::swap()`.

11.5.4.2 `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits> void stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::swap (map< Key, T, Compare, Allocator, Policy, Traits > & other) throw ()` [inline]

Swap this `db::map` object with the `db::map` object `other`.

The method does not throw and guarantees that all valid iterators of either object stay valid.

Definition at line 3608 of file `dbxmap.hpp`.

References `swap()`.

Referenced by `stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::operator=()`.

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

- [dbmap.hpp](#)
- [dbxmap.hpp](#)

11.6 `stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::index< SKey, SCompare >` Class Template Reference

```
#include <dbmap.hpp>
```

11.6.1 Detailed Description

`template<class Key, class T, class Compare = std::less<Key>, class Allocator = std::allocator<std::pair<const Key, T> >, class Policy = default_policy<0>, class Traits = default_traits<Allocator>> template<class SKey, class SCompare> class stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::index< SKey, SCompare >`

Member class template `db::map::index`.

Examples:

[dbmaptest.cpp](#).

Definition at line 871 of file `dbmap.hpp`.

Public Types

- typedef `map< Key, T, Compare, Allocator, Policy, Traits > db_type`

- typedef `db_type::key_type` **key_type**
- typedef `db_type::mapped_type` **mapped_type**
- typedef `db_type::value_type` **value_type**
- typedef `db_type::policy_type` **policy_type**
- typedef `db_type::traits_type` **traits_type**
- typedef `db_type::size_type` **size_type**
- typedef `db_type::difference_type` **difference_type**
- typedef `db_type::char_type` **char_type**
- typedef `db_type::string_type` **string_type**
- typedef `SKey` **skey_type**
- typedef `SCompare` **skey_compare**
- typedef `proxy::skeyfct_type` **skeyfct_type**
- typedef `proxy::iterator` **iterator**
- typedef `proxy::const_iterator` **const_iterator**
- typedef `proxy::reverse_iterator` **reverse_iterator**
- typedef `proxy::const_reverse_iterator` **const_reverse_iterator**
- typedef `db_type::flags_type` **flags_type**

Public Member Functions

- **index** (const `db_type` &db, const `string_type` &name)
- **index** (const `db_type` &db, `skeyfct_type` skeyfct, const `string_type` &name=`string_type`(), `flags_type` flags=`noflags`, const `skey_compare` &comp=`skey_compare`())
- **index** (const `index< SKey, SCompare >` &rhs, const `string_type` &name=`string_type`())
- `index< SKey, SCompare >` & **operator=** (const `index< SKey, SCompare >` &rhs)
- void **swap** (`index< SKey, SCompare >` &other) throw ()
- iterator **begin** ()
- const_iterator **begin** () const
- iterator **end** ()
- const_iterator **end** () const
- reverse_iterator **rbegin** ()
- const_reverse_iterator **rbegin** () const
- reverse_iterator **rend** ()
- const_reverse_iterator **rend** () const
- bool **empty** () const
- size_type **size** () const
- size_type **max_size** () const
- void **erase** (iterator position)
- size_type **erase** (const `skey_type` &skey)
- void **erase** (iterator first, iterator last)
- void **clear** ()
- const `db_type` & **dbmap** () const
- `skey_compare` **skey_comp** () const
- `skeyfct_type` **get_skeyfct** () const
- bool **endiancv** () const
- `flags_type` **get_flags** () const
- const `string_type` & **get_name** () const
- iterator **find** (const `skey_type` &skey)
- const_iterator **find** (const `skey_type` &skey) const
- reverse_iterator **rfind** (const `skey_type` &skey)

- `const_reverse_iterator` **rfind** (`const skey_type &skey`) `const`
- `size_type` **count** (`const skey_type &skey`) `const`
- `iterator` **lower_bound** (`const skey_type &skey`)
- `const_iterator` **lower_bound** (`const skey_type &skey`) `const`
- `iterator` **upper_bound** (`const skey_type &skey`)
- `const_iterator` **upper_bound** (`const skey_type &skey`) `const`
- `std::pair< iterator, iterator >` **equal_range** (`const skey_type &skey`)
- `std::pair< const_iterator, const_iterator >` **equal_range** (`const skey_type &skey`) `const`

Static Public Attributes

- static `const flags_type` **noflags** = `proxy::noflags`
- static `const flags_type` **reindex** = `proxy::reindex`

Private Types

- `typedef local::pindex< Key, T, Compare, Allocator, Policy, Traits, SKey, SCompare >` **proxy**

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

- [dbmap.hpp](#)
- [dbxmap.hpp](#)

11.7 `stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::locker` Class Reference

```
#include <dbmap.hpp>
```

11.7.1 Detailed Description

```
template<class Key, class T, class Compare = std::less<Key>, class Allocator =
std::allocator<std::pair<const Key, T> >, class Policy = default_policy<0>, class Traits =
default_traits<Allocator>> class stm::db::map< Key, T, Compare, Allocator, Policy, Traits
>::locker
```

xxx

Member class [db::map::locker](#).

Definition at line 847 of file `dbmap.hpp`.

Public Types

- `typedef map< Key, T, Compare, Allocator, Policy, Traits >` **db_type**
- `typedef boost::function< void()>` **cb_type**

Public Member Functions

- **locker** (const [db_type](#) &db, cb_type unlockfct=0)
- `template<class IndexT>`
locker (const IndexT &idx, cb_type unlockfct=0)

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

- [dbmap.hpp](#)
- [dbxmap.hpp](#)

11.8 `stm::db::multimap`< Key, T, Compare, Allocator, Policy, Traits > Class Template Reference

```
#include <dbmap.hpp>
```

11.8.1 Detailed Description

```
template<class Key, class T, class Compare = std::less<Key>, class Allocator =  
std::allocator<std::pair<const Key, T> >, class Policy = default_policy<0>, class Traits =  
default_traits<Allocator>> class stm::db::multimap< Key, T, Compare, Allocator, Policy, Traits >
```

Class template [db::multimap](#).

Definition at line 1337 of file `dbmap.hpp`.

Public Types

- typedef [multimap](#)< Key, T, Compare, Allocator, Policy, Traits > **db_type**
- typedef Key **key_type**
- typedef T **mapped_type**
- typedef std::pair< const Key, T > **value_type**
- typedef Compare **key_compare**
- typedef proxy::value_compare **value_compare**
- typedef Allocator **allocator_type**
- typedef Policy **policy_type**
- typedef Traits **traits_type**
- typedef Allocator::reference **reference**
- typedef Allocator::const_reference **const_reference**
- typedef Allocator::pointer **pointer**
- typedef Allocator::const_pointer **const_pointer**
- typedef proxy::iterator **iterator**
- typedef proxy::const_iterator **const_iterator**
- typedef proxy::reverse_iterator **reverse_iterator**
- typedef proxy::const_reverse_iterator **const_reverse_iterator**
- typedef traits_type::size_type **size_type**
- typedef traits_type::difference_type **difference_type**
- typedef **traits_type::char_type** **char_type**
- typedef **traits_type::string_type** **string_type**
- typedef policy_type::flags_type **flags_type**

Public Member Functions

- **multimap** (const **string_type** &name=**string_type**(), flags_type flags=noflags, const key_compare &comp=key_compare(), const allocator_type &alloc=allocator_type())
- template<class InputIterator>
multimap (InputIterator first, InputIterator last, const **string_type** &name=**string_type**(), flags_type flags=noflags, const key_compare &comp=key_compare(), const allocator_type &alloc=allocator_type())
- **multimap** (const `multimap< Key, T, Compare, Allocator, Policy, Traits >` &rhs, const **string_type** &name=**string_type**())
- `multimap< Key, T, Compare, Allocator, Policy, Traits >` & **operator=** (const `multimap< Key, T, Compare, Allocator, Policy, Traits >` &rhs)
- void **swap** (`multimap< Key, T, Compare, Allocator, Policy, Traits >` &other) throw ()
- iterator **begin** ()
- const_iterator **begin** () const
- iterator **end** ()
- const_iterator **end** () const
- reverse_iterator **rbegin** ()
- const_reverse_iterator **rbegin** () const
- reverse_iterator **rend** ()
- const_reverse_iterator **rend** () const
- bool **empty** () const
- size_type **size** () const
- size_type **max_size** () const
- iterator **insert** (iterator hint, const value_type &value)
- iterator **insert** (const value_type &value)
- template<typename InputIterator>
void **insert** (InputIterator first, InputIterator last)
- void **erase** (iterator position)
- size_type **erase** (const key_type &key)
- void **erase** (iterator first, iterator last)
- void **clear** ()
- key_compare **key_comp** () const
- value_compare **value_comp** () const
- allocator_type **get_allocator** () const
- bool **endiancvt** () const
- flags_type **get_flags** () const
- const **string_type** & **get_name** () const
- iterator **find** (const key_type &key)
- const_iterator **find** (const key_type &key) const
- reverse_iterator **rfind** (const key_type &key)
- const_reverse_iterator **rfind** (const key_type &key) const
- size_type **count** (const key_type &key) const
- iterator **lower_bound** (const key_type &key)
- const_iterator **lower_bound** (const key_type &key) const
- iterator **upper_bound** (const key_type &key)
- const_iterator **upper_bound** (const key_type &key) const
- std::pair< iterator, iterator > **equal_range** (const key_type &key)
- std::pair< const_iterator, const_iterator > **equal_range** (const key_type &key) const

Static Public Attributes

- static const flags_type **noflags** = proxy::noflags
- static const flags_type **truncate** = proxy::truncate
- static const flags_type **forceendiancv**t = proxy::forceendiancv
- static const flags_type **multiprocessing** = proxy::multiprocessing

Private Types

- typedef local::pxmap< Key, T, Compare, Allocator, Policy, Traits, true > **proxy**

Classes

- class `locker`
Member class `db::multimap::locker`.

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

- [dbmap.hpp](#)
- [dbxmap.hpp](#)

11.9 `stm::db::multimap< Key, T, Compare, Allocator, Policy, Traits >::locker` Class Reference

```
#include <dbmap.hpp>
```

11.9.1 Detailed Description

```
template<class Key, class T, class Compare = std::less<Key>, class Allocator =  
std::allocator<std::pair<const Key, T> >, class Policy = default_policy<0>, class Traits =  
default_traits<Allocator>> class stm::db::multimap< Key, T, Compare, Allocator, Policy, Traits  
>::locker
```

Member class `db::multimap::locker`.

Definition at line 1510 of file `dbmap.hpp`.

Public Types

- typedef `multimap< Key, T, Compare, Allocator, Policy, Traits >` **db_type**
- typedef `boost::function< void()>` **cb_type**

Public Member Functions

- **locker** (const `db_type` &db, `cb_type` unlockfct=0)

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

- [dbmap.hpp](#)
- [dbxmap.hpp](#)

11.10 `stm::db::multiset< Key, Compare, Allocator, Policy, Traits >` Class Template Reference

```
#include <dbmap.hpp>
```

11.10.1 Detailed Description

`template<class Key, class Compare = std::less<Key>, class Allocator = std::allocator<Key>, class Policy = default_policy<0>, class Traits = default_traits<Allocator>> class stm::db::multiset< Key, Compare, Allocator, Policy, Traits >`

Class template [db::multiset](#).

Definition at line 1536 of file `dbmap.hpp`.

Public Types

- typedef `multiset< Key, Compare, Allocator, Policy, Traits >` **db_type**
- typedef Key **key_type**
- typedef key_type **value_type**
- typedef Compare **key_compare**
- typedef Compare **value_compare**
- typedef Allocator **allocator_type**
- typedef Policy **policy_type**
- typedef Traits **traits_type**
- typedef Allocator::reference **reference**
- typedef Allocator::const_reference **const_reference**
- typedef Allocator::pointer **pointer**
- typedef Allocator::const_pointer **const_pointer**
- typedef proxy::iterator **iterator**
- typedef proxy::const_iterator **const_iterator**
- typedef proxy::reverse_iterator **reverse_iterator**
- typedef proxy::const_reverse_iterator **const_reverse_iterator**
- typedef traits_type::size_type **size_type**
- typedef traits_type::difference_type **difference_type**
- typedef **traits_type::char_type** **char_type**
- typedef **traits_type::string_type** **string_type**
- typedef policy_type::flags_type **flags_type**

Public Member Functions

- **multiset** (const **string_type** &name=**string_type**(), flags_type flags=noflags, const key_compare &comp=key_compare(), const allocator_type &alloc=allocator_type())
- `template<class InputIterator>`
multiset (InputIterator first, InputIterator last, const **string_type** &name=**string_type**(), flags_type flags=noflags, const key_compare &comp=key_compare(), const allocator_type &alloc=allocator_type())
- **multiset** (const `multiset< Key, Compare, Allocator, Policy, Traits >` &rhs, const **string_type** &name=**string_type**())
- `multiset< Key, Compare, Allocator, Policy, Traits >` & **operator=** (const `multiset< Key, Compare, Allocator, Policy, Traits >` &rhs)

- void **swap** (`multiset< Key, Compare, Allocator, Policy, Traits > &other`) throw ()
- iterator **begin** ()
- const_iterator **begin** () const
- iterator **end** ()
- const_iterator **end** () const
- reverse_iterator **rbegin** ()
- const_reverse_iterator **rbegin** () const
- reverse_iterator **rend** ()
- const_reverse_iterator **rend** () const
- bool **empty** () const
- size_type **size** () const
- size_type **max_size** () const
- iterator **insert** (iterator hint, const value_type &value)
- iterator **insert** (const value_type &value)
- template<typename InputIterator>
void **insert** (InputIterator first, InputIterator last)
- void **erase** (iterator position)
- size_type **erase** (const key_type &key)
- void **erase** (iterator first, iterator last)
- void **clear** ()
- key_compare **key_comp** () const
- value_compare **value_comp** () const
- allocator_type **get_allocator** () const
- bool **endiancv** () const
- flags_type **get_flags** () const
- const **string_type** & **get_name** () const
- iterator **find** (const key_type &key)
- const_iterator **find** (const key_type &key) const
- reverse_iterator **rfind** (const key_type &key)
- const_reverse_iterator **rfind** (const key_type &key) const
- size_type **count** (const key_type &key) const
- iterator **lower_bound** (const key_type &key)
- const_iterator **lower_bound** (const key_type &key) const
- iterator **upper_bound** (const key_type &key)
- const_iterator **upper_bound** (const key_type &key) const
- std::pair< iterator, iterator > **equal_range** (const key_type &key)
- std::pair< const_iterator, const_iterator > **equal_range** (const key_type &key) const

Static Public Attributes

- static const flags_type **noflags** = proxy::noflags
- static const flags_type **truncate** = proxy::truncate
- static const flags_type **forceendiancv** = proxy::forceendiancv
- static const flags_type **multiprocessing** = proxy::multiprocessing

Private Types

- typedef local::pxmap< Key, [null_type](#), Compare, Allocator, Policy, Traits, true > **proxy**

Classes

- class `locker`
Member class `db::multiset::locker`.

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

- `dbmap.hpp`
- `dbxmap.hpp`

11.11 `stm::db::multiset< Key, Compare, Allocator, Policy, Traits >::locker` Class Reference

```
#include <dbmap.hpp>
```

11.11.1 Detailed Description

```
template<class Key, class Compare = std::less<Key>, class Allocator = std::allocator<Key>, class Policy = default_policy<0>, class Traits = default_traits<Allocator>> class stm::db::multiset< Key, Compare, Allocator, Policy, Traits >::locker
```

Member class `db::multiset::locker`.

Definition at line 1708 of file `dbmap.hpp`.

Public Types

- typedef `multiset< Key, Compare, Allocator, Policy, Traits > db_type`
- typedef `boost::function< void()> cb_type`

Public Member Functions

- `locker` (const `db_type` &db, `cb_type` unlockfct=0)

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

- `dbmap.hpp`
- `dbxmap.hpp`

11.12 `stm::db::null_type` Struct Reference

```
#include <dbmap.hpp>
```

11.12.1 Detailed Description

Empty type with default constructor doing nothing.

Definition at line 356 of file `dbmap.hpp`.

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

- [dbmap.hpp](#)
- [dbxmap.hpp](#)

11.13 `stm::db::optional< T >` Struct Template Reference

```
#include <dbmap.hpp>
```

11.13.1 Detailed Description

`template<typename T> struct stm::db::optional< T >`

The class template `optional<T>` is a `boost::variant` type consisting of a `db::null_type` member, if the instance holds no T value and a T member, if it holds a T value.

Examples:

[dbmaptest.cpp](#).

Definition at line 378 of file `dbmap.hpp`.

Public Member Functions

- `bool exists () const`
Returns true, if this object holds a T value, else false.

11.13.2 Member Function Documentation

11.13.2.1 `template<typename T> bool stm::db::optional< T >::exists () const` `[inline]`

Returns true, if this object holds a T value, else false.

Definition at line 2994 of file `dbxmap.hpp`.

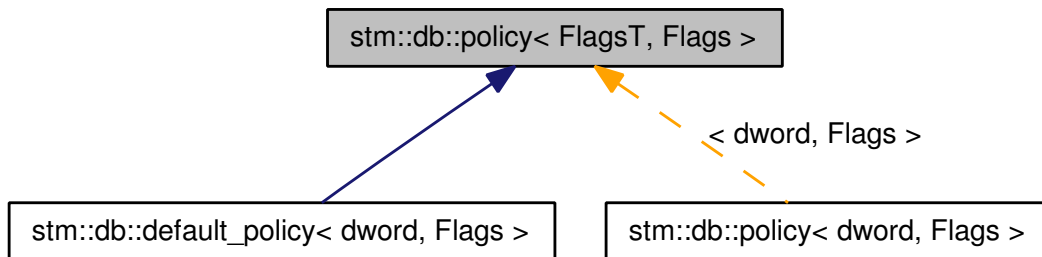
The documentation for this struct was generated from the following files:

- [dbmap.hpp](#)
- [dbxmap.hpp](#)

11.14 `stm::db::policy< FlagsT, Flags >` Struct Template Reference

```
#include <dbmap.hpp>
```

Inheritance diagram for `stm::db::policy< FlagsT, Flags >`:



11.14.1 Detailed Description

```
template<class FlagsT, FlagsT Flags> struct stm::db::policy< FlagsT, Flags >
```

Class template serving as Policy template parameter for class templates `db::map`, `db::set`, `db::multimap` and `db::multiset`.

Definition at line 310 of file `dbmap.hpp`.

Public Types

- typedef `FlagsT` `flags_type`
- typedef `boost::mpl::bool_<(Flags &flags_type(locking))==flags_type(locking)>` `lock`

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

- [dbmap.hpp](#)

11.15 `stm::db::set< Key, Compare, Allocator, Policy, Traits >` Class Template Reference

```
#include <dbmap.hpp>
```

11.15.1 Detailed Description

```
template<class Key, class Compare = std::less<Key>, class Allocator = std::allocator<Key>, class Policy = default_policy<0>, class Traits = default_traits<Allocator>> class stm::db::set< Key, Compare, Allocator, Policy, Traits >
```

The `db::set` class template is a kind of associative container that supports unique keys (contains at most one of each key value) and provides for fast retrieval based on these keys.

The `db::set` class template supports bidirectional iterators (24.1.4). A `db::set` satisfies all of the requirements of a container and of a reversible container (23.1) and of an associative container (23.1.2). A `db::set` also provides most operations described in (23.1.2) for unique keys. This means that a `db::set` supports the `a_uniq` operations in (23.1.2) but not the `a_eq` operations. For a `db::set<Key>` the `key_type` is `Key` and the `value_type` is also `Key`. Class template `db::set` is modelled after the container `std::set` (23.3.3) and additionally permits named persistent storage of its contents as its implementation is based on the Berkeley database engine. So by means of `db::set` iterators arbitrary big containers are effectively manageable.

Descriptions are provided here only for operations on `db::set` that are not described in (23.3.3) for `std::set` and for operations where there is additional semantic information.

Definition at line 1055 of file `dbmap.hpp`.

Type definitions

- typedef `set`< Key, Compare, Allocator, Policy, Traits > **db_type**
- typedef Key **key_type**
- typedef key_type **value_type**
- typedef Compare **key_compare**
- typedef Compare **value_compare**
- typedef Allocator **allocator_type**
- typedef Policy **policy_type**
- typedef Traits **traits_type**
- typedef Allocator::reference **reference**
- typedef Allocator::const_reference **const_reference**
- typedef Allocator::pointer **pointer**
- typedef Allocator::const_pointer **const_pointer**
- typedef proxy::iterator **iterator**
- typedef proxy::const_iterator **const_iterator**
- typedef proxy::reverse_iterator **reverse_iterator**
- typedef proxy::const_reverse_iterator **const_reverse_iterator**
- typedef traits_type::size_type **size_type**
- typedef traits_type::difference_type **difference_type**
- typedef **traits_type::char_type** `char_type`
- typedef **traits_type::string_type** `string_type`
- typedef policy_type::flags_type **flags_type**

Bitmask type.

Flag value definitions

////////////////////////////////////

- static const **flags_type** `noflags` = proxy::noflags
- static const **flags_type** `truncate` = proxy::truncate
- static const **flags_type** `forceendiancv` = proxy::forceendiancv
- static const **flags_type** `multiprocessing` = proxy::multiprocessing

Construction and destruction

////////////////////////////////////

- `set` ()
Default constructor.
- `set` (const **string_type** &name, **flags_type** flags=noflags, const key_compare &comp=key_compare(), const allocator_type &alloc=allocator_type())
If name is empty, the `db::set` is not stored persistently on disk.

- `template<class InputIterator>`
`set` (`InputIterator first`, `InputIterator last`, `const string_type &name=string_type()`, `flags_type flags=noflags`, `const key_compare &comp=key_compare()`, `const allocator_type &alloc=allocator_type()`)
If name is empty, the `db::set` is not stored persistently on disk.
- `set` (`const set< Key, Compare, Allocator, Policy, Traits > &rhs`, `const string_type &name=string_type()`)
If name is empty, the `db::set` is not stored persistently on disk.
- `set< Key, Compare, Allocator, Policy, Traits > & operator=` (`const set< Key, Compare, Allocator, Policy, Traits > &rhs`)
Assignment is only allowed for a non persistent right hand side.
- `void swap` (`set< Key, Compare, Allocator, Policy, Traits > &other`) `throw ()`
Swap this `db::set` object with the `db::set` object other.
- `~set ()`

Iterators

//

- iterator `begin ()`
- `const_iterator begin () const`
- iterator `end ()`
- `const_iterator end () const`
- reverse_iterator `rbegin ()`
- `const_reverse_iterator rbegin () const`
- reverse_iterator `rend ()`
- `const_reverse_iterator rend () const`

Capacity

//

- `bool empty () const`
- `size_type size () const`
- `size_type max_size () const`

Modifiers

//

- iterator `insert` (`iterator hint`, `const value_type &value`)
- `std::pair< typename db_type::iterator, bool > insert` (`const value_type &value`)
- `template<typename InputIterator>`
`void insert` (`InputIterator first`, `InputIterator last`)

- void **erase** (iterator position)
- `size_type` **erase** (const `key_type` &key)
- void **erase** (iterator first, iterator last)
- void **clear** ()

Observers

//

- `key_compare` **key_comp** () const
- `value_compare` **value_comp** () const
- `allocator_type` **get_allocator** () const
- bool **endiancv** () const
- `flags_type` **get_flags** () const
- const `string_type` & **get_name** () const

`db::set` key operations

//

- iterator **find** (const `key_type` &key)
- const_iterator **find** (const `key_type` &key) const
- reverse_iterator **rfind** (const `key_type` &key)
- const_reverse_iterator **rfind** (const `key_type` &key) const
- `size_type` **count** (const `key_type` &key) const
- iterator **lower_bound** (const `key_type` &key)
- const_iterator **lower_bound** (const `key_type` &key) const
- iterator **upper_bound** (const `key_type` &key)
- const_iterator **upper_bound** (const `key_type` &key) const
- `std::pair< iterator, iterator >` **equal_range** (const `key_type` &key)
- `std::pair< const_iterator, const_iterator >` **equal_range** (const `key_type` &key) const

Private Types

- typedef `local::pxmap< Key, null_type, Compare, Allocator, Policy, Traits, false >` **proxy**

Classes

- class `locker`

xxx

11.15.2 Member Typedef Documentation

11.15.2.1 `template<class Key, class Compare = std::less<Key>, class Allocator = std::allocator<Key>, class Policy = default_policy<0>, class Traits = default_traits<Allocator>>`
typedef `policy_type::flags_type` `stm::db::set< Key, Compare, Allocator, Policy, Traits >::flags_type`

Bitmask type.

Definition at line 1101 of file `dbmap.hpp`.

11.15.3 Constructor & Destructor Documentation

11.15.3.1 `template<class Key, class Compare, class Allocator, class Policy, class Traits> stm::db::set< Key, Compare, Allocator, Policy, Traits >::set () [inline]`

Default constructor.

Definition at line 4643 of file `dbxmap.hpp`.

11.15.3.2 `template<class Key, class Compare, class Allocator, class Policy, class Traits> stm::db::set< Key, Compare, Allocator, Policy, Traits >::set (const string_type & name, flags_type flags = noflags, const key_compare & comp = key_compare (), const allocator_type & alloc = allocator_type ()) [inline, explicit]`

If name is empty, the `db::set` is not stored persistently on disk.

Definition at line 4648 of file `dbxmap.hpp`.

11.15.3.3 `template<class Key, class Compare, class Allocator, class Policy, class Traits> template<class InputIterator> stm::db::set< Key, Compare, Allocator, Policy, Traits >::set (InputIterator first, InputIterator last, const string_type & name = string_type (), flags_type flags = noflags, const key_compare & comp = key_compare (), const allocator_type & alloc = allocator_type ()) [inline]`

If name is empty, the `db::set` is not stored persistently on disk.

Definition at line 4661 of file `dbxmap.hpp`.

11.15.3.4 `template<class Key, class Compare, class Allocator, class Policy, class Traits> stm::db::set< Key, Compare, Allocator, Policy, Traits >::set (const set< Key, Compare, Allocator, Policy, Traits > & rhs, const string_type & name = string_type ()) [inline]`

If name is empty, the `db::set` is not stored persistently on disk.

Only allowed, if the constructed `db::set` is not persistent, or if it has a name different from that of the right hand side.

Definition at line 4677 of file `dbxmap.hpp`.

References `stm::db::set< Key, Compare, Allocator, Policy, Traits >::begin()`, `stm::db::set< Key, Compare, Allocator, Policy, Traits >::end()`, `stm::db::set< Key, Compare, Allocator, Policy, Traits >::get_name()`, and `StmDebugAidsErrmsg`.

11.15.4 Member Function Documentation

11.15.4.1 `template<class Key, class Compare, class Allocator, class Policy, class Traits> set< Key, Compare, Allocator, Policy, Traits > & stm::db::set< Key, Compare, Allocator, Policy, Traits >::operator= (const set< Key, Compare, Allocator, Policy, Traits > & rhs) [inline]`

Assignment is only allowed for a non persistent right hand side.

Definition at line 4709 of file `dbxmap.hpp`.

References `stm::db::set< Key, Compare, Allocator, Policy, Traits >::get_name()`, and `stm::db::set< Key, Compare, Allocator, Policy, Traits >::swap()`.

11.15.4.2 `template<class Key, class Compare, class Allocator, class Policy, class Traits> void stm::db::set< Key, Compare, Allocator, Policy, Traits >::swap (set< Key, Compare, Allocator, Policy, Traits > & other) throw () [inline]`

Swap this `db::set` object with the `db::set` object other.

The method does not throw and guarantees that all valid iterators of either object stay valid.

Definition at line 4693 of file `dbxmap.hpp`.

References `swap()`.

Referenced by `stm::db::set< Key, Compare, Allocator, Policy, Traits >::operator=()`.

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

- [dbmap.hpp](#)
- [dbxmap.hpp](#)

11.16 `stm::db::set< Key, Compare, Allocator, Policy, Traits >::locker` Class Reference

```
#include <dbmap.hpp>
```

11.16.1 Detailed Description

`template<class Key, class Compare = std::less<Key>, class Allocator = std::allocator<Key>, class Policy = default_policy<0>, class Traits = default_traits<Allocator>> class stm::db::set< Key, Compare, Allocator, Policy, Traits >::locker`

xxx

Member class [db::set::locker](#).

Definition at line 1306 of file `dbmap.hpp`.

Public Types

- typedef `set< Key, Compare, Allocator, Policy, Traits >` **db_type**
- typedef `boost::function< void()>` **cb_type**

Public Member Functions

- **locker** (const [db_type](#) &db, [cb_type](#) unlockfct=0)
- `template<class IndexT>`
locker (const IndexT &idx, [cb_type](#) unlockfct=0)

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

- [dbmap.hpp](#)
- [dbxmap.hpp](#)

11.17 `stm::db::stream< T, Enable >` Struct Template Reference

```
#include <dbmap.hpp>
```

11.17.1 Detailed Description

`template<typename T, class Enable = void> struct stm::db::stream< T, Enable >`

The class template `stream<T>` on the first hand serves as `stream` type to append values of type `T` in a byte serialized form to the buffers used in the access implementation of Berkeley databases and on the other hand to extract values of type `T` from those buffers.

This non-specialized version of the class template serves as interface definition for the conversions between the value and its `buffer` representation. No implementation of its `put` and `get` methods is provided. Instead for POD types and some other types and type families partial specializations implementing these operations are provided. For all other types and/or type families the user has to supply its own specializations implementing the same interface by means of `put` and `get` methods of other specializations and of `buffer::append()`, `buffer::extract()` and `buffer::endiancvt()`.

Definition at line 463 of file `dbmap.hpp`.

Static Public Member Functions

- static void `put (buffer &buf, const T &value)`
Append the value's byte serialized representation to buf.
- static void `get (buffer &buf, T &value)`
Extract value in its byte serialized representation from buf.

11.17.2 Member Function Documentation

11.17.2.1 `template<typename T, class Enable = void> static void stm::db::stream< T, Enable >::put (buffer & buf, const T & value) [static]`

Append the value's byte serialized representation to buf.

Examples:

[dbmaptest.cpp](#).

11.17.2.2 `template<typename T, class Enable = void> static void stm::db::stream< T, Enable >::get (buffer & buf, T & value) [static]`

Extract value in its byte serialized representation from buf.

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

- [dbmap.hpp](#)

11.18 `stm::db::stream< std::pair< T1, T2 > >` Struct Template Reference

```
#include <dbxmap.hpp>
```

11.18.1 Detailed Description

template<typename T1, typename T2> **struct** stm::db::stream< std::pair< T1, T2 > >

Partial specialization of class template stream<T, Enable> for std::pair.

Definition at line 3534 of file dbxmap.hpp.

Public Types

- typedef std::pair< T1, T2 > **value_type**

Static Public Member Functions

- static void **get** (buffer &buf, const value_type &value)
- static void **put** (buffer &buf, value_type &value)

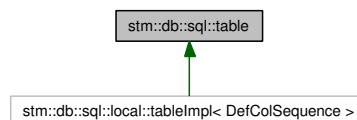
The documentation for this struct was generated from the following file:

- [dbxmap.hpp](#)

11.19 stm::db::sql::table Class Reference

```
#include <dbsql.hpp>
```

Inheritance diagram for stm::db::sql::table:



11.19.1 Detailed Description

Abstract base class representing a SQL database [table](#).

Definition at line 177 of file dbsql.hpp.

Public Member Functions

- virtual **~table** ()
Virtual destructor.
- virtual const std::string & **name** () const =0
Returns the table's name.
- virtual const layout & **columns** () const =0
Returns the table's layout.
- virtual layout & **columns** ()=0

Returns the table's layout.

- virtual `size_t size () const =0`

Returns the table's size.

- virtual `bool fixed () const =0`
- virtual `void setFixed (bool val=true)=0`
- `bool typeDefined (const std::string &name) const throw ()`
- `template<class TableTypes>`
`TableTypes::dbmap_type & getDbmap ()`
- `template<class TableTypes>`
`TableTypes::indexes & getDbindexes ()`
- `template<class TableTypes>`
`TableTypes::recnomap_type & getRecnomap ()`
- `template<class TableTypes>`
`TableTypes::recnoindexes & getRecnoindexes ()`
- `template<class TableTypes>`
`TableTypes::invrecnomap_type & getInvrecnomap ()`
- `template<class TableTypes>`
`TableTypes::invrecnoindexes & getInvrecnoindexes ()`
- `result * createResult (const std::string &columnName, bool descending=false)`
- virtual `result * createResult (size_t column, bool descending=false)=0`
- virtual `result * currentResult ()=0`
- virtual `const result * currentResult () const =0`
- `void demandSorting (const std::string &columnName)`
- virtual `void demandSorting (size_t column)=0`

Static Public Member Functions

- `template<class TableTypes>`
`static void defineType (const std::string &name) throw (std::runtime_error)`

Define a new `table` type named `name` determined by the template parameter `TableTypes`, which shall be a specialization of the class template `table::types<DefColSequence>`.

Classes

- struct `average`
- struct `average< bool >`
- struct `average< db::null_type >`
- struct `average< std::string >`
- struct `average< std::vector< byte > >`
- class `column`
- struct `defcol`
- class `descr`
- class `descrmap`
- class `field`
- class `layout`
- class `locker`
- struct `types`

11.19.2 Constructor & Destructor Documentation

11.19.2.1 stm::db::sql::table::~~table () [inline, virtual]

Virtual destructor.

Definition at line 2138 of file dbxsql.hpp.

11.19.3 Member Function Documentation

11.19.3.1 virtual const std::string& stm::db::sql::table::name () const [pure virtual]

Returns the table's name.

Referenced by stm::db::sql::connection::execsql().

11.19.3.2 virtual const layout& stm::db::sql::table::columns () const [pure virtual]

Returns the table's layout.

11.19.3.3 virtual layout& stm::db::sql::table::columns () [pure virtual]

Returns the table's layout.

This non const version is required in order to manipulate a column's format.

11.19.3.4 virtual size_t stm::db::sql::table::size () const [pure virtual]

Returns the table's size.

11.19.3.5 template<class TableTypes> void stm::db::sql::table::defineType (const std::string & name) throw (std::runtime_error) [inline, static]

Define a new [table](#) type named name determined by the template parameter TableTypes, which shall be a specialization of the class template table::types<DefColSequence>.

This can be done successfully, if name is not empty and no [table](#) type of this name has already been defined. The method throws on error. Table types have to be created on every application run anew, as they cannot be stored persistently. Intentionally there is no means to delete an existing [table](#) type during the runtime of the application.

Definition at line 2148 of file dbxsql.hpp.

References StmDebugAidsErrmsg.

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

- [dbsql.hpp](#)
- [dbxsql.hpp](#)
- [dbsql.cpp](#)

11.20 `stm::db::sql::local::tableImpl< DefColSequence >::openIndex< indexId, IndexT >` Struct Template Reference

11.20.1 Detailed Description

```
template<class DefColSequence>template<size_t indexId, typename IndexT> struct
stm::db::sql::local::tableImpl< DefColSequence >::openIndex< indexId, IndexT >
```

Helper functor for constructor which opens an index.

Definition at line 1046 of file `dbxsql.hpp`.

Public Types

- typedef `IndexT` `index_type`

Public Member Functions

- void **operator**() (`tableImpl< DefColSequence > *pTableImpl`, `std::vector< boost::function< void(tableImpl< DefColSequence > *) > >` &`indexCreator`, `index_type` &`index`, `size_type` `flags`, `layout::iterator` `cit`)

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

- [dbxsql.hpp](#)

11.21 `stm::db::sql::local::tableImpl< DefColSequence >::openIndexes< indexCnt, Indexes, IndexFlags >` Struct Template Reference

11.21.1 Detailed Description

```
template<class DefColSequence>template<size_t indexCnt, typename Indexes, typename
IndexFlags> struct stm::db::sql::local::tableImpl< DefColSequence >::openIndexes< index-
Cnt, Indexes, IndexFlags >
```

Helper functor for constructor which opens indexes.

Definition at line 985 of file `dbxsql.hpp`.

Public Member Functions

- void **operator**() (`Indexes` &`indexes`, `layout::iterator` `cit`, `std::vector< boost::function< void(tableImpl< DefColSequence > *) > >` &`indexCreator`)

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

- [dbxsql.hpp](#)

11.22 `stm::db::sql::local::tableImpl< DefColSequence >::registerResultCreator< indexId, IndexT >` Struct Template Reference

11.22.1 Detailed Description

```
template<class DefColSequence>template<size_t indexId, typename IndexT> struct
stm::db::sql::local::tableImpl< DefColSequence >::registerResultCreator< indexId, IndexT
>
```

Helper functor for constructor which registers a result creation function.

Definition at line 1242 of file `dbxsql.hpp`.

Public Types

- typedef `IndexT` `index_type`

Public Member Functions

- void `operator()` (`std::vector< boost::function< result *(tableImpl< DefColSequence > *) > >` &`resultCreator`)

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

- [dbxsql.hpp](#)

11.23 `stm::db::sql::local::tableImpl< DefColSequence >::registerResultCreators< indexCnt, Indexes >` Struct Template Reference

11.23.1 Detailed Description

```
template<class DefColSequence>template<size_t indexCnt, typename Indexes> struct
stm::db::sql::local::tableImpl< DefColSequence >::registerResultCreators< indexCnt, Indexes >
```

Helper functor for constructor which registers result creation functions.

Definition at line 1204 of file `dbxsql.hpp`.

Public Member Functions

- void `operator()` (`Indexes` &`indexes`)

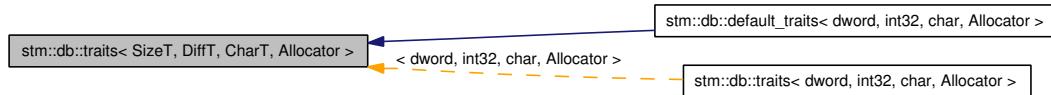
The documentation for this struct was generated from the following file:

- [dbxsql.hpp](#)

11.24 stm::db::traits< SizeT, DiffT, CharT, Allocator > Struct Template Reference

```
#include <dbmap.hpp>
```

Inheritance diagram for stm::db::traits< SizeT, DiffT, CharT, Allocator >:



11.24.1 Detailed Description

template<class SizeT, class DiffT, class CharT, class Allocator> struct stm::db::traits< SizeT, DiffT, CharT, Allocator >

Class template serving as Traits template parameter for class templates [db::map](#), [db::set](#), [db::multimap](#) and [db::multiset](#).

Definition at line 329 of file dbmap.hpp.

Public Types

- typedef SizeT **size_type**
- typedef DiffT **difference_type**
- typedef CharT **char_type**
- typedef Allocator **allocator_type**
- typedef `std::basic_string< char_type, std::char_traits< char_type >, typename allocator_type::template rebind< char_type >::other >` **string_type**

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

- [dbmap.hpp](#)

11.25 stm::db::local::xmap< ProxyT >::descr_type Struct Reference

```
#include <dbxmap.hpp>
```

11.25.1 Detailed Description

template<class ProxyT> struct stm::db::local::xmap< ProxyT >::descr_type

Type to describe the observable xmap values.

Definition at line 412 of file dbxmap.hpp.

Public Member Functions

- **descr_type** (const **string_type** &dbname=**string_type**(), flags_type dbflags=noflags, const key_compare &dbcomp=key_compare(), const allocator_type &dballoc=allocator_type())

Public Attributes

- `string_type name`
Berkeley database name.
- `size_type size`
Number of records.
- `bool endiancvt`
True, if arithmetic types shall be endian converted.
- `flags_type flags`
Database flags.
- `key_compare comp`
Comparison object.
- `allocator_type alloc`
Allocator object.

11.25.2 Member Data Documentation

11.25.2.1 `template<class ProxyT> string_type stm::db::local::xmap< ProxyT >::descr_type::name`

Berkeley database name.

Definition at line 427 of file `dbxmap.hpp`.

11.25.2.2 `template<class ProxyT> size_type stm::db::local::xmap< ProxyT >::descr_type::size`

Number of records.

Definition at line 430 of file `dbxmap.hpp`.

11.25.2.3 `template<class ProxyT> bool stm::db::local::xmap< ProxyT >::descr_type::endiancvt`

True, if arithmetic types shall be endian converted.

Definition at line 433 of file `dbxmap.hpp`.

11.25.2.4 `template<class ProxyT> flags_type stm::db::local::xmap< ProxyT >::descr_type::flags`

Database flags.

Definition at line 436 of file `dbxmap.hpp`.

11.25.2.5 `template<class ProxyT> key_compare stm::db::local::xmap< ProxyT >::descr_type::comp`

Comparison object.

Definition at line 439 of file dbxmap.hpp.

11.25.2.6 `template<class ProxyT> allocator_type stm::db::local::xmap< ProxyT >::descr_type::alloc`

Allocator object.

Definition at line 442 of file dbxmap.hpp.

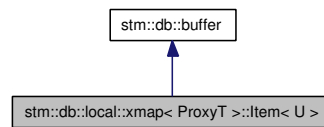
The documentation for this struct was generated from the following file:

- [dbxmap.hpp](#)

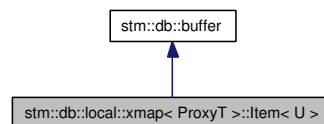
11.26 `stm::db::local::xmap< ProxyT >::Item< U >` Class Template Reference

```
#include <dbxmap.hpp>
```

Inheritance diagram for `stm::db::local::xmap< ProxyT >::Item< U >`:



Collaboration diagram for `stm::db::local::xmap< ProxyT >::Item< U >`:

**11.26.1** Detailed Description

```
template<class ProxyT>template<typename U> class stm::db::local::xmap< ProxyT >::Item< U >
```

The class template `Item<U>` serves as proxy for key and data values of type `U` to realize the access to Berkeley databases.

It implements the interface [buffer](#) and inherits from the Berkeley database class `Dbt`.

Definition at line 226 of file dbxmap.hpp.

Public Types

- typedef `boost::remove_const< U >::type` [item_type](#)

The type template parameter `U` is accessible as `item_type`.

Public Member Functions

- `Item` (const dbimpl_type *pDbImpl)
Default construction of an `Item<U>` object.
- `Item` (const dbimpl_type *pDbImpl, const Dbt *pDbt)
Construction of an `Item<U>` object serving as `buffer` to read data from a Berkeley database `Dbt` object.
- `Item` (const dbimpl_type *pDbImpl, const item_type &value)
Construction of an `Item<U>` object reflecting a value.
- `~Item` ()
The destructor deallocates any `buffer` allocated for the base class `Dbt`.
- `Item< U > & operator=` (const item_type &value)
Reflect a new value.
- void `copyTo` (Dbt &dbt)
Copy this `Item`'s content to a Berkeley `Dbt` object, such that the Berkeley database is responsible for deallocation.
- void `operator()` (item_type &value)
Updates value with this `Item<U>` object's reflected value.

Private Types

- typedef boost::detail::max_align `Align`
`Align` is a type with the maximal alignment requirements for this system.

Private Member Functions

- `size_type size` () const
- const `byte * data` () const
- `byte * data` ()
- void `resize` (`size_type` size)
Ensure that for the data `buffer` of the `Dbt` base class at least `size` bytes satisfying all alignment requirements are allocated.
- void `append` (const `byte` *src, `size_type` size)
Append `size` bytes beginning at `src` to the `buffer`.
- const `byte * extract` (`size_type` size)
Return a pointer to the next `size` bytes of the `buffer`.
- bool `endiancvt` () const
Return true, if arithmetic types have to be endian converted;.

Private Attributes

- `const dbimpl_type * pDbImpl_`
Address of this Items's primary database implementation.
- `allocator_type::template rebind< Align >::other alloc_`
Allocator object derived from the database allocator type.
- `Align * chunksvec_`
If not NULL, chunksvec_ is the array of chunks each of length sizeof(Align) allocated for this buffer's Dbt base class.
- `size_type chunkssize_`
If chunkssize_ is not 0, it is the number of bytes allocated for the chunks of length sizeof(Align) used for this buffer's Dbt base class.
- `size_type readoffs_`
Current read offset.

Friends

- class `xmap< ProxyT >`
- class `xiter`

11.26.2 Member Typedef Documentation

11.26.2.1 `template<class ProxyT> template<typename U> typedef boost::remove_const<U>::type stm::db::local::xmap< ProxyT >::Item< U >::item_type`

The type template parameter U is accessible as `item_type`.

Definition at line 234 of file `dbxmap.hpp`.

11.26.2.2 `template<class ProxyT> template<typename U> typedef boost::detail::max_align stm::db::local::xmap< ProxyT >::Align [private]`

`Align` is a type with the maximal alignment requirements for this system.

Definition at line 303 of file `dbxmap.hpp`.

11.26.3 Constructor & Destructor Documentation

11.26.3.1 `template<class ProxyT> template<typename U> stm::db::local::xmap< ProxyT >::Item< U >::Item(const dbimpl_type * pDbImpl) [inline]`

Default construction of an `Item<U>` object.

Definition at line 237 of file `dbxmap.hpp`.

11.26.3.2 `template<class ProxyT> template<typename U> stm::db::local::xmap< ProxyT >::Item< U >::Item (const dbimpl_type * pDbImpl, const Dbt * pDbt)` [inline]

Construction of an `Item<U>` object serving as `buffer` to read data from a Berkeley database `Dbt` object.

Definition at line 247 of file `dbxmap.hpp`.

11.26.3.3 `template<class ProxyT> template<typename U> stm::db::local::xmap< ProxyT >::Item< U >::Item (const dbimpl_type * pDbImpl, const item_type & value)` [inline]

Construction of an `Item<U>` object reflecting a value.

Definition at line 256 of file `dbxmap.hpp`.

11.26.3.4 `template<class ProxyT> template<typename U> stm::db::local::xmap< ProxyT >::Item< U >::~~Item ()` [inline]

The destructor deallocates any `buffer` allocated for the base class `Dbt`.

Definition at line 268 of file `dbxmap.hpp`.

References `stm::db::local::xmap< ProxyT >::Item< U >::alloc_`, `stm::db::local::xmap< ProxyT >::Item< U >::chunksize_`, and `stm::db::local::xmap< ProxyT >::Item< U >::chunksvec_`.

11.26.4 Member Function Documentation

11.26.4.1 `template<class ProxyT> template<typename U> Item<U>& stm::db::local::xmap< ProxyT >::Item< U >::operator= (const item_type & value)` [inline]

Reflect a new value.

Definition at line 271 of file `dbxmap.hpp`.

References `stm::db::local::xmap< ProxyT >::Item< U >::readoffs_`, and `stm::db::local::xmap< ProxyT >::Item< U >::resize()`.

11.26.4.2 `template<class ProxyT> template<typename U> void stm::db::local::xmap< ProxyT >::Item< U >::copyTo (Dbt & dbt)` [inline]

Copy this `Item`'s content to a Berkeley `Dbt` object, such that the Berkeley database is responsible for deallocation.

Definition at line 282 of file `dbxmap.hpp`.

References `StmDebugAidsErrmsg`.

11.26.4.3 `template<class ProxyT> template<typename U> void stm::db::local::xmap< ProxyT >::Item< U >::operator() (item_type & value)` [inline]

Updates value with this `Item<U>` object's reflected value.

Definition at line 295 of file `dbxmap.hpp`.

11.26.4.4 `template<class ProxyT> template<typename U> void stm::db::local::xmap< ProxyT >::Item< U >::resize (size_type size)` [inline, private]

Ensure that for the data `buffer` of the `Dbt` base class at least `size` bytes satisfying all alignment requirements are allocated.

On return `chunksize_` reflects the number of bytes allocated for `Align` objects.

Definition at line 321 of file `dbxmap.hpp`.

References `stm::db::local::xmap< ProxyT >::Item< U >::alloc_`, `stm::db::local::xmap< ProxyT >::Item< U >::chunksize_`, `stm::db::local::xmap< ProxyT >::Item< U >::chunksvec_`, and `StmDebugAidsVerify`.

Referenced by `stm::db::local::xmap< ProxyT >::Item< U >::append()`, and `stm::db::local::xmap< ProxyT >::Item< U >::operator=()`.

11.26.4.5 `template<class ProxyT> template<typename U> void stm::db::local::xmap< ProxyT >::Item< U >::append (const byte * src, size_type size) [inline, private]`

Append `size` bytes beginning at `src` to the `buffer`.

Definition at line 350 of file `dbxmap.hpp`.

References `stm::db::local::xmap< ProxyT >::Item< U >::resize()`.

11.26.4.6 `template<class ProxyT> template<typename U> const byte* stm::db::local::xmap< ProxyT >::Item< U >::extract (size_type size) [inline, private]`

Return a pointer to the next `size` bytes of the `buffer`.

Definition at line 359 of file `dbxmap.hpp`.

References `stm::db::local::xmap< ProxyT >::Item< U >::readoffs_`.

11.26.4.7 `template<class ProxyT> template<typename U> bool stm::db::local::xmap< ProxyT >::Item< U >::endiancvt () const [inline, private, virtual]`

Return true, if arithmetic types have to be endian converted;.

Implements `stm::db::buffer`.

Definition at line 367 of file `dbxmap.hpp`.

References `stm::db::local::xmap< ProxyT >::Item< U >::pDbImpl_`, and `StmDebugAidsVerify`.

11.26.5 Member Data Documentation

11.26.5.1 `template<class ProxyT> template<typename U> const dbimpl_type* stm::db::local::xmap< ProxyT >::Item< U >::pDbImpl_ [private]`

Address of this Items's primary database implementation.

Definition at line 374 of file `dbxmap.hpp`.

Referenced by `stm::db::local::xmap< ProxyT >::Item< U >::endiancvt()`.

11.26.5.2 `template<class ProxyT> template<typename U> allocator_type::template rebind<Align>::other stm::db::local::xmap< ProxyT >::Item< U >::alloc_ [private]`

Allocator object derived from the database allocator type.

Definition at line 377 of file `dbxmap.hpp`.

Referenced by `stm::db::local::xmap< ProxyT >::Item< U >::resize()`, and `stm::db::local::xmap< ProxyT >::Item< U >::~~Item()`.

11.26.5.3 `template<class ProxyT> template<typename U> Align* stm::db::local::xmap< ProxyT >::Item< U >::chunksvec_ [private]`

If not NULL, `chunksvec_` is the array of chunks each of length `sizeof(Align)` allocated for this buffer's Dbt base class.

Definition at line 381 of file `dbxmap.hpp`.

Referenced by `stm::db::local::xmap< ProxyT >::Item< U >::resize()`, and `stm::db::local::xmap< ProxyT >::Item< U >::~~Item()`.

11.26.5.4 `template<class ProxyT> template<typename U> size_type stm::db::local::xmap< ProxyT >::Item< U >::chunkssize_ [private]`

If `chunkssize_` is not 0, it is the number of bytes allocated for the chunks of length `sizeof(Align)` used for this buffer's Dbt base class.

Definition at line 385 of file `dbxmap.hpp`.

Referenced by `stm::db::local::xmap< ProxyT >::Item< U >::resize()`, and `stm::db::local::xmap< ProxyT >::Item< U >::~~Item()`.

11.26.5.5 `template<class ProxyT> template<typename U> size_type stm::db::local::xmap< ProxyT >::Item< U >::readoffs_ [private]`

Current read offset.

Definition at line 388 of file `dbxmap.hpp`.

Referenced by `stm::db::local::xmap< ProxyT >::Item< U >::extract()`, and `stm::db::local::xmap< ProxyT >::Item< U >::operator=()`.

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

- [dbxmap.hpp](#)

11.27 `stm::db::local::xmap< ProxyT >::secDb` Struct Reference

```
#include <dbxmap.hpp>
```

11.27.1 Detailed Description

`template<class ProxyT> struct stm::db::local::xmap< ProxyT >::secDb`

Type of secondary database.

Definition at line 2550 of file `dbxmap.hpp`.

Public Attributes

- `indexDb * pIndexDb`
- `indexDescr * pIndexDescr`

11.28 `stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >` Class Template Reference70

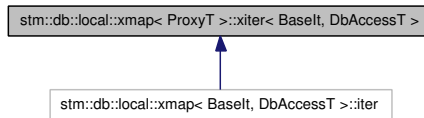
The documentation for this struct was generated from the following file:

- [dbxmap.hpp](#)

11.28 `stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >` Class Template Reference

```
#include <dbxmap.hpp>
```

Inheritance diagram for `stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >`:



11.28.1 Detailed Description

`template<class ProxyT>template<class BaseIt, class DbAccessT> class stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >`

The class template `xiter<BaseIt, DbAccessT>` implements class template `iter<BaseIt, DbAccessT>`, which in turn is the base of all `db::map`, `db::map::index`, `db::set`, `db::multimap` and `db::multiset` iterators.

Its only purpose is to hide the implementation from its users. To do this like this is necessary, because it is not possible to grant friendship to nested types of nested class templates, especially to the classes `index<SKey, SCompare, Spolicy>::iterator`. So the members of class template `xiter<BaseIt, DbAccessT>` have to be declared public and class template `iter<BaseIt, DbAccessT>` which declares the public interface of the iterators inherits protected from `xiter<BaseIt, DbAccessT>`.

Definition at line 1384 of file `dbxmap.hpp`.

Public Types

- enum {
 LruMaxSize = 1024,
 CheckConsistency = 1 }
• typedef `xiter< BaseIt, DbAccessT >` **iter_type**
• typedef `BaseIt` **base_type**
• typedef `DbAccessT` **dbaccess_type**
• typedef `base_type::is_const` **is_const**
• typedef `dbaccess_type::is_index` **is_index**
• typedef `dbaccess_type::key_type` **key_type**

Type of primary keys for database iterators and type of secondary keys for index iterators.

Public Member Functions

- `xiter()`
Construct an uninitialized iterator.

11.28 `stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >` Class Template Reference 71

- `xiter` (const typename dbaccess_type::pointer_type pointer)
Construct the past the end iterator.
- `xiter` (const base_type &baseit, const typename dbaccess_type::pointer_type pointer, Dbc *pDbc)
Construct an iterator pointing to the same location as baseit.
- `xiter` & `assign` (const base_type &baseit, const typename dbaccess_type::pointer_type pointer, Dbc *pDbc=NULL)
- void `updateDirty` (boost::mpl::false_)
- void `updateDirty` (boost::mpl::true_)
- const `key_type` & `getKey` (boost::mpl::false_) const
- const `key_type` & `getKey` (boost::mpl::true_) const
- reference `getRef` (boost::mpl::false_)
- reference `getRef` (boost::mpl::true_)
- void `assertValid` (bool tolerateInvalid=false) const
- void `swap` (`iter_type` &rhs) throw ()
- `size_type` `count` () const
- void `unlink` (bool toErase=false)
Decrement the reference count of the iterator's key and insert it into the last recently used key list, if it has become 0 and the iterator's record is not about to be deleted.
- void `locate` (`size_type` posFlag, const `key_type` &refKey=`key_type`())
- bool `locateCursor` (const `key_type` &key, `size_type` posFlag, boost::mpl::false_)
- bool `locateCursor` (const `key_type` &key, `size_type` posFlag, boost::mpl::true_)
- bool `isInitialized` () const
- bool `isEnd` () const
- void `setEnd` (boost::mpl::false_)
- void `setEnd` (boost::mpl::true_)
- void `setEnd` ()
- bool `checkCursor` (boost::mpl::false_) const
- bool `checkCursor` (boost::mpl::true_) const
- bool `check` (bool tolerateInvalid=false) const
Returns true, if this iterator is the past the end iterator, or if the key of the record referenced by its cursor is the same as the iterator's key.

Public Attributes

- base_type `current_`
The base iterator of this `xiter`.
- dbaccess_type `dbAccess_`
If `dbaccess_type::operator pointer_type ()` yields NULL, this iterator is uninitialized.
- Dbc * `pDbc_`
If not NULL, `pDbc_` is a pointer to a Berkeley database cursor referring to this iterator's database record.

Friends

- class `iter`

11.28.2 Member Typedef Documentation

11.28.2.1 `template<class ProxyT> template<class BaseIt, class DbAccessT> typedef dbaccess_ - type::key_type stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >::key_type`

Type of primary keys for database iterators and type of secondary keys for index iterators.

Definition at line 1407 of file `dbxmap.hpp`.

11.28.3 Constructor & Destructor Documentation

11.28.3.1 `template<class ProxyT> template<class BaseIt, class DbAccessT> stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >::xiter () [inline, explicit]`

Construct an uninitialized iterator.

Definition at line 1410 of file `dbxmap.hpp`.

11.28.3.2 `template<class ProxyT> template<class BaseIt, class DbAccessT> stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >::xiter (const typename dbaccess_ - type::pointer_type pointer) [inline, explicit]`

Construct the past the end iterator.

Definition at line 1419 of file `dbxmap.hpp`.

References `stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >::dbAccess_`, and `StmDebugAidsVerify`.

11.28.3.3 `template<class ProxyT> template<class BaseIt, class DbAccessT> stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >::xiter (const base_type & baseit, const typename dbaccess_type::pointer_type pointer, Dbc *pDbc) [inline]`

Construct an iterator pointing to the same location as `baseit`.

Definition at line 1431 of file `dbxmap.hpp`.

References `stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >::current_`, `stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >::dbAccess_`, `stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >::pDbc_`, and `StmDebugAidsErrMsg`.

11.28.4 Member Function Documentation

11.28.4.1 `template<class ProxyT> template<class BaseIt, class DbAccessT> void stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >::unlink (bool toErase = false) [inline]`

Decrement the reference count of the iterator's key and insert it into the last recently used key list, if it has become 0 and the iterator's record is not about to be deleted.

Definition at line 1574 of file `dbxmap.hpp`.

11.28 `stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >` Class Template Reference 73

References `stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >::current_`, `stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >::dbAccess_`, `put()`, `StmDebugAidsErrmsg`, and `StmDebugAidsVerify`.

11.28.4.2 `template<class ProxyT> template<class BaseIt, class DbAccessT> bool stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >::check (bool tolerateInvalid = false) const` [inline]

Returns true, if this iterator is the past the end iterator, or if the key of the record referenced by its cursor is the same as the iterator's key.

Definition at line 1880 of file `dbxmap.hpp`.

References `stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >::current_`.

11.28.5 Member Data Documentation

11.28.5.1 `template<class ProxyT> template<class BaseIt, class DbAccessT> base_type stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >::current_`

The base iterator of this [xiter](#).

Definition at line 1889 of file `dbxmap.hpp`.

Referenced by `stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >::check()`, `stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >::unlink()`, and `stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >::xiter()`.

11.28.5.2 `template<class ProxyT> template<class BaseIt, class DbAccessT> dbaccess_type stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >::dbAccess_`

If `dbaccess_type::operator pointer_type ()` yields NULL, this iterator is uninitialized.

Else this operator returns the address of the constant `dbimpl_type` object of the iterator's primary database implementation (database iterator) or of the constant `index_type` object of the iterator's database index implementation (index iterator). Then the operator `dbaccess_type::operator()` yields the address of the `dbimpl_type` object of the iterator's primary database implementation and via the operator `dbaccess_type::operator → ()` `dbAccess_` is a smart pointer to the iterator's `dbimpl_type` object.

Definition at line 1900 of file `dbxmap.hpp`.

Referenced by `stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >::unlink()`, and `stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >::xiter()`.

11.28.5.3 `template<class ProxyT> template<class BaseIt, class DbAccessT> Dbc* stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >::pDbc_`

If not NULL, `pDbc_` is a pointer to a Berkeley database cursor referring to this iterator's database record.

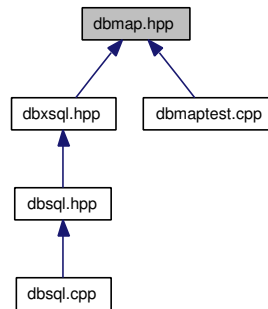
That means that if `pDbc_` is not NULL, the iterator is not uninitialized. If it is NULL the iterator is either uninitialized (`dbaccess_type::pointer_type (dbAccess_) == NULL`) or represents a non-existing database record expressed as past the end iterator of the cache.

Definition at line 1908 of file `dbxmap.hpp`.

Referenced by `stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >::xiter()`.

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

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **stm**
- namespace [stm::db](#)

Classes

- struct [stm::db::policy< FlagsT, Flags >](#)
Class template serving as Policy template parameter for class templates [db::map](#), [db::set](#), [db::multimap](#) and [db::multiset](#).
- struct [stm::db::default_policy< Flags >](#)
Default [policy](#) template parameter for class templates [db::map](#), [db::set](#), [db::multimap](#) and [db::multiset](#).
- struct [stm::db::traits< SizeT, DiffT, CharT, Allocator >](#)
Class template serving as Traits template parameter for class templates [db::map](#), [db::set](#), [db::multimap](#) and [db::multiset](#).
- struct [stm::db::default_traits< Allocator >](#)
Default [traits](#) template parameter for class templates [db::map](#), [db::set](#), [db::multimap](#) and [db::multiset](#).
- struct [stm::db::null_type](#)
Empty type with default constructor doing nothing.
- struct [stm::db::optional< T >](#)
*The class template [optional<T>](#) is a [boost::variant](#) type consisting of a [db::null_type](#) member, if the instance holds no *T* value and a *T* member, if it holds a *T* value.*
- class [stm::db::buffer](#)
The interface class [buffer](#) serves as a [buffer](#) abstraction for the conversion of application key and data objects into their byte serialized form needed to store them in Berkeley databases and vice versa.
- struct [stm::db::stream< T, Enable >](#)
*The class template [stream<T>](#) on the first hand serves as [stream](#) type to append values of type *T* in a byte serialized form to the buffers used in the access implementation of Berkeley databases and on the other hand to extract values of type *T* from those buffers.*

- class `stm::db::map< Key, T, Compare, Allocator, Policy, Traits >`
The `db::map` class template is a kind of associative container that supports unique keys (contains at most one of each key value) and provides for fast retrieval of values of another type `T` based on the keys.
- class `stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::locker`
xxx
- class `stm::db::map< Key, T, Compare, Allocator, Policy, Traits >::index< SKey, SCompare >`
Member class template `db::map::index`.
- class `stm::db::set< Key, Compare, Allocator, Policy, Traits >`
The `db::set` class template is a kind of associative container that supports unique keys (contains at most one of each key value) and provides for fast retrieval based on these keys.
- class `stm::db::set< Key, Compare, Allocator, Policy, Traits >::locker`
xxx
- class `stm::db::multimap< Key, T, Compare, Allocator, Policy, Traits >`
Class template `db::multimap`.
- class `stm::db::multimap< Key, T, Compare, Allocator, Policy, Traits >::locker`
Member class `db::multimap::locker`.
- class `stm::db::multiset< Key, Compare, Allocator, Policy, Traits >`
Class template `db::multiset`.
- class `stm::db::multiset< Key, Compare, Allocator, Policy, Traits >::locker`
Member class `db::multiset::locker`.

Typedefs

- typedef `dword` `stm::db::size_type`
For now the Berkeley database engine supports only 32 bit sizes.
- typedef `int32` `stm::db::difference_type`
For now the Berkeley database engine supports only 32 bit sizes.
- typedef `char` `stm::db::char_type`
For now only plain char characters are supported.

Functions

- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator== (const map< Key, T, Compare, Allocator, Policy, Traits > &x, const map< Key, T, Compare, Allocator, Policy, Traits > &y)`
- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
`bool stm::db::operator< (const map< Key, T, Compare, Allocator, Policy, Traits > &x, const map< Key, T, Compare, Allocator, Policy, Traits > &y)`

- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
`void stm::db::swap` (multiset< Key, Compare, Allocator, Policy, Traits > &x, multiset< Key, Compare, Allocator, Policy, Traits > &y)

Variables

- `const size_type stm::db::locking = 1`
Policy flags (bitwise orable).

12.2 dbmaptest.cpp File Reference

12.2.1 Detailed Description

Implementation of a console program demonstrating the usage and testing the implementation of the SysToMath DbMap C++ Library `stmdbmap`.

Version:

1.04-r364

Date:

2008-01-05 11:01:52 (Tom)

Author:

Tom Michaelis
SysToMath
Wittelsbacherstr. 7
D-80469 Munich

Contact:

<http://www.SysToMath.com>
<mailto:Tom.Michaelis@SysToMath.com>

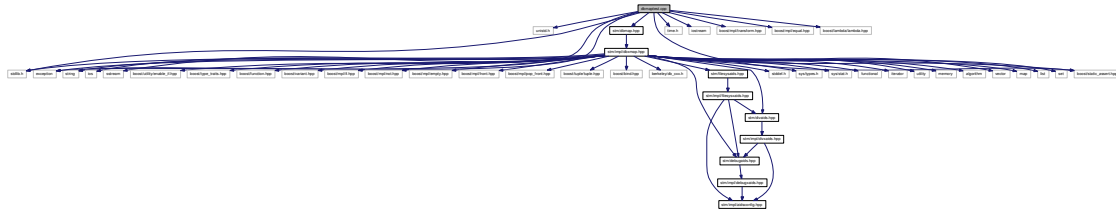
This C++ program file contains the implementation of the console program `dbmaptest`.

Definition in file [dbmaptest.cpp](#).

```
#include <unistd.h>
#include <stdlib.h>
#include <time.h>
#include <exception>
#include <string>
#include <iostream>
#include <stm/dbmap.hpp>
#include <boost/mpl/transform.hpp>
#include <boost/mpl/equal.hpp>
#include <boost/static_assert.hpp>
```

```
#include <boost/lambda/lambda.hpp>
```

Include dependency graph for dbmaptest.cpp:



Namespaces

- namespace **stm**
- namespace **stm::db**

Classes

- struct **Data**
- struct **stm::db::stream**< **Data** >

Defines

- #define **_MAX_PATH** 1024

Typedefs

- typedef **stm::db::map**< unsigned int, **Data** > **DataMap**
- typedef **DataMap::index**< std::string > **CompIndex**
- typedef **DataMap::index**< unsigned int > **SalIndex**

Functions

- static void **test** ()
- static bool **getCompany** (const **DataMap::value_type** &value, **CompIndex::key_type** &skkey)
- static bool **getSalary** (const **DataMap::value_type** &value, **SalIndex::key_type** &skkey)
- int **main** (int argc, char *argv[])

Variables

- const char * **Copyright** = "(C) 2003-2008 Tom Michaelis, SysToMath"
- const char * **Version** = "1.04-r364"

12.3 dbsql.cpp File Reference

12.3.1 Detailed Description

Implementation of the SQL interface of named persistent associative containers.

Version:

1.04-r6

Date:

2008-01-01 13:13:47 (tom)

Author:

Tom Michaelis
SysToMath
Wittelsbacherstr. 7
D-80469 Munich

Contact:

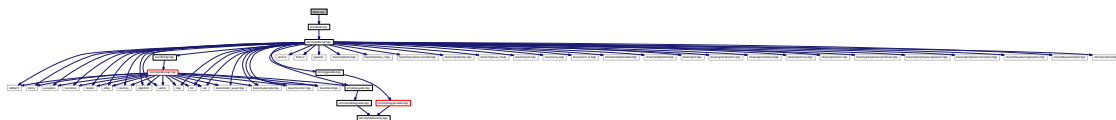
<http://www.SysToMath.com>
<mailto:Tom.Michaelis@SysToMath.com>

This C++ program file contains definitions implementing the rudimentary SQL interface of named persistent associative containers. They are declared in the C++ header file [dbsql.hpp](#) belonging to the SysToMath DbMap C++ Library `stmdbsql`.

Definition in file [dbsql.cpp](#).

```
#include <stm/dbsql.hpp>
```

Include dependency graph for `dbsql.cpp`:

**Namespaces**

- namespace `stm`
- namespace `stm::db`
- namespace `stm::db::sql`
- namespace `stm::db::sql::local`

12.4 dbsql.hpp File Reference

12.4.1 Detailed Description

Declarations for the SQL interface of named persistent associative containers with unique and equivalent keys.

Version:

1.04-r6

Date:

2008-01-01 13:13:47 (tom)

Author:

Tom Michaelis
SysToMath
Wittelsbacherstr. 7
D-80469 Munich

Contact:

<http://www.SysToMath.com>
<mailto:Tom.Michaelis@SysToMath.com>

This C++ header file contains declarations of functions, function templates, classes and class templates for the SQL interface of named persistent associative containers. They are implemented in the C++ inline file [dbxsql.hpp](#) and the C++ program file [dbsql.cpp](#) belonging to the SysToMath DB C++ Library `stmdbsql`.

The declarations of this header file are all contained in the namespace `stm::db::sql`.

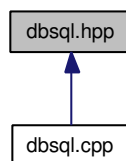
Definition in file [dbsql.hpp](#).

```
#include <stm/impl/dbxsql.hpp>
```

Include dependency graph for `dbsql.hpp`:



This graph shows which files directly or indirectly include this file:

**Namespaces**

- namespace `stm`
- namespace `stm::db`
- namespace `stm::db::sql`

Classes

- struct `stm::db::sql::noconversion< T >`
- class `stm::db::sql::table`

Abstract base class representing a SQL database [table](#).

- class `stm::db::sql::table::field`
- struct `stm::db::sql::table::field::format`
- struct `stm::db::sql::table::average< T >`
- struct `stm::db::sql::table::average< db::null_type >`
- struct `stm::db::sql::table::average< bool >`
- struct `stm::db::sql::table::average< std::string >`
- struct `stm::db::sql::table::average< std::vector< byte > >`
- struct `stm::db::sql::table::defcol< T, Flags, Convert >`
- struct `stm::db::sql::table::types< DefColSequence >`
- class `stm::db::sql::table::column`
- class `stm::db::sql::table::layout`
- class `stm::db::sql::table::descr`
- class `stm::db::sql::table::descmap`
- class `stm::db::sql::table::locker`
- class `stm::db::sql::result`
- class `stm::db::sql::connection`

Class representing a SQL database [connection](#).

Variables

- const size_type `stm::db::sql::delayedsort` = `local::DelayedSortFlag`
Flag value for class template `table::defcol` indicating a sortable column whose index shall be constructed delayed, if applicable.
- const size_type `stm::db::sql::recnosort` = `local::DelayedSortFlag` | `local::RecnoFlag`
Flag value for class template `table::defcol` indicating a sortable column whose index shall be constructed delayed and support fast record based positioning, if applicable.
- const size_type `stm::db::sql::notsortable` = `local::NotSortableValue`
Flag value for class template `table::defcol` overwrites all other flags and indicates a not sortable column.

12.5 dbxmap.hpp File Reference

12.5.1 Detailed Description

Implementation of named persistent associative containers.

Version:

1.04-r9

Date:

2008-01-03 12:59:48 (Tom)

Author:

Tom Michaelis
SysToMath
Wittelsbacherstr. 7
D-80469 Munich

Contact:

<http://www.SysToMath.com>
<mailto:Tom.Michaelis@SysToMath.com>

This C++ inline file contains inline definitions and declarations implementing named persistent associative containers. They are declared in the C++ header file [dbmap.hpp](#) belonging to the SysToMath DbMap C++ Library `stmdbmap`.

Definition in file [dbxmap.hpp](#).

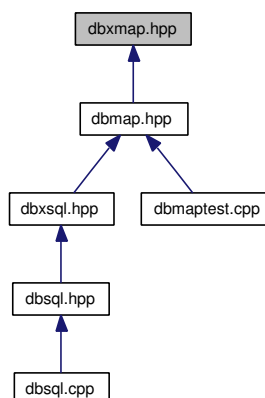
```
#include <stddef.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <string>
#include <exception>
#include <functional>
#include <iterator>
#include <utility>
#include <memory>
#include <algorithm>
#include <vector>
#include <map>
#include <list>
#include <set>
#include <ios>
#include <sstream>
#include <boost/static_assert.hpp>
#include <boost/utility/enable_if.hpp>
#include <boost/type_traits.hpp>
#include <boost/function.hpp>
#include <boost/variant.hpp>
#include <boost/mpl/if.hpp>
#include <boost/mpl/not.hpp>
#include <boost/mpl/empty.hpp>
#include <boost/mpl/front.hpp>
```

```
#include <boost/mpl/pop_front.hpp>
#include <boost/tuple/tuple.hpp>
#include <boost/bind.hpp>
#include <berkeley/db_cxx.h>
#include <stm/filesysaids.hpp>
#include <stm/debugaids.hpp>
#include <stm/divaids.hpp>
```

Include dependency graph for dbxmap.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **stm**
- namespace **stm::db**
- namespace **stm::db::local**

Classes

- class **stm::db::local::xmap**< **ProxyT** >
- class **stm::db::local::xmap**< **ProxyT** >::Item< **U** >

The class template Item<U> serves as proxy for key and data values of type U to realize the access to Berkeley databases.

- class **stm::db::local::xmap**< **ProxyT** >::value_compare
- struct **stm::db::local::xmap**< **ProxyT** >::descr_type

Type to describe the observable xmap values.

- struct `stm::db::local::xmap< ProxyT >::cacheelem_type`
- class `stm::db::local::xmap< ProxyT >::cache_type`
- class `stm::db::local::xmap< ProxyT >::DbAccess`
- class `stm::db::local::xmap< ProxyT >::DatabaseIt< constness >`
- class `stm::db::local::xmap< ProxyT >::indexDb`
- struct `stm::db::local::xmap< ProxyT >::indexDescr`
- class `stm::db::local::xmap< ProxyT >::index< SKey, SCompare >`
- struct `stm::db::local::xmap< ProxyT >::index< SKey, SCompare >::descr_type`
- class `stm::db::local::xmap< ProxyT >::index< SKey, SCompare >::DbAccess`
- class `stm::db::local::xmap< ProxyT >::index< SKey, SCompare >::IndexIt< constness >`
- class `stm::db::local::xmap< ProxyT >::index< SKey, SCompare >::iterator`
- class `stm::db::local::xmap< ProxyT >::index< SKey, SCompare >::const_iterator`
- class `stm::db::local::xmap< ProxyT >::xiter< BaseIt, DbAccessT >`

The class template `xiter<BaseIt, DbAccessT>` implements class template `iter<BaseIt, DbAccessT>`, which in turn is the base of all `db::map`, `db::map::index`, `db::set`, `db::multimap` and `db::multiset` iterators.

- class `stm::db::local::xmap< ProxyT >::riter< BaseIt >`
- class `stm::db::local::xmap< ProxyT >::iter< BaseIt, DbAccessT >`
- class `stm::db::local::xmap< ProxyT >::iterator`
- class `stm::db::local::xmap< ProxyT >::const_iterator`
- struct `stm::db::local::xmap< ProxyT >::secDb`

Type of secondary database.

- struct `stm::db::local::xmap< ProxyT >::secDbMap`
- class `stm::db::local::pxmap< Key, T, Compare, Allocator, Policy, Traits, multi >`
- class `stm::db::local::plocker< Key, T, Compare, Allocator, Policy, Traits, multi, active >`
- class `stm::db::local::plocker< Key, T, Compare, Allocator, Policy, Traits, multi, boost::mpl::true_ >`
- class `stm::db::local::pindex< Key, T, Compare, Allocator, Policy, Traits, SKey, SCompare >`
- struct `stm::db::is_variant< T >`
- struct `stm::db::is_variant< boost::variant< BOOST_VARIANT_ENUM_PARAMS(T)> >`
- struct `stm::db::is_optional< T >`
- struct `stm::db::is_optional< optional< T > >`
- struct `stm::db::is_bytelike< T >`
- struct `stm::db::may_endianvt< T, Enable >`
- struct `stm::db::may_endianvt< T, typename boost::enable_if< boost::mpl::and_< boost::is_arithmetic< T >, boost::mpl::not_< boost::mpl::bool_< is_bytelike< T >::value > > >::type >`
- struct `stm::db::is_conslist< T, Enable >`
- struct `stm::db::is_conslist< boost::tuples::null_type >`
- struct `stm::db::is_conslist< boost::tuples::cons< HT, TT > >`
- struct `stm::db::is_conslist< conslist< Sequence > >`
- struct `stm::db::is_conslist< T, typename boost::enable_if< boost::is_same< boost::tuples::cons< typename T::head_type, typename T::tail_type >, typename T::inherited > >::type >`
- struct `stm::db::conslist< Sequence, typename boost::enable_if< boost::mpl::and_< boost::mpl::is_sequence< Sequence >, boost::mpl::empty< Sequence > > >::type >`

- struct `stm::db::conslist`< Sequence, typename boost::enable_if< boost::mpl::and_< boost::mpl::is_sequence< Sequence >, boost::mpl::not_< boost::mpl::empty< Sequence >>>>::type >
- struct `stm::db::stream`< T, typename boost::enable_if< boost::is_empty< T >>::type >
- struct `stm::db::stream`< T, typename boost::enable_if< boost::is_pod< T >>::type >
- struct `stm::db::stream`< std::vector< T, Allocator > >
- struct `stm::db::stream`< std::basic_string< charT, traits, Allocator > >
- struct `stm::db::stream`< T, typename boost::enable_if< is_variant< T >>::type >
- struct `stm::db::stream`< T, typename boost::enable_if< is_optional< T >>::type >
- struct `stm::db::stream`< T, typename boost::enable_if< is_conslist< T >>::type >
- struct `stm::db::stream`< std::pair< T1, T2 > >

Partial specialization of class template stream<T, Enable> for std::pair.

Functions

- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>
bool `stm::db::operator==` (const map< Key, T, Compare, Allocator, Policy, Traits > &x, const map< Key, T, Compare, Allocator, Policy, Traits > &y)
- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>
bool `stm::db::operator<` (const map< Key, T, Compare, Allocator, Policy, Traits > &x, const map< Key, T, Compare, Allocator, Policy, Traits > &y)
- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>
bool `stm::db::operator!=` (const map< Key, T, Compare, Allocator, Policy, Traits > &x, const map< Key, T, Compare, Allocator, Policy, Traits > &y)
- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>
bool `stm::db::operator>` (const map< Key, T, Compare, Allocator, Policy, Traits > &x, const map< Key, T, Compare, Allocator, Policy, Traits > &y)
- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>
bool `stm::db::operator>=` (const map< Key, T, Compare, Allocator, Policy, Traits > &x, const map< Key, T, Compare, Allocator, Policy, Traits > &y)
- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>
bool `stm::db::operator<=` (const map< Key, T, Compare, Allocator, Policy, Traits > &x, const map< Key, T, Compare, Allocator, Policy, Traits > &y)
- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits >
void `stm::db::swap` (map< Key, T, Compare, Allocator, Policy, Traits > &x, map< Key, T, Compare, Allocator, Policy, Traits > &y)
- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits, class SKey, class SCompare>
bool `stm::db::operator==` (const typename map< Key, T, Compare, Allocator, Policy, Traits >::template index< SKey, SCompare > &x, const typename map< Key, T, Compare, Allocator, Policy, Traits >::template index< SKey, SCompare > &y)
- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits, class SKey, class SCompare>
bool `stm::db::operator<` (const typename map< Key, T, Compare, Allocator, Policy, Traits >::template index< SKey, SCompare > &x, const typename map< Key, T, Compare, Allocator, Policy, Traits >::template index< SKey, SCompare > &y)
- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits, class SKey, class SCompare>
bool `stm::db::operator!=` (const typename map< Key, T, Compare, Allocator, Policy, Traits >::template index< SKey, SCompare > &x, const typename map< Key, T, Compare, Allocator, Policy, Traits >::template index< SKey, SCompare > &y)
- template<class Key, class T, class Compare, class Allocator, class Policy, class Traits, class SKey, class SCompare>
bool `stm::db::operator>` (const typename map< Key, T, Compare, Allocator, Policy, Traits >::template index< SKey, SCompare > &x, const typename map< Key, T, Compare, Allocator, Policy, Traits >::template index< SKey, SCompare > &y)

- `template<class Key, class T, class Compare, class Allocator, class Policy, class Traits>`
void **stm::db::swap** (multimap< Key, T, Compare, Allocator, Policy, Traits > &x, multimap< Key, T, Compare, Allocator, Policy, Traits > &y)
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
bool **stm::db::operator==** (const multiset< Key, Compare, Allocator, Policy, Traits > &x, const multiset< Key, Compare, Allocator, Policy, Traits > &y)
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
bool **stm::db::operator<** (const multiset< Key, Compare, Allocator, Policy, Traits > &x, const multiset< Key, Compare, Allocator, Policy, Traits > &y)
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
bool **stm::db::operator!=** (const multiset< Key, Compare, Allocator, Policy, Traits > &x, const multiset< Key, Compare, Allocator, Policy, Traits > &y)
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
bool **stm::db::operator>** (const multiset< Key, Compare, Allocator, Policy, Traits > &x, const multiset< Key, Compare, Allocator, Policy, Traits > &y)
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
bool **stm::db::operator>=** (const multiset< Key, Compare, Allocator, Policy, Traits > &x, const multiset< Key, Compare, Allocator, Policy, Traits > &y)
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
bool **stm::db::operator<=** (const multiset< Key, Compare, Allocator, Policy, Traits > &x, const multiset< Key, Compare, Allocator, Policy, Traits > &y)
- `template<class Key, class Compare, class Allocator, class Policy, class Traits>`
void **stm::db::swap** (multiset< Key, Compare, Allocator, Policy, Traits > &x, multiset< Key, Compare, Allocator, Policy, Traits > &y)

12.6 dbxsql.hpp File Reference

12.6.1 Detailed Description

Implementation of the SQL interface of named persistent associative containers.

Version:

1.04-r364

Date:

2008-01-05 11:01:51 (Tom)

Author:

Tom Michaelis
SysToMath
Wittelsbacherstr. 7
D-80469 Munich

Contact:

<http://www.SysToMath.com>
<mailto:Tom.Michaelis@SysToMath.com>

This C++ inline file contains inline definitions and declarations implementing the rudimentary SQL interface of named persistent associative containers. They are declared in the C++ header file [dbxsql.hpp](#) belonging to the SysToMath DbMap C++ Library `stmdbxsql`.

Definition in file [dbxsql.hpp](#).

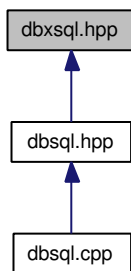
```
#include <stddef.h>
#include <errno.h>
#include <limits.h>
#include <string>
#include <exception>
#include <typeinfo>
#include <functional>
#include <iterator>
#include <utility>
#include <memory>
#include <algorithm>
#include <vector>
#include <map>
#include <list>
#include <set>
#include <boost/static_assert.hpp>
#include <boost/mpl/size.hpp>
#include <boost/tuple/tuple.hpp>
#include <boost/mpl/size_t.hpp>
#include <boost/mpl/vector/vector20.hpp>
#include <boost/mpl/identity.hpp>
#include <boost/mpl/eval_if.hpp>
#include <boost/mpl/at.hpp>
#include <boost/array.hpp>
#include <boost/function.hpp>
#include <boost/mem_fn.hpp>
#include <boost/bind.hpp>
#include <boost/lambda/lambda.hpp>
#include <boost/lambda/bind.hpp>
#include <boost/spirit.hpp>
#include <boost/spirit/dynamic.hpp>
#include <boost/spirit/attribute.hpp>
#include <boost/spirit/core.hpp>
#include <boost/spirit/actor.hpp>
#include <boost/spirit/phoenix/primitives.hpp>
```

```
#include <boost/spirit/phoenix/operators.hpp>
#include <boost/spirit/phoenix/functions.hpp>
#include <boost/filesystem/operations.hpp>
#include <boost/filesystem/path.hpp>
#include <stm/debugaids.hpp>
#include <stm/loggeraids.hpp>
#include <stm/dbmap.hpp>
#include <stm/internal/config.h>
```

Include dependency graph for dbxsql.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **stm**
- namespace **stm::db**
- namespace **stm::db::sql**
- namespace **stm::db::sql::local**
- namespace **stm::db::sql::select**

Classes

- class **stm::db::sql::local::xtable**
- class **stm::db::sql::local::xtypes< DefColSequence >**
- struct **stm::db::sql::local::xtypes< DefColSequence >::comp_type< Convert >**
- struct **stm::db::sql::local::xtypes< DefColSequence >::comp_type< noconversion< T > >**
- struct **stm::db::sql::local::xtypes< DefColSequence >::indexcomp_type< Convert >**
- struct **stm::db::sql::local::xtypes< DefColSequence >::indexcomp_type< noconversion< T > >**
- struct **stm::db::sql::local::xtypes< DefColSequence >::get_flags_type< Functor >**
- struct **stm::db::sql::local::xtypes< DefColSequence >::get_value_type< Functor >**
- struct **stm::db::sql::local::xtypes< DefColSequence >::make_index_type< IndexColDef >**

- struct `stm::db::sql::local::xtypes< DefColSequence >::make_index_type< table::defcol< T, notsortable, Comp > >`
- struct `stm::db::sql::local::xtypes< DefColSequence >::make_recnoindex_type< IndexColDef >`
- struct `stm::db::sql::local::xtypes< DefColSequence >::make_recnoindex_type< table::defcol< T, notsortable, Comp > >`
- struct `stm::db::sql::local::xtypes< DefColSequence >::make_invrecnoindex_type< IndexColDef >`
- struct `stm::db::sql::local::xtypes< DefColSequence >::make_invrecnoindex_type< table::defcol< T, notsortable, Comp > >`
- class `stm::db::sql::local::xcolumn`
- class `stm::db::sql::local::xdescr`
- class `stm::db::sql::local::xlocker`
- class `stm::db::sql::local::xconnection`
- class `stm::db::sql::local::tableImpl< DefColSequence >`
- struct `stm::db::sql::local::tableImpl< DefColSequence >::openIndexes< indexCnt, Indexes, IndexFlags >`
Helper functor for constructor which opens indexes.
- struct `stm::db::sql::local::tableImpl< DefColSequence >::openIndexes< indexCnt, boost::tuples::null_type, boost::tuples::null_type >`
- struct `stm::db::sql::local::tableImpl< DefColSequence >::openIndex< indexId, IndexT >`
Helper functor for constructor which opens an index.
- struct `stm::db::sql::local::tableImpl< DefColSequence >::openIndex< indexId, db::null_type >`
- struct `stm::db::sql::local::tableImpl< DefColSequence >::registerResultCreators< indexCnt, Indexes >`
Helper functor for constructor which registers result creation functions.
- struct `stm::db::sql::local::tableImpl< DefColSequence >::registerResultCreators< indexCnt, boost::tuples::null_type >`
- struct `stm::db::sql::local::tableImpl< DefColSequence >::registerResultCreator< indexId, IndexT >`
Helper functor for constructor which registers a result creation function.
- struct `stm::db::sql::local::tableImpl< DefColSequence >::registerResultCreator< indexId, db::null_type >`
- struct `stm::db::sql::local::updateResult< ValueT, col >`
- struct `stm::db::sql::local::updateResult< ValueT, 0 >`
- class `stm::db::sql::local::resultImpl< TableImplT, sortColNr, descending >`
- struct `stm::db::stream< db::null_type >`
- struct `stm::db::stream< sql::table::column >`
- struct `stm::db::stream< sql::table::layout >`
- struct `stm::db::stream< sql::table::descr >`
- struct `stm::db::sql::select::type`
- struct `stm::db::sql::select::type::setmember_impl< MemberType, memPtr >`
- struct `stm::db::sql::select::type::setmember_impl< MemberType, memPtr >::result< Struct, MemberT >`
- struct `stm::db::sql::select::type::getmember_impl< MemberType, memPtr >`

- struct `stm::db::sql::select::type::getmember_impl< MemberType, memPtr >::result< Struct, MemberT >`
- struct `stm::db::sql::select::push_back_impl`
- struct `stm::db::sql::select::push_back_impl::result< Container, Item >`
- struct `stm::db::sql::select::env`
- struct `stm::db::sql::select::syntax`
- struct `stm::db::sql::select::syntax::definition< ScannerT >`

Defines

- `#define STM_LIB_NAME "stmdbsql"`
- `#define STM_DBXSQL_HPP`

Enumerations

- enum {
FlagBits = sizeof (size_type) * CHAR_BIT,
MSFlagBit = 1 << (FlagBits - 1),
NotSortableValue = ~MSFlagBit,
DelayedSortFlag = MSFlagBit >> 1,
RecnoFlag = DelayedSortFlag >> 1 }

Functions

- template<typename Skeys>
static void `stm::db::sql::local::checkSecondaryKeys` (table::layout::const_iterator cit)
Helper for [table](#) layout consistency check.
- template<>
static void `stm::db::sql::local::checkSecondaryKeys< boost::tuples::null_type >`
(table::layout::const_iterator)

Variables

- `std::map< std::string, boost::function< table *(const std::string &, table::layout &, db::size_type)>>` **stm::db::sql::local::tableTypes**
- const phoenix::function< type::setmember_impl< std::string,&type::tablename >>
stm::db::sql::select::setTablename = type::setmember_impl<std::string, &type::tablename>
()
- const phoenix::function< type::setmember_impl< std::string,&type::indexname >>
stm::db::sql::select::setIndexname = type::setmember_impl<std::string, &type::indexname> ()
- const phoenix::function< type::setmember_impl< bool,&type::descending >>
stm::db::sql::select::setDescending = type::setmember_impl<bool, &type::descending> ()
- const phoenix::function< type::setmember_impl< bool,&type::all >> **stm::db::sql::select::setAll**
= type::setmember_impl<bool, &type::all> ()
- const phoenix::function< type::getmember_impl< std::string,&type::tablename >>
stm::db::sql::select::getTablename = type::getmember_impl<std::string, &type::tablename> ()

- `const phoenix::function< type::getmember_impl< std::vector< std::string >.&type::columns >> stm::db::sql::select::getColumns = type::getmember_impl<std::vector<std::string>,&type::columns> ()`
- `const phoenix::function< push_back_impl > stm::db::sql::select::push_back = push_back_impl ()`

13 SysToMath DB C++ Libraries Example Documentation

13.1 dbmaptest.cpp

This example is a CLI based console C++ program demonstrating the usage and testing the implementation of the SysToMath DbMap C++ Library `stmdbmap`.

```

/* ***** BEGIN LICENSE BLOCK *****
 * Version: MPL 1.1/GPL 2.0/LGPL 2.1
 *
 * The contents of this file are subject to the Mozilla Public License Version
 * 1.1 (the "License"); you may not use this file except in compliance with
 * the License. You may obtain a copy of the License at
 * http://www.mozilla.org/MPL/
 *
 * Software distributed under the License is distributed on an "AS IS" basis,
 * WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License
 * for the specific language governing rights and limitations under the
 * License.
 *
 * The Original Code is the SysToMath DB C++ Libraries package (LibStmDb).
 *
 * The Initial Developer of the Original Code is
 * Tom Michaelis, SysToMath.
 * Portions created by the Initial Developer are Copyright (C) 2003-2008
 * the Initial Developer. All Rights Reserved.
 *
 * Contributor(s):
 *
 * Alternatively, the contents of this file may be used under the terms of
 * either the GNU General Public License Version 2 or later (the "GPL"), or
 * the GNU Lesser General Public License Version 2.1 or later (the "LGPL"),
 * in which case the provisions of the GPL or the LGPL are applicable instead
 * of those above. If you wish to allow use of your version of this file only
 * under the terms of either the GPL or the LGPL, and not to allow others to
 * use your version of this file under the terms of the MPL, indicate your
 * decision by deleting the provisions above and replace them with the notice
 * and other provisions required by the GPL or the LGPL. If you do not delete
 * the provisions above, a recipient may use your version of this file under
 * the terms of any one of the MPL, the GPL or the LGPL.
 *
 * ***** END LICENSE BLOCK ***** */

/*****
 *
 * First Release 2003-03-15
 *
 *****/

const char *Copyright = "(C) 2003-2008 Tom Michaelis, SysToMath";
const char *Version = "1.04-r364";

#ifdef _WIN32
#include <direct.h>
#else
#define _MAX_PATH 1024

```

```
#include <unistd.h>
#endif
#include <stdlib.h>
#include <time.h>
#include <exception>
#include <string>
#include <iostream>

// Test of stm::db::map interface.

#include <stm/dbmap.hpp>

static void test ();

struct Data
{
    Data (const std::string &c = std::string (), unsigned int s = 0) :
        company (c),
        salary (s)
    {}

    std::string company;
    unsigned int salary;
};

namespace stm {
namespace db {
template <>
struct stream<Data>
{
    static void get (buffer &buf, const Data &value)
    {
        buf << value.company << value.salary;
        return;
    }

    static void put (buffer &buf, Data &value)
    {
        buf >> value.company >> value.salary;
        return;
    }
};
} // namespace stm::db

typedef stm::db::map<unsigned int, Data> DataMap;
typedef DataMap::index<std::string> CompIndex;
typedef DataMap::index<unsigned int> SalIndex;

static bool getCompany (const DataMap::value_type &value, CompIndex::skey_type &skey)
{
    skey = value.second.company;
    return true;
}

static bool getSalary (const DataMap::value_type &value, SalIndex::skey_type &skey)
{
    skey = value.second.salary;
    return true;
}

int main (int argc, char *argv [])
```

```

{
    static std::string prog = argv [0];
    prog = prog.substr (prog.find_last_of ("/\\") + 1);
    srand ((unsigned int) time (NULL));
    if (argc > 1 && !strcmp (argv [1], "-t"))
    {
        test ();
    }
    try
    {
        char buffer [_MAX_PATH];
        std::string dbPath = getcwd (buffer, _MAX_PATH);
        DataMap dataMap (dbPath + "../..../test/Data");
        CompIndex compIndex (dataMap, getCompany, "Company");
        SalIndex salIndex (dataMap, getSalary, "Salary");
        for (int i = 0; i < 200; ++ i)
        {
            std::string company;
            size_t len = 4 + rand () % 7;
            for (size_t l = 0; l < len; ++ l)
            {
                company += char ('a' + rand () % 26);
            }
            dataMap [rand () % 5000] = Data (company, (rand () * 100) % 4711);
        }
        for (int i = 0; i < 500; ++ i)
        {
            dataMap.erase (rand () % 5000);
        }
        for (DataMap::const_iterator i = dataMap.begin (); i != dataMap.end (); ++ i)
        {
            std::cout << "Primary key: " << i -> first << ", "
                << "Company: " << i -> second.company << ", "
                << "Salary: " << i -> second.salary << std::endl;
        }
        std::cout << std::endl;
        for (SalIndex::const_iterator i = salIndex.begin (); i != salIndex.end (); ++ i)
        {
            std::cout << "Secondary key: " << i () << ", "
                << "Primary key: " << i -> first << ", "
                << "Company: " << i -> second.company << ", "
                << "Salary: " << i -> second.salary << std::endl;
        }
        std::cout << std::endl;

        // Test of const non-const handling
        DataMap::iterator nci = dataMap.begin ();
        DataMap::const_iterator ci;
        ci = nci;
        ++ ci;
        SalIndex::iterator ncx = salIndex.begin ();
        SalIndex::const_iterator cx;
        cx = ncx;
        ++ cx;
        ci = cx;
        ++ ci;
        ci = ncx;
        ++ ci;

        CompIndex::iterator lb = compIndex.lower_bound ("y");
        CompIndex::iterator ub = compIndex.lower_bound ("z");
        for (CompIndex::iterator i = lb; i != ub; ++ i)
        {
            std::cout << "Secondary key: " << i () << ", "
                << "Primary key: " << i -> first << ", "
                << "Company: " << i -> second.company << ", "
                << "Salary: " << i -> second.salary << std::endl;
        }
    }
}

```

```

    }
    std::cout << int (std::distance (lb, ub)) << " records begin with 'y'." << std::endl;
    std::cout << "The database contains " << dataMap.size ()
        << " (" << compIndex.size () << ") records before erasure." << std::endl;
    CompIndex::iterator d = compIndex.lower_bound ("z");
    std::cout << int (std::distance (d, compIndex.end ())) << " records begin with 'z'." << std::endl;
    compIndex.erase (d, compIndex.end ());
    std::cout << "The database contains " << dataMap.size ()
        << " (" << compIndex.size () << ") records after erasure." << std::endl;
}
catch (std::exception &exc)
{
    std::cerr << prog << ": " << exc.what () << std::endl;
}
return 0;
}

#include <boost/mpl/transform.hpp>
#include <boost/mpl/equal.hpp>
#include <boost/static_assert.hpp>
#include <boost/lambda/lambda.hpp>
static void test ()
{
    typedef boost::mpl::list<char, short, int, long, float, double> types;
    typedef boost::mpl::list<char *, short *, int *, long *, float *, double *> pointers;
    typedef boost::mpl::transform <types, boost::add_pointer<boost::mpl::_1> >::type result;
    BOOST_STATIC_ASSERT ((boost::mpl::equal<result, pointers>::type::value));

    typedef stm::db::conslist
    <
        boost::mpl::list
        <
            int,
            double,
            int,
            int,
            int,
            int,
            int,
            int,
            int,
            int,
            int
        >
    > ConsList;
    ConsList aconslist;
    aconslist.get<0> () = 1;
    aconslist.get<1> () = 3.14;
    aconslist.get<9> () = 13;
    std::cout << aconslist.get<0> () << ", "
        << aconslist.get<1> () << ", "
        << aconslist.get<9> () << std::endl;
    typedef stm::db::optional<int> OptInt;
    typedef boost::variant<int, std::string, float> Var1;
    Var1 var1 = "hello!";
    typedef boost::make_variant_over
    <
        boost::mpl::pop_front<Var1::types>::type
    >::type Var2;
    Var2 var2 = boost::get<std::string> (var1);
    std::cout << "is_variant<Var1>: " << stm::db::is_variant<Var1>::value << std::endl;
    std::cout << "is_variant<Var2>: " << stm::db::is_variant<Var2>::value << std::endl;
    std::cout << "is_variant<int>: " << stm::db::is_variant<int>::value << std::endl;
    typedef boost::tuple<int, std::string, float> Tuple;
    std::cout << "is_conslist<Tuple>: " << stm::db::is_conslist<Tuple>::value << std::endl;
    std::cout << "length<Tuple>: " << boost::tuples::length<Tuple>::value << std::endl;
    std::cout << "is_conslist<ConsList>: " << stm::db::is_conslist<ConsList>::value << std::endl;
}

```

```

std::cout << "length<ConsList>: " << boost::tuples::length<ConsList>::value << std::endl;
std::cout << "is_conslist<int>: " << stm::db::is_conslist<int>::value << std::endl;
std::cout << "is_optional<OptInt>: " << stm::db::is_optional<OptInt>::value << std::endl;
std::cout << "is_variant<OptInt>: " << stm::db::is_variant<OptInt>::value << std::endl;
std::cout << "is_optional<int>: " << stm::db::is_optional<int>::value << std::endl;
std::cout << "sizeof (bool): " << sizeof (bool) << std::endl;
std::cout << "sizeof (boost::tuples::null_type): " << sizeof (boost::tuples::null_type) << std::endl;
std::cout << "boost::is_empty<boost::tuples::null_type>::value: " << boost::is_empty<boost::tuples::nu
std::cout << "boost::is_empty<boost::mpl::false_>::value: " << boost::is_empty<boost::mpl::false_>::va
std::cout << "boost::is_empty<boost::tuple<> >::value: " << boost::is_empty<boost::tuple<> >::value <<
std::cout << "boost::is_empty<stm::db::conslist<boost::mpl::list<> > >::value: " << boost::is_empty<st
std::cout << "boost::is_same<boost::tuple<>, stm::db::conslist<boost::mpl::list<> > >::value: "
    << boost::is_same<boost::tuple<>, stm::db::conslist<boost::mpl::list<> > >::value << std::en

    exit (0);
}

```

14 SysToMath DB C++ Libraries Page Documentation

14.1 Microsoft Visual Studio Tool Family

The Microsoft Visual Studio tool family consists of the tool sets:

- Microsoft Visual Studio .NET 2003 (vc71)
- Microsoft Visual Studio 2005 (vc80)

14.1.1 Automatic Linking with Microsoft Visual Studio

On Microsoft Visual Studio .NET 2003 (vc71) and Microsoft Visual Studio 2005 (vc80) the necessary libraries are linked automatically when one of the library object interface header files [stm/dbmap.hpp](#) or [stm/dbsql.hpp](#) is included, unless this mechanism is suppressed by the definition of the preprocessor symbol STM_NO_LIB, STM_DBMAP_NO_LIB or STM_DBSQL_NO_LIB before that inclusion.

The choice of the libraries depends on the tool set used (vc71 or vc80) and on the system runtime library selected for the executable to be built. The SysToMath DB C++ Libraries package provides for each of its library modules two static library configurations (lib files) and two dynamic ones (dll files). In the following list *module* stands for *berkeleydb42* or *stmdbsql* and *vcnn* for *vc71* or *vc80*:

- **DebugDIIMt**: Dynamic debug library *module-vcnn-mt-gd.dll* together with its import library *module-vcnn-mt-gd.lib* and its debug database file *module-vcnn-mt-gd.pdb* chosen when linking against the multithreaded debug DLL runtime (Compiler switch /MDd) and defining the preprocessor variable STM_DYN_LINK before the SysToMath DB C++ Libraries header file inclusion.
- **DebugMtStaticRt**: Static debug library *libmodule-vcnn-mt-sgd.lib* together with its debug database file *libmodule-vcnn-mt-sgd.pdb* chosen when linking against the multi-threaded static debug runtime (Compiler switch /MTd).
- **ReleaseDIIMt**: Dynamic release library *module-vcnn-mt.dll* together with its import library *module-vcnn-mt.lib* chosen when linking against the multithreaded release DLL runtime (Compiler switch /MD) and defining the preprocessor variable STM_DYN_LINK before the SysToMath DB C++ Libraries header file inclusion.

- **ReleaseMtStaticRt:** Static release library `libmodule-vcnn-mt-s.lib` chosen when linking against the multithreaded static release runtime (Compiler switch `/MT`).

14.1.2 Environment

It is recommended that all static libraries (`lib` files) and their debug database files (`pdb` files) are located in a directory contained in the compiler system library search path. Moreover, to satisfy the application runtime requirements, it is recommended that all dynamic link libraries (`dll` files) and their debug database files (`pdb` files) are located in the application directory or in a directory contained in the system executable search path.

If you used the installation program `LibStmDbSetup.exe` to install the SysToMath DB C++ Libraries with the installation root directory, say `C:\Program Files\SysToMath` where you also installed the prerequisite library packages

- `PthreadsWin32-2`
- `LibStmC`
- `Boost-1.33`
- `LibStmCpp`
- `BerkeleyDb-4.2`

and you use Microsoft Visual Studio .NET 2003 (`vc71`) or Microsoft Visual Studio 2005 (`vc80`), then the aforementioned compiler system directory recommendations are satisfied, if you add the following entries in Visual Studio, menu Tools, Options, Projects, VC++ Directories:

- Executable Files: Add `C:\Program Files\SysToMath\bin\w32`
- Library Files: Add `C:\Program Files\SysToMath\lib\w32`
- Include Files: Add `C:\Program Files\SysToMath\include`
- Include Files: Add `C:\Program Files\SysToMath\include\ptw32`
- Include Files: Add `C:\Program Files\SysToMath\include\boost-1_33`
- Include Files: Add `C:\Program Files\SysToMath\include\db-4.2.52`

The example program can then be found in `C:\Program Files\SysToMath\samples\LibStmDb` together with ready to use Visual Studio solution files `w32\LibStmDb-vc71.sln` and `w32\LibStmDb-vc80.sln`.

14.2 SysToMath DbMap C++ Library

14.2.1 Content

The SysToMath DbMap C++ Library consists of the following object:

- [DbMap: Named Persistent Associative Containers](#)

14.2.2 Usage

To compile an application which uses the [SysToMath DbMap C++ Library](#), the public library object interface header file

- [stm/dbmap.hpp](#)

shall be included and be located in a directory contained in the compiler system include search path. Moreover, the implementation header file

- [stm/impl/dbxmap.hpp](#)

shall also be located in a directory contained in the compiler system include search path.

For an example see [dbmaptest.cpp](#).

14.3 SysToMath DbSql C++ Library

14.3.1 Content

The SysToMath DbSql C++ Library consists of the following object:

- [DbSql: SQL Interface of Named Persistent Associative Containers](#)

14.3.2 Usage

To compile an application which uses the [SysToMath DbSql C++ Library](#), the public library object interface header file

- [stm/dbsql.hpp](#)

shall be included and be located in a directory contained in the compiler system include search path. Moreover, the implementation header file

- [stm/impl/dbxsql.hpp](#)

shall also be located in a directory contained in the compiler system include search path.

Index

- ~Item
 - stm::db::local::xmap::Item, 66
- ~table
 - stm::db::sql::table, 58
- activateTable
 - ModDbSql, 17
- Align
 - stm::db::local::xmap::Item, 65
- alloc
 - stm::db::local::xmap::descr_type, 63
- alloc_
 - stm::db::local::xmap::Item, 67
- append
 - stm::db::buffer, 30
 - stm::db::local::xmap::Item, 67
- char_type
 - ModDbMap, 10
- check
 - stm::db::local::xmap::xiter, 72
- checkSecondaryKeys
 - DbSqlImpl, 20
- chunkssize_
 - stm::db::local::xmap::Item, 68
- chunksvec_
 - stm::db::local::xmap::Item, 68
- Class template db::map, 10
- Class template db::set, 12
- columns
 - stm::db::sql::table, 58
- comp
 - stm::db::local::xmap::descr_type, 62
- copyTo
 - stm::db::local::xmap::Item, 66
- createTable
 - ModDbSql, 16, 17
- current_
 - stm::db::local::xmap::xiter, 72
- db::map comparison operators, 11
- db::map specialized algorithms, 11
- db::set comparison operators, 12
- db::set specialized algorithms, 13
- dbAccess_
 - stm::db::local::xmap::xiter, 72
- DbMap Implementation, 13
- dbmap.hpp, 73
- DbMap: Named Persistent Associative Containers, 6
- dbmaptest.cpp, 78
- DbSql Implementation, 19
- dbsql.cpp, 80
- dbsql.hpp, 80
- DbSql: SQL Interface of Named Persistent Associative Containers, 14
- DbSqlImpl
 - checkSecondaryKeys, 20
- dbxmap.hpp, 82
- dbxsql.hpp, 88
- deactivateTable
 - ModDbSql, 17
- defineType
 - stm::db::sql::table, 58
- delayedsort
 - ModDbSql, 18
- deleteTable
 - ModDbSql, 17
- difference_type
 - ModDbMap, 10
- endiancvt
 - stm::db::buffer, 31
 - stm::db::local::xmap::descr_type, 62
 - stm::db::local::xmap::Item, 67
- errmsgs
 - ModDbSql, 18
- errorno
 - ModDbSql, 18
- execsql
 - ModDbSql, 18
- exists
 - stm::db::optional, 48
- existsTable
 - ModDbSql, 16
- extract
 - stm::db::buffer, 30
 - stm::db::local::xmap::Item, 67
- flags
 - stm::db::local::xmap::descr_type, 62
- flags_type
 - stm::db::map, 38
 - stm::db::set, 52
- get
 - stm::db::stream, 55
- getTableDescr
 - ModDbSql, 18
- Item
 - stm::db::local::xmap::Item, 65, 66

- item_type
 - stm::db::local::xmap::Item, 65
- key_type
 - stm::db::local::xmap::xiter, 71
- locking
 - ModDbMap, 10
- map
 - stm::db::map, 38
- ModDbMap
 - char_type, 10
 - difference_type, 10
 - locking, 10
 - size_type, 10
- ModDbSql
 - activateTable, 17
 - createTable, 16, 17
 - deactivateTable, 17
 - delayedsort, 18
 - deleteTable, 17
 - errmsg, 18
 - errno, 18
 - execsql, 18
 - existsTable, 16
 - getTableDescr, 18
 - notsortable, 18
 - path, 16
 - recnosort, 18
 - tables, 17
- name
 - stm::db::local::xmap::descr_type, 62
 - stm::db::sql::table, 58
- notsortable
 - ModDbSql, 18
- operator<<
 - stm::db::buffer, 31
- operator>>
 - stm::db::buffer, 31
- operator()
 - stm::db::local::xmap::Item, 66
- operator=
 - stm::db::local::xmap::Item, 66
 - stm::db::map, 39
 - stm::db::set, 53
- path
 - ModDbSql, 16
- pDbc_
 - stm::db::local::xmap::xiter, 72
- pDbImpl_
 - stm::db::local::xmap::Item, 67
- put
 - stm::db::stream, 55
- readoffs_
 - stm::db::local::xmap::Item, 68
- recnosort
 - ModDbSql, 18
- resize
 - stm::db::local::xmap::Item, 66
- set
 - stm::db::set, 53
- size
 - stm::db::local::xmap::descr_type, 62
 - stm::db::sql::table, 58
- size_type
 - ModDbMap, 10
- stm::db, 25
 - stm::db::buffer, 30
 - append, 30
 - endiancvt, 31
 - extract, 30
 - operator<<, 31
 - operator>>, 31
 - stm::db::default_policy, 33
 - stm::db::default_traits, 33
 - stm::db::local::xmap::descr_type, 61
 - alloc, 63
 - comp, 62
 - endiancvt, 62
 - flags, 62
 - name, 62
 - size, 62
 - stm::db::local::xmap::Item, 63
 - ~Item, 66
 - Align, 65
 - alloc_, 67
 - append, 67
 - chunkssize_, 68
 - chunksvec_, 68
 - copyTo, 66
 - endiancvt, 67
 - extract, 67
 - Item, 65, 66
 - item_type, 65
 - operator(), 66
 - operator=, 66
 - pDbImpl_, 67
 - readoffs_, 68
 - resize, 66
 - stm::db::local::xmap::secDb, 68
 - stm::db::local::xmap::xiter, 69
 - check, 72
 - current_, 72

- dbAccess_, 72
- key_type, 71
- pDbc_, 72
- unlink, 71
- xiter, 71
- stm::db::map, 34
 - flags_type, 38
 - map, 38
 - operator=, 39
 - swap, 39
- stm::db::map::index, 39
- stm::db::map::locker, 41
- stm::db::multimap, 42
- stm::db::multimap::locker, 44
- stm::db::multiset, 45
- stm::db::multiset::locker, 47
- stm::db::null_type, 47
- stm::db::optional, 48
 - exists, 48
- stm::db::policy, 48
- stm::db::set, 49
 - flags_type, 52
 - operator=, 53
 - set, 53
 - swap, 53
- stm::db::set::locker, 54
- stm::db::sql::connection, 31
- stm::db::sql::local::tableImpl::openIndex, 59
- stm::db::sql::local::tableImpl::openIndexes, 59
- stm::db::sql::local::tableImpl::registerResultCreator, 60
- stm::db::sql::local::tableImpl::registerResultCreators, 60
- stm::db::sql::table, 56
 - ~table, 58
 - columns, 58
 - defineType, 58
 - name, 58
 - size, 58
- stm::db::stream, 54
 - get, 55
 - put, 55
- stm::db::stream< std::pair< T1, T2 > >, 55
- stm::db::traits, 61
- stmdbmap/ Directory Reference, 24
- stmdbmap/stm/ Directory Reference, 23
- stmdbmap/stm/impl/ Directory Reference, 22
- stmdbmap/test/ Directory Reference, 25
- stmdbsql/ Directory Reference, 24
- stmdbsql/stm/ Directory Reference, 23
- stmdbsql/stm/impl/ Directory Reference, 22
- swap
 - stm::db::map, 39
 - stm::db::set, 53
- SysToMath DbMap C++ Library, 6
- SysToMath DbMap C++ Library Test, 21
- SysToMath DbSql C++ Library, 13
- tables
 - ModDbSql, 17
- unlink
 - stm::db::local::xmap::xiter, 71
- xiter
 - stm::db::local::xmap::xiter, 71