SwatDB

Generated by Doxygen 1.8.13

Contents

Chapter 1

SwatDB header files

These are .h files that can be included by any module in swatdb. They include .h interface files to the different layers and .h files that define common type definitions, typically used inter-layer. In test program, use these typedef types rather than the underlying base type to which they are currently defined.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

Class Documentation

4.1 BufferManager Class Reference

```
#include <bufmgr.h>
```

Public Member Functions

- BufferManager (DiskManager ∗disk_mgr)

  BufferManager constructor. Initializes the buf_pool and frame_table, and stores a pointer to SwatDB's DiskManager.
- ∼BufferManager ()

  BufferManager destructor.
- std::pair< Page ∗, PageId > allocatePage (FileId file_id)

  Allocates a Page for the file of given FileId. The Page is allocated both in the buffer pool, and on disk.
- void deallocatePage (PageId page_id)

  Removes the Page of the given PageId from the buffer pool, and deallocates the Page from the appopriate file on disk.
- Page ∗ getPage (PageId page_id)

  Gets Page by page_id, pins the Page, and returns a pointer to the Page object.
- void releasePage (PageId page_id, bool dirty)

  Unpins a Page in the buffer pool.
- void setDirty (PageId page_id)

  Set the Page of the given PageId dirty.
- void flushPage (PageId page_id)

  Flushes the Page of the given PageId to disk.
- void createFile (FileId file_id)

  Calls createFile() method on the DiskManager to create new Unix file that corresponds to the given FileId.
- void removeFile (FileId file_id)

  Calls removeFile() method on the DiskManager. Checks that none of the file's pages are pinned in the buffer pool. Removes any of the file's pages from the buffer pool before removing from disk.
- void clearBuffer ()

  THIS METHOD IS FOR DEBUGGING ONLY. Clears the entire buffer pool, resetting all frames and removing pages from the buffer_map even if pinned. Does not flush any dirty pages either.
- BufferState getBufferState ()

  THIS METHOD IS FOR DEBUGGING ONLY. Returns the current state of the buffer pool.
- void printAllFrames ()

THIS METHOD IS FOR DEBUGGING ONLY. Prints Frame state of every Frame in the buffer pool, including pin count, valid bit, dirty bit, and ref_bit. If Page is valid, PageId is printed.

- void printValidFrames ()

  THIS METHOD IS FOR DEBUGGING ONLY. Prints Frame state of every valid Frame in the buffer pool, including PageId, pin count, valid bit, dirty bit, and ref_bit.
- void printFrame (FrameId frame_id)

  THIS METHOD IS FOR DEBUGGING ONLY. Prints Frame state of given FrameId, including pin count, valid bit, dirty bit, and ref_bit. If Page is valid, PageId is printed.
- void printPage (PageId page_id)

  THIS METHOD IS FOR DEBUGGING ONLY. Prints Frame state of given PageId, including FrameId, pin count, valid bit, dirty bit, and ref_bit. If Page is not in the buffer map, prints "Page Not Found".
- void printBufferState ()

  THIS METHOD IS FOR DEBUGGING ONLY. Prints current buffer state, including total number of pages, number of valid pages, number of pinned pages, number of dirty pages, number of pages whose ref bit is set and the current clock hand position.

4.1.1 Detailed Description

SwatDb BufferManager Class. BufferManager manages in memory space of DBMS at page level granularity. At higher level, pages of data could be allocated, deallocated, retrieved to memory and fliushed to disk, using various methods.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 BufferManager()

```
BufferManager::BufferManager (
            DiskManager * disk_mgr )
```

BufferManager constructor. Initializes the buf_pool and frame_table, and stores a pointer to SwatDB's DiskManager.

Precondition

disk_mgr points to an initialized DiskManager object.

Postcondition

A BufferManager object will be initialized with an empty buffer pool. disk_mgr is set to the given DiskManager∗ and clock_hand is set to 0.

Parameters

disk_mgr	A pointer to SwatDB's DiskManager object. (DiskManager∗).

4.1.2.2 ∼BufferManager()

```
BufferManager::∼BufferManager ( )
```

BufferManager destructor.

Precondition

None.

Postcondition

Every valid and dirty Page in buffer pool is written to disk.

4.1.3 Member Function Documentation

4.1.3.1 allocatePage()

```
std::pair<Page∗,PageId> BufferManager::allocatePage (
            FileId file_id )
```

Allocates a Page for the file of given FileId. The Page is allocated both in the buffer pool, and on disk.

Precondition

A valid FileId is provided and there is a free Page on disk or there is enough space in Unix file. There is also free space in the buffer pool, or a Page which can be evicted from the buffer pool.

Postcondition

A Frame is allocated, and a corresponding Page in the buffer pool is allocated. The Frame's page_id is set, valid is set to true, and the pin_count is set to 1. The PageId is added to the BufferMap. Finally, a pair of a pointer to the allocated Page and PageId is returned.

Parameters

file_id	A FileId to which a Page should be allocated.

Returns

std::pair of Page∗ and PageId of the allocated Page.

Exceptions

InvalidFileIdDiskMgr	If page_id.file_id not valid.
InsufficientSpaceBufMgr	If there is not enough space in buffer pool.
InsufficientSpaceDiskMgr	If there is not enough space in the Unix file.

4.1.3.2 createFile()

```
void BufferManager::createFile (
            FileId file_id )
```

Calls createFile() method on the DiskManager to create new Unix file that corresponds to the given FileId.

Precondition

FileId is valid.

Postcondition

Unix file that corresponds to the file_id is created.

Parameters

file←_id	FileId of the file to be created.

See also

DiskManager::createFile()

4.1.3.3 deallocatePage()

```
void BufferManager::deallocatePage (
            PageId page_id )
```

Removes the Page of the given PageId from the buffer pool, and deallocates the Page from the appopriate file on disk.

Precondition

A valid PageId of an unpinned Page is provided as a parameter.

Postcondition

If the Page is in the buffer pool, the Frame is reset, and the Page is removed from the buffer pool. The Page is deallocated from disk.

Parameters

page←Id	PageId of the Page to be deallocated

Exceptions

InvalidPageIdBufMgr	If page_id is not valid.
PagePinnedBufMgr	If the Page is pinned.

4.1.3.4 flushPage()

```
void BufferManager::flushPage (
             PageId page_id )
```

Flushes the Page of the given PageId to disk.

Precondition

A PageId of a pinned Page is provided as input.

Postcondition

If the Page is set dirty, the Page is written to disk through the disk_mgr. Page is still pinned.

Parameters

page↩ _id	PageId of the Page to set dirty.

Exceptions

PageNotFoundBufMgr	If page_id not in buf_map.
InvalidFileIdDiskMgr	If page_id.file_id not valid.
InvalidPageNumDiskMgr	If page_id.page_num not valid.

4.1.3.5 getBufferState()

```
BufferState BufferManager::getBufferState ( )
```

THIS METHOD IS FOR DEBUGGING ONLY. Returns the current state of the buffer pool.

See also

BufferState

4.1.3.6 getPage()

```
Page* BufferManager::getPage (
            PageId page_id )
```

Gets Page by page_id, pins the Page, and returns a pointer to the Page object.

Precondition

A PageId of an allocated Page is provided as a parameter and buffer pool is not full of pinned pages.

Postcondition

If the page_id is in buf_map, the Page is pinned and its pointer is returned. Else, a Frame is allocated in the buffer pool according to the page replacement policy, the Page is read from disk_mgr into the buffer pool, and the page_id, pin count, and valid bit are set. Page∗ is returned.

Parameters

page←_id	A PageId corresponding to the pointer to be returned.

Returns

Pointer to the Page with page_id.

Exceptions

InvalidPageIdBufMgr	If page_id is not valid.
InsufficientSpaceBufMgr	If buffer pool is full.

4.1.3.7 releasePage()

```
void BufferManager::releasePage (
            PageId page_id,
            bool dirty )
```

Unpins a Page in the buffer pool.

Precondition

A PageId of a pinned Page is provided as input. The Page is in the buffer pool and is pinned by the executing thread/process.

Postcondition

The pin count of the Page is decremented. ref_bit is set to true. Dirty bit is set if the dirty parameter is true.

Parameters

page←↩ _id	PageId of the Page to be released.

Exceptions

PageNotPinnedBufMgr	If Page is not pinned. (pin_count is 0).
PageNotFoundBufMgr	If page_id is not in buf_map.

4.1.3.8 removeFile()

```
void BufferManager::removeFile (
             FileId file_id )
```

Calls removeFile() method on the DiskManager. Checks that none of the file's pages are pinned in the buffer pool. Removes any of the file's pages from the buffer pool before removing from disk.

Precondition

A valid FileId is given as a parameter. None of the file's pages are pinned in the buffer pool.

Postcondition

If the file has pages in the buffer pool, the corresponding frames are reset and pages are removed from buf_map. The file is removed from disk via DiskManager->removeFile().

Parameters

file←↩ _id	FileId of the file to be removed.

Exceptions

PagePinnedBufMgr	If the are pinned pages of file_id.

See also

DiskManager::removeFile()

4.1.3.9 setDirty()

```
void BufferManager::setDirty (
             PageId page_id )
```

Set the Page of the given PageId dirty.

Precondition

A PageId of a pinned Page is provided as input.

Postcondition

The Page is set dirty.

Parameters

page← _id	PageId of the Page to set dirty.

Exceptions

PageNotFoundBufMgr	If page_id is not in the buffer pool.

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

- /home/koh2/swatdb/SwatDB/include/bufmgr.h

4.2 BufferMap Class Reference

```
#include <bufmgr.h>
```

Public Member Functions

- BufferMap ()

    Constructor for BufferMap.
- ∼BufferMap ()

    Destructor for BufferMap.
- FrameId get (PageId page_id)

    Returns FrameId corresponding to the given PageId.
- bool contains (PageId page_id)

    Returns true if the map contains the given PageId, else false.
- void insert (PageId page_id, FrameId frame_id)

    Inserts the pair <page_id, frame_id> into the map.
- void remove (PageId page_id)

    Removes the key-value pair corresponding to the given PageId from the map.

4.2.1 Detailed Description

BufferMap is a wrapper class for std::unordered_map<PageId, FrameId> that maps PageIds to a Frame index in the buffer pool. Has different method names from std::unordered_map. Have get(), contains(), insert(), and remove() methods.

4.2.2 Member Function Documentation

4.2.2.1 contains()

```
bool BufferMap::contains (
            PageId page_id )
```

Returns true if the map contains the given PageId, else false.

Precondition

A lock is held on the map. A PageId is provided.

Postcondition

Returns true if the map contains the PageId, else false. Lock is still held on the map.

Parameters

page←_id	A PageId to be searched in the map.

Returns

bool indicating whether the given PageId exists in the BufferMap.

4.2.2.2 get()

```
FrameId BufferMap::get (
            PageId page_id )
```

Returns FrameId corresponding to the given PageId.

Precondition

A lock is held on the map. This map contains the given PageId.

Postcondition

Returns the corresponding FrameId. Lock is still held on the map.

Parameters

page←↩ _id	A PageId for which the corresponding FrameId will be returned.

Returns

FrameId which correspond to the given PageId.

Exceptions

PageNotFoundBufMgr	if the given PageId is not in the map.

4.2.2.3 insert()

```
void BufferMap::insert (
            PageId page_id,
            FrameId frame_id )
```

Inserts the pair <page_id, frame_id> into the map.

Precondition

A lock is held on the map. A PageId and FrameId are provided as input.

Postcondition

If the map contains page_id, then the FrameId in the FrameId will be updated. Else a new <PageId, FrameId> pair is added to the map. Lock is still held on the map.

Parameters

page_id	A PageId key.
frame←↩ _id	A FrameId value.

4.2.2.4 remove()

```
void BufferMap::remove (
            PageId page_id )
```

Removes the key-value pair corresponding to the given PageId from the map.

Precondition

A lock is held on the map. The given PageId is in the map.

Postcondition

The key-value pair searched by page_id is removed from the map.

Parameters

page←↩ _id	The PageId key for the key-value pair to be removed.

Exceptions

PageIdNotFoundBufMgr	If page_id is not in the map.

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

- /home/koh2/swatdb/SwatDB/include/bufmgr.h

4.3 BufferState Struct Reference

```
#include <bufmgr.h>
```

Public Attributes

- std::uint32_t total
- std::uint32_t valid
- std::uint32_t pinned
- std::uint32_t unpinned
- std::uint32_t dirty
- std::uint32_t ref_bit
- std::uint32_t clock_hand

4.3.1 Detailed Description

THIS STRUCT IS FOR DEBUGGING ONLY. Struct that represents the state of the buffer pool.

4.3.2 Member Data Documentation

4.3.2.1 clock_hand

```
std::uint32_t BufferState::clock_hand
```

The current position of the clock hand.

4.3.2.2 dirty

```
std::uint32_t BufferState::dirty
```

The number of dirty pages in the buffer pool.

4.3.2.3 pinned

```
std::uint32_t BufferState::pinned
```

The number of pinned pages in the buffer pool.

4.3.2.4 ref_bit

```
std::uint32_t BufferState::ref_bit
```

The number of pages that have ref_bit set in the buffer pool.

4.3.2.5 total

```
std::uint32_t BufferState::total
```

The total number of pages in the buffer pool.

4.3.2.6 unpinned

```
std::uint32_t BufferState::unpinned
```

The number of unpinned pages in the buffer pool.

4.3.2.7 valid

```
std::uint32_t BufferState::valid
```

The number of valid pages in the buffer pool.

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

- /home/koh2/swatdb/SwatDB/include/bufmgr.h

4.4 BufHash Struct Reference

```
#include <bufmgr.h>
```

Public Member Functions

- std::size_t **operator()** (const PageId &page_id) const

4.4.1 Detailed Description

Hash function for BufferMap.

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

- /home/koh2/swatdb/SwatDB/include/bufmgr.h

4.5 Catalog Class Reference

```
#include <catalog.h>
```

Public Member Functions

- Catalog ()

  creates an empty Catalog (should only be one in the system)
- Catalog (std::string db_metadata_file)

  create a catalog from existing DB state stored in meta data file
- ∼Catalog ()

  Destroys the Catalog object.
- FileId addEntry (std::string name, Schema ∗schema, File ∗file, CatType type, std::string file_name)

  Adds an entry to the catalog with a defined schema. May be called to create an entry for an index or relation as the SwatDB instance runs, or may be called as a SwatDB instance is booted from a saved state.
- void deleteEntry (FileId file_id)

  Deletes an entry from the database.
- std::string getFileName (FileId file_id)

  Returns the filename associated with the given FileId.
- FileId getFileId (std::string name)

  Returns the FileId associated with the File identified by the given relation/index name.
- File ∗ getFile (FileId file_id)

  Returns the File object associated with the given FileId.
- Schema ∗ getSchema (FileId file_id)

  Returns the schema of the requested file.
- CatType getType (FileId file_id)

  Returns the type of the requested file.
- std::vector< FileId > getFileIds ()

  Gets the set of valid FileIDs in the system.

Protected Member Functions

- void _setFile (FileId file_id, File ∗file_ptr)

    Set the File ∗ field of a Catalog entry.
- void _saveDBStateToFile (std::string db_metadata_filename)

    Valled by SwatDB on shutdown to save DB meta data state to a file.

Friends

- class **SwatDB**
- class **FileManager**

4.5.1 Detailed Description

SwatDB Catalog Class: defines the interface to the part of SwatDB that keeps track of information about Relations and Indices in the system. High-level layers may add index and relation entries, and examine their schema for query processing. Low level layers may access some internal information about a Relation or Index through methods provided by the Catalog class. A Relation and Index is uniquely identified in the system by its FileId value.

The Catalog class is the only one that needs to know the format of a saved DB metadata file. It is the class that reads and parses the DB metadata as part of initing the SwatDB state to an existing DB, and also the one that saves Catalog state to a DB metadata file on shutdown. The main SwatDB class controls invoking the Catalog class constructor to init the catalog from a saved DB metadata file or to init an empty Catalog on start-up. It also is the only class that can invoke the Catalog's _saveDBStateToFile method, which it may do on shutdown of the DB. Currently, the metadata file format is the following (NOTE: TODO missing is representation of entries schema in metadata): num_entries 1st entry's Relation or Index name (string) 1st entry's File or Relation Type (int (its CatType value)) 1st entry's Disk file name (string) 2nd entry's Relation of Index name (string) ... FileIds are unique for a full execution of SwatDB regardless of if relations and indices are created or deleted at runtime. They do not persist across two separate boots of SwatDB.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 Catalog()

```
Catalog::Catalog (
             std::string db_metadata_file )
```

create a catalog from existing DB state stored in meta data file

Parameters

db_metadata_file	file containing metadata information about the DB with which to initialize the Catalog

4.5.2.2 ∼Catalog()

```
Catalog::∼Catalog ( )
```

Destroys the Catalog object.

NOTE: The destructor does not determine whether the state of the SwatDB should be saved or not in the current implementation.

4.5.3 Member Function Documentation

4.5.3.1 _saveDBStateToFile()

```
void Catalog::_saveDBStateToFile (
             std::string db_metadata_filename )  [protected]
```

Valled by SwatDB on shutdown to save DB meta data state to a file.

Parameters

4.5.3.2 _setFile()

```
void Catalog::_setFile (
             FileId file_id,
             File * file_ptr )  [protected]
```

Set the File ∗ field of a Catalog entry.

This is called by the FileManager constructor when SwatDB is booted and inited from exising db state.

Precondition

file_id is a valid FileId and its Catalog entry has a nullptr value for its file field.

Postcondition

The file field of the Catalog entry for file_id is set to file_ptr

Parameters

file_id	A valid FileId.
file_ptr	A pointer to a valid File object

Exceptions

InvalidFileIdCat	if the FileId is not valid.
FileAlreadyExistCat	if the file entry already has a valid File ∗

4.5.3.3 addEntry()

```
FileId Catalog::addEntry (
             std::string name,
             Schema * schema,
             File * file,
             CatType type,
             std::string file_name )
```

Adds an entry to the catalog with a defined schema. May be called to create an entry for an index or relation as the SwatDB instance runs, or may be called as a SwatDB instance is booted from a saved state.

Precondition

An entry with matching name, type and file_name does not already exist in the system.

Postcondition

A new file or index has been added to the system, the passed File object's file_id field is set by this method since a file's id is determined by adding an entry for it to the Catalog.

Parameters

name	The name of the relation.
schema	The schema object assciated with the relation (may be null).
file	A pointer to the File object associated with the relation/index (may be null).
type	The type of entry (index or relation).
file_name	The name of the file into which the DiskManager stores the relation.

Returns

FileID of the added entry.

Exceptions

FileAlreadyExistCat	if the file_name already exists in the database.
RelationAlreadyExistCat	if a relation named name already exists in the databse.
IndexAlreadyExistsCat	if an index named name already exists in the database.

4.5.3.4 deleteEntry()

```
void Catalog::deleteEntry (
             FileId file_id )
```

Deletes an entry from the database.

Precondition

A valid FileId is provided as input.

Postcondition

The relation or index identified by the given FileId is removed from the database, along with all other stored data associated with it.

Parameters

file←→_id	The FileID of the file/index to remove from the database.

Exceptions

InvalidFileIdCat	if the FileId is not valid.

4.5.3.5 getFile()

```
File* Catalog::getFile (
             FileId file_id )
```

Returns the File object associated with the given FileId.

Precondition

The given FileId is valid.

Postcondition

The File object associated with the given FileId in the system is returned.

Parameters

file←→_id	A FileId.

Returns

File object associated with the given FileId.

Exceptions

InvalidFileIdCat	if the given FileId is not valid.

4.5.3.6 getFileId()

```
FileId Catalog::getFileId (
             std::string name )
```

Returns the FileId associated with the File identified by the given relation/index name.

Precondition

A File identified by the given name exists in the SwatDB system.

Postcondition

FileId of the File identified by the given relation/index name is is returned.

Parameters

file←_id	A FileId.

Returns

FileId of the File identified by the given relation/index name.

Exceptions

InvalidNameCat	if the given name is not valid.

4.5.3.7 getFileIds()

```
std::vector<FileId> Catalog::getFileIds ( )
```

Gets the set of valid FileIDs in the system.

Returns

std::vector<FileId> vector of FileIDs in the system. The vector is empty if there are no files in the database.

4.5.3.8 getFileName()

```
std::string Catalog::getFileName (
            FileId file_id )
```

Returns the filename associated with the given FileId.

Precondition

The given FileId is valid.

Postcondition

The filename associated with the given FileId in the system is returned.

Parameters

file←↩ _id	A FileId.

Returns

std::string of the file name associated with the given FileId.

Exceptions

InvalidFileIdCat	if the given FileId is not valid.

4.5.3.9 getSchema()

```
Schema* Catalog::getSchema (
            FileId file_id )
```

Returns the schema of the requested file.

Precondition

A valid FileId is provided as input.

Postcondition

The schema pointer of the requested file is returned. (note: this points to the same Schema object as in the catalog—it is not a copy).

Parameters

file←↪ _id	A valid FileId.

Returns

Schema∗ associated with the file identified by the given FileId.

Exceptions

InvalidFileIdCat	if the FileId is not valid.

4.5.3.10 getType()

```
CatType Catalog::getType (
            FileId file_id )
```

Returns the type of the requested file.

Precondition

A valid FileId is provided as input.

Postcondition

The type of the file with this FileId is returned.

Parameters

file←↪ _id	A valid FileId.

Returns

CatType value of file identified by the given FileId.

Exceptions

InvalidFileIdCat	if the FileId is not valid.

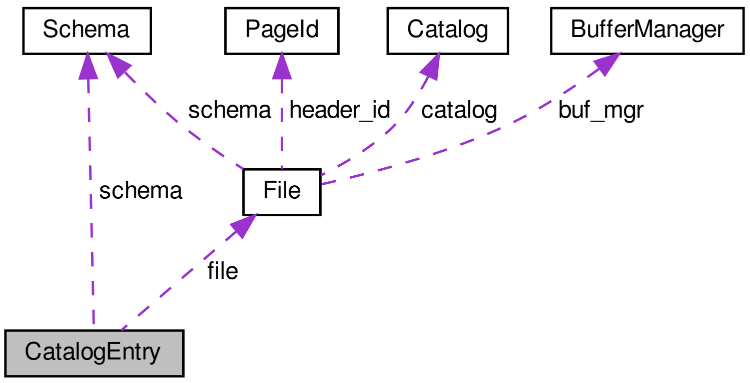
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/catalog.h

4.6 CatalogEntry Struct Reference

```
#include <catalog.h>
```

Collaboration diagram for CatalogEntry:



Public Attributes

- FileId file_id
- std::string name
- CatType entry_type
- Schema ∗ schema
- File ∗ file
- std::string file_name
- bool valid
- bool alloced

4.6.1 Detailed Description

Struct for an entry in the catalog. There is one entry for each relation and index in the system.

4.6.2 Member Data Documentation

4.6.2.1 alloced

```
bool CatalogEntry::alloced
```

Set to true if this entry has been allocated for use.

4.6.2.2 entry_type

```
CatType CatalogEntry::entry_type
```

The type of entry in the catalog (Relation or Index).

4.6.2.3 file

```
File* CatalogEntry::file
```

A pointer to the SwatDB file object (e.g. HeapFile).

4.6.2.4 file_id

```
FileId CatalogEntry::file_id
```

The entry's FileId used as a system-wide identifier for this relation or index.

4.6.2.5 file_name

```
std::string CatalogEntry::file_name
```

The name of the underlying unix file for the file object.

4.6.2.6 name

```
std::string CatalogEntry::name
```

The name of the index or relation.

4.6.2.7 schema

```
Schema* CatalogEntry::schema
```

A pointer to the schema object of this entry.

4.6.2.8 valid

```
bool CatalogEntry::valid
```

Set to true if this entry has valid contents.

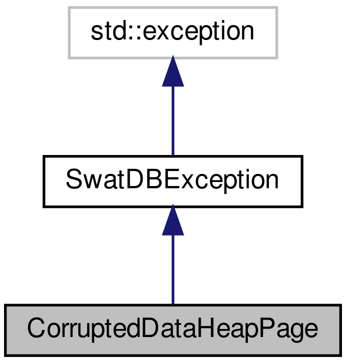
The documentation for this struct was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/catalog.h

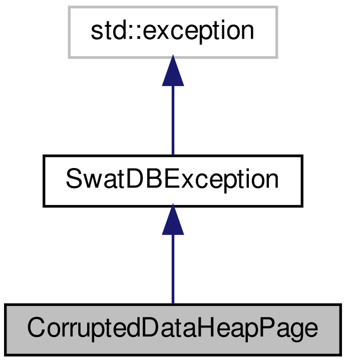
4.7 CorruptedDataHeapPage Class Reference

`#include <exceptions.h>`

Inheritance diagram for CorruptedDataHeapPage:



Collaboration diagram for CorruptedDataHeapPage:



Public Member Functions

- CorruptedDataHeapPage ()

  Constructor.
- ∼CorruptedDataHeapPage () throw ()

  Destructor.

Additional Inherited Members

4.7.1 Detailed Description

CorruptedDataHeapPage is thrown by HeapPage if data on HeapPage is corrupted.

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

- /home/koh2/swatdb/SwatDB/include/exceptions.h

4.8 Data Class Reference

```
#include <data.h>
```

Public Member Functions

- Data ()=delete

  Disable base constructor.
- Data (const Data &other)=delete

  Disable copy constructor.
- Data & operator= (const Data &other)=delete

  Disable copy assignment constructor.
- Data (std::uint32_t size, const char ∗other_data)

  Constructor with given char array and size.
- Data (std::uint32_t size, std::uint32_t capacity)

  Constructor with given size and capacity.
- Data (std::uint32_t capacity)

  Constructor with given capacity.
- ∼Data ()

  Destructor.
- char ∗ getData ()

  Getter for data char array.
- std::uint32_t getSize ()

  Getter for size.
- void setSize (std::uint32_t new_size)

  Setter for size.
- std::uint32_t getCapacity ()

  Getter for capacity.

4.8.1 Detailed Description

SwatDB Data Class. Data a is class that allows storing and moving data of specified size in a more convenient way. A lot of times used as a serialized object that could be given structure by storing it as data member of another object with appropriate methods.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 Data() [1/3]

```
Data::Data (
            std::uint32_t size,
            const char * other_data )
```

Constructor with given char array and size.

Precondition

Valid char* and size are provided.

Postcondition

size bytes long char array is dynamically allocated and the size number of bytes are copied from the given char* pointer. size and capacity are set to the given size.

4.8.2.2 Data() [2/3]

```
Data::Data (
            std::uint32_t size,
            std::uint32_t capacity )
```

Constructor with given size and capacity.

Precondition

None.

Postcondition

capacity bytes long char array is dynamically allocated and capacity is set to the given capacity. size is set to the given size.

Exceptions

InvalidSizeData	If size is greater than capacity.

4.8.2.3 Data() [3/3]

```
Data::Data (
            std::uint32_t capacity )
```

Constructor with given capacity.

Precondition

None.

Postcondition

capacity bytes long char array is dynamically allocated and capacity is set to the given capacity. size is set to 0.

4.8.2.4 ∼Data()

```
Data::∼Data ( )
```

Destructor.

Precondition

Data object is valid.

Postcondition

data array is deallocated.

4.8.3 Member Function Documentation

4.8.3.1 getCapacity()

```
std::uint32_t Data::getCapacity ( )
```

Getter for capacity.

Precondition

None.

Postcondition

capacity of the Data object is returned.

Returns

capacity of Data.

4.8.3.2 getData()

```
char* Data::getData ( )
```

Getter for data char array.

Precondition

None.

Postcondition

char* to data array is returned.

Returns

char* to data array.

4.8.3.3 getSize()

```
std::uint32_t Data::getSize ( )
```

Getter for size.

Precondition

None.

Postcondition

size is returned.

Returns

size of Data.

4.8.3.4 setSize()

```
void Data::setSize (
              std::uint32_t new_size )
```

Setter for size.

Precondition

None.

Postcondition

size is set.

Parameters

size	New size to be set to.

Exceptions

InvalidSizeData	If new_size is greater than capacity.

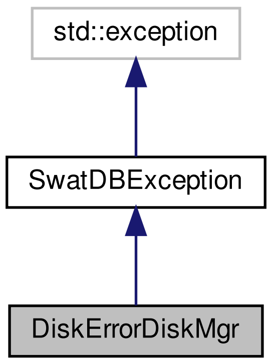
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/data.h

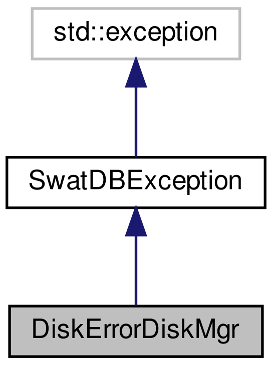
4.9 DiskErrorDiskMgr Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for DiskErrorDiskMgr:



Collaboration diagram for DiskErrorDiskMgr:



Public Member Functions

- DiskErrorDiskMgr ()

  Constructor.
- ∼DiskErrorDiskMgr () throw ()

  Destructor.

Additional Inherited Members

4.9.1 Detailed Description

DiskErrorDiskMgr is thrown by DiskManager if there was error during file operations.

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

- /home/koh2/swatdb/SwatDB/include/exceptions.h

4.10 DiskManager Class Reference

```
#include <diskmgr.h>
```

Public Member Functions

- DiskManager (Catalog ∗catalog)

  DiskManager constructor. Uses Catalog object pointer and a vector of FileIds to initialize file_map.
- ∼DiskManager ()

  DiskManager destructor. All pages have been written to disk prior to this call.
- void createFile (FileId file_id)

  Creates a new Unix file, opens the fstream, initializes DiskFileInfo object, and adds <FileId, DiskFileInfo∗> pair to fileMap. Adds header data to Unix file via SerializedFileInfo.
- void removeFile (FileId file_id)

  Deletes the Unix file corresponding to file_id, and removes the <FileId, DiskFileInfo∗> key-value pair from fileMap.
- PageId allocatePage (FileId file_id)

  Allocates a Page to the file which corresponds to file_id.
- void deallocatePage (PageId page_id)

  Deallocates page by adding its offest to unused_pages in the appropriate DiskFileInfo struct.
- void readPage (PageId page_id, Page ∗page)

  Reads the page data from the Unix file into the Page object pointer.
- void writePage (PageId page_id, Page ∗page)

  Writes the page data of the given Page object at the right offset in the appropriate Unix file.
- bool isValidPage (PageId page_id)

  Checks if the page of a given pageId is valid.
- void printFile (FileId file_id)

  THIS METHOD IS FOR DEBUGGING ONLY. Prints contents of a file inlcuding FileId, size, capacity, and contents of each page.
- std::uint32_t getCapacity (FileId file_id)

  Get method for the capacity of a file.
- std::uint32_t getSize (FileId file_id)

  Get method for size of a file.

4.10.1 Detailed Description

SwatDB DiskManager Class. DiskManager manages page level disk operations of SwatDB, including writing, reading, allocating, and deallocating pages. As SwatDB is built for pedagogical reason, this layer is built on top of regular Unix file system, rather than raw device/DIRECT_IO.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 DiskManager()

```
DiskManager::DiskManager (
            Catalog * catalog )
```

DiskManager constructor. Uses Catalog object pointer and a vector of FileIds to initialize file_map.

Precondition

Valid Catalog∗ is passed as an input.

Postcondition

A vector of FileIds are retrieved from the Catalog and appropriate fstream is opened for each FileId. Then a DiskManager object is initialized and each <FileId, DiskFileInfo∗> pair is added to file_map.

Parameters

catalog	A pointer to the DBMS's catalog object (Catalog∗).

Exceptions

DiskErrorDiskMgr	if file operation fails

4.10.2.2 ∼DiskManager()

```
DiskManager::∼DiskManager ( )
```

DiskManager destructor. All pages have been written to disk prior to this call.

Precondition

None

Postcondition

The metadata for each relation/index file is written to the appropriate Unix file via _flushDiskFileInfo().

See also

DiskManager::_flushDiskFileInfo()

4.10.3 Member Function Documentation

4.10.3.1 allocatePage()

```
PageId DiskManager::allocatePage (
            FileId file_id )
```

Allocates a Page to the file which corresponds to file_id.

Precondition

A FileId in file_map is provided as input and there is enough space in the corresponding Unix file.

Postcondition

If unused_pages set of the corresponding DiskFileInfo struct is not empty, then a PageNum in the set is popped, and is used with the file_id to construct an appropriate PageId, which is returned. If the unused↩ _pages set of the corresponding DiskFileInfo struct is empty, and capacity is less than MAX_PAGE_NUM, then it is used with the file_id to construct an appropriate PageId, which is returned. Capacity is incremented and the size of underlying Unix file is increased by PAGE_SIZE. The updated metadata about the file is not immediately written to the file to minimize disk I/O.

Parameters

file←↩ _id	A FileId corresponding to the file in which a page will be allocated.

Returns

PageId of the allocated page.

Exceptions

InsufficientSpaceDiskMgr	If file_id is in file_map and of the file is equal to MAX_CAPCITY.
InvalidFileIdDiskMgr	If file_id is not in file_map.
DiskErrorDiskMgr	If file operation fails.

4.10.3.2 createFile()

```
void DiskManager::createFile (
             FileId file_id )
```

Creates a new Unix file, opens the fstream, initializes DiskFileInfo object, and adds <FileId, DiskFileInfo∗> pair to fileMap. Adds header data to Unix file via SerializedFileInfo.

Precondition

A FileId that is added to catalog and is not in file_map.

Postcondition

A new Unix file is created, the fstream is opened, DiskFileInfo is initialized for that file and <FileId, DiskFile←↩ Info∗> pair is added to file_map.

Parameters

file←↩ _id	A FileId of the file to be created.

Exceptions

FileIdAlreadyExistDiskMgr	If file_id is already in file_map.
FileAlreadyExistDiskMgr	If Unix file already exists.
DiskErrorDiskMgr	If file operation fails.

4.10.3.3 deallocatePage()

```
void DiskManager::deallocatePage (
            PageId page_id )
```

Deallocates page by adding its offest to unused_pages in the appropriate DiskFileInfo struct.

Precondition

PageId of an allocated Page is provided as input.

Postcondition

page_id.page_num is added to the unused_pages set in the appropriate DiskFileInfo struct. The updated metadata about the file is not immediately written to the file due to performance reasons.

Parameters

page←_id	A PageId corresponding to the page to be deallocated.

Exceptions

InvalidPageNumDiskMgr	If page_id.page_num is out of range or if the page_id.page_num is in unused_pages.
InvalidFileIdDiskMgr	If page_id.file_id is not in file_map.

4.10.3.4 getCapacity()

```
std::uint32_t DiskManager::getCapacity (
            FileId file_id )
```

Get method for the capacity of a file.

Precondition

A FileId in file_map is provided as input.

Postcondition

capacity of the corresponding file is returned.

Parameters

file←_id	A FileId corresponding to the Unix file to get capacity.

Returns

capacity of the corresponding file.

Exceptions

InvalidFileIdDiskMgr	If file_id is not in file_map.

4.10.3.5 getSize()

```
std::uint32_t DiskManager::getSize (
            FileId file_id )
```

Get method for size of a file.

Precondition

A FileId in file_map is provided as input.

Postcondition

size of the corresponding file is returned.

Parameters

file←↩ _id	A FileId corresponding to the Unix file to get size.

Returns

size of the corresponding file.

Exceptions

InvalidFileIdDiskMgr	If file_id is not in file_map.

4.10.3.6 isValidPage()

```
bool DiskManager::isValidPage (
            PageId page_id )
```

Checks if the page of a given pageId is valid.

Precondition

PageId of a Page is provided as input.

Postcondition

If page_id.file_id is not in file_map or page_id.page_num is out of range or is in unused_pages, false is returned. Otherwise true is returned.

Parameters

page↩ _id	A PageId for validity to be checked.

Returns

bool indicating whether the page is valid.

4.10.3.7 printFile()

```
void DiskManager::printFile (
            FileId file_id )
```

THIS METHOD IS FOR DEBUGGING ONLY. Prints contents of a file inlcuding FileId, size, capacity, and contents of each page.

Precondition

A FileId in file_map is provided as input.

Postcondition

The content of the corresponding file,including size, capacity and content of each page is printed.

Parameters

file↩ _id	A FileId corresponding to the Unix file to be printed

Exceptions

InvalidFileIdDiskMgr	If file_id is not in file_map.
DiskErrorDiskMgr	If file operation fails.

4.10.3.8 readPage()

```
void DiskManager::readPage (
            PageId page_id,
            Page * page )
```

Reads the page data from the Unix file into the Page object pointer.

Precondition

PageId of an allocated Page, and a valid Page object pointer are provided as input.

Postcondition

The data of the page is read from the Unix file into the Page object.

Parameters

page←_id	A PageId of the page to be read from Unix file.
page	A Page pointer to be initialized.

Exceptions

InvalidFileIdDiskMgr	If page_id.file_id is not in file_map.
InvalidPageNumDiskMgr	If page_id.page_num is out of range or if the page_id.page_num is in unused_pages.
DiskErrorDiskMgr	If file operation fails.

4.10.3.9 removeFile()

```
void DiskManager::removeFile (
            FileId file_id )
```

Deletes the Unix file corresponding to file_id, and removes the <FileId, DiskFileInfo∗> key-value pair from fileMap.

Precondition

A FileId in file_map is provided as input.

Postcondition

The corresponding file's fstream is closed, the Unix file is removed, and the <FileId, DiskFileInfo∗> pair is removed from file_map.

Parameters

file←↩ _id	A file_id of the file to be removed.

Exceptions

InvalidFileIdDiskMgr	If file_id is not in file_map.

4.10.3.10 writePage()

```
void DiskManager::writePage (
            PageId page_id,
            Page * page )
```

Writes the page data of the given Page object at the right offset in the appropriate Unix file.

Precondition

PageId of an allocated Page, and a valid Page reference are provided as input.

Postcondition

The Page object data is written to the Unix file at the right offset.

Parameters

page←↩ _id	A PageId to write Page object data in the appropriate Unix file.
page	A Page∗ object containing data to be written to the appropriate Unix file.

Exceptions

InvalidFileIdDiskMgr	If page_id.file_id is not in file_map.
InvalidPageNumDiskMgr	If page_id.page_num is out of range or if the page_id.page_num is in unused_pages.
DiskErrorDiskMgr	If file operation fails.

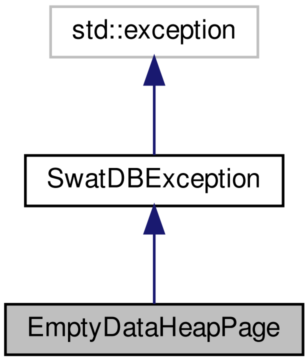
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/diskmgr.h

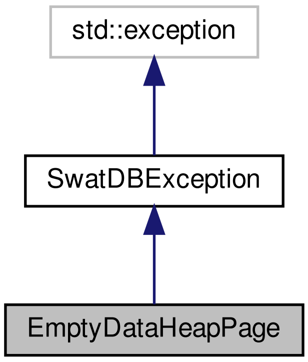
4.11 EmptyDataHeapPage Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for EmptyDataHeapPage:



Collaboration diagram for EmptyDataHeapPage:



Public Member Functions

- EmptyDataHeapPage ()

  Constructor.
- ∼EmptyDataHeapPage () throw ()

  Destructor.

Additional Inherited Members

4.11.1 Detailed Description

EmptyDataHeapPage is thrown by HeapPage if the record to be inserted has data field with size 0.

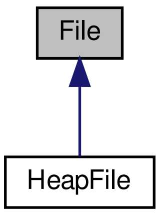
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/exceptions.h

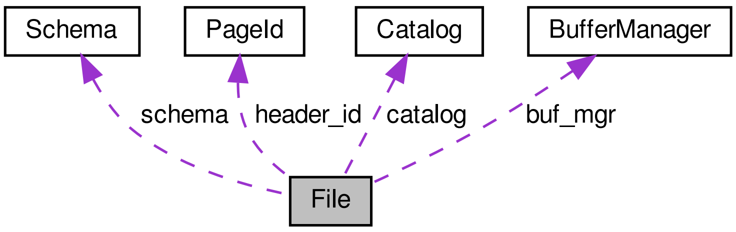
4.12 File Class Reference

```
#include <file.h>
```

Inheritance diagram for File:



Collaboration diagram for File:



Public Member Functions

- File (Catalog ∗catalog, BufferManager ∗buf_mgr, Schema ∗schema)

    Constructor. Initializes common state associated with every file, including catalog, buf_mgr, and schema.
- virtual ∼File ()

    Destructor. Data members are cleaned up, but none of the dynamically allocated data members are not deleted. The underlying Unix file is not deleted.
- virtual void createHeader ()=0

    Allocates and initializes the header. Is a virtual method to be overridden at each derived class.
- virtual void flushHeader ()=0

    Flushes Header Page to disk. Is a virtual method to be overridden at each derived class.
- FileId getMyFid ()

    Returns the FileId of the File.
- Schema ∗ getSchema ()

    Returns the schema of the File.
- PageId getHeaderId ()

    Returns Header PageId.

Protected Member Functions

- void _setMyFid (FileId file_id)

    Sets the file_id and the schema fields of this File.

Protected Attributes

- FileId file_id
- Catalog ∗ catalog
- BufferManager ∗ buf_mgr
- Schema ∗ schema
- PageId header_id

Friends

- class **Catalog**
- class **FileManager**

4.12.1 Detailed Description

SwatDB File Class. The base class for all file-type objects in the system. A File is used to represent a relation or index in the system. This base class inludes state and methods that are common to every type of file in the system.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 File()

```
File::File (
            Catalog ∗ catalog,
            BufferManager ∗ buf_mgr,
            Schema ∗ schema )
```

Constructor. Initializes common state associated with every file, including catalog, buf_mgr, and schema.

Parameters

catalog	Pointer to the SwatDB Catalog. Needed for getting file and idex relation files and schema.
buf_mgr	Pointer to the SwatDB Buffer Manager. Needed for de/allocating Pages, for getting and flushing Pages.
schema.	Pointer to Schema for the File.

Precondition

Input paramaters (pointers) are all valid.

Postcondition

The data members of File are initialized. file_id and header_id, however, are not set.

4.12.3 Member Function Documentation

4.12.3.1 _setMyFid()

```
void File::_setMyFid (
            FileId file_id )  [protected]
```

Sets the file_id and the schema fields of this File.

NOTE: this is called by

1. Catalog.createFile so that a File object can be added to the Catalog as part of an entry before the Catalog has determined its fileId (our solution to a chicken and egg problem).
2. FileManager._loadFile via its constructor when initializing SwatDB from saved db state, we add entries to the Catalog without Files, and the File manager creates File ∗ for them and updates each file's fid field after creation (could just have another constructor to handle this case)

Precondition

The given FileId is valid.

Postcondition

file_id is set to the given value.

Parameters

file←_id	The File's FileId.

4.12.3.2 getHeaderId()

```
PageId File::getHeaderId ( )
```

Returns Header PageId.

Precondition

None.

Postcondition

Header PageId of the File is returned.

Returns

The Header PageId.

4.12.3.3 getMyFid()

```
FileId File::getMyFid ( )
```

Returns the FileId of the File.

Returns

The FileId of the File.

4.12.3.4 getSchema()

```
Schema* File::getSchema ( )
```

Returns the schema of the File.

Returns

schema of the File.

4.12.4 Member Data Documentation

4.12.4.1 buf_mgr

`BufferManager* File::buf_mgr [protected]`

Pointer to the SwatDB BufferManager

4.12.4.2 catalog

`Catalog* File::catalog [protected]`

Pointer to the SwatDB Catalog

4.12.4.3 file_id

`FileId File::file_id [protected]`

The File's FileId.

4.12.4.4 header_id

`PageId File::header_id [protected]`

PageId for the header Page of the File.

4.12.4.5 schema

`Schema* File::schema [protected]`

Pointer to the Schema associated with this File. (this is just for fast access to a file's Schema vs. going through the Catalog every time this is needed)

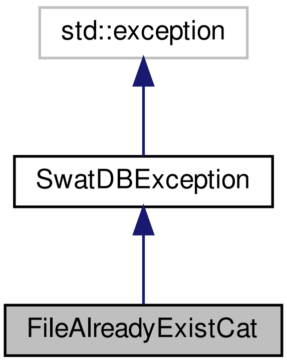
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/file.h

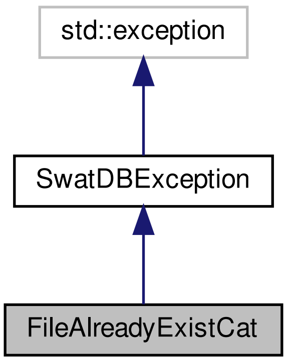
4.13 FileAlreadyExistCat Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for FileAlreadyExistCat:



Collaboration diagram for FileAlreadyExistCat:



Public Member Functions

- FileAlreadyExistCat (const std::string &filename)

  Constructor.
- ∼FileAlreadyExistCat () throw ()

  Destructor.
- std::string getFileName () const throw ()

  Returns the File name of the file that already exists.

Additional Inherited Members

4.13.1 Detailed Description

Catalog Exceptions. FileAlreadyExistCat is thrownn by Catalog on index or relation create if the underlying file name for the index or relation is already used.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 FileAlreadyExistCat()

```
FileAlreadyExistCat::FileAlreadyExistCat (
            const std::string & filename )  [explicit]
```

Constructor.

Parameters

file.	File name of the file that already exists.

4.13.3 Member Function Documentation

4.13.3.1 getFileName()

```
std::string FileAlreadyExistCat::getFileName ( ) const throw )
```

Returns the File name of the file that already exists.

Returns

File name of the file that already exists.

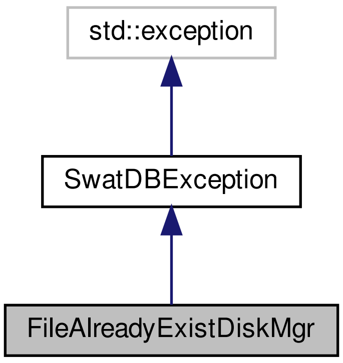
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/exceptions.h

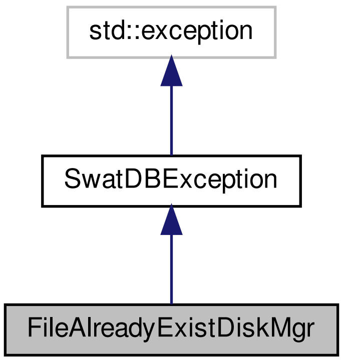
4.14 FileAlreadyExistDiskMgr Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for FileAlreadyExistDiskMgr:



Collaboration diagram for FileAlreadyExistDiskMgr:



Public Member Functions

- FileAlreadyExistDiskMgr (const std::string &file)

  Constructor.
- ∼FileAlreadyExistDiskMgr () throw ()

  Destructor.
- std::string getFileName () const throw ()

  Returns the file name of the file that already exists.

Additional Inherited Members

4.14.1 Detailed Description

FileAlreadyExistDiskMgr is thrown by DiskManager if Unix file already exists.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 FileAlreadyExistDiskMgr()

```
FileAlreadyExistDiskMgr::FileAlreadyExistDiskMgr (
            const std::string & file )  [explicit]
```

Constructor.

Parameters

file.	File name of the file that already exists.

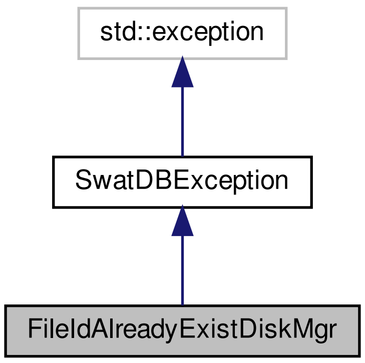
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/exceptions.h

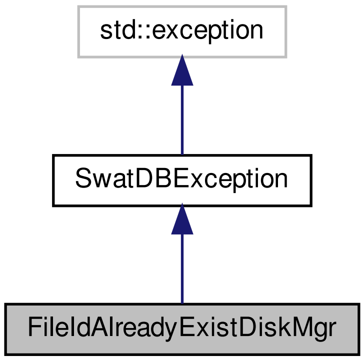
4.15 FileIdAlreadyExistDiskMgr Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for FileIdAlreadyExistDiskMgr:



Collaboration diagram for FileIdAlreadyExistDiskMgr:



Public Member Functions

- FileIdAlreadyExistDiskMgr (FileId file_id)

    Constructor.
- ∼FileIdAlreadyExistDiskMgr () throw ()

    Destructor.
- FileId getFileId () const throw ()

    Returns the FileId of the file that already exists.

Additional Inherited Members

4.15.1 Detailed Description

FileIdAlreadyExistDiskMgr is thrown by DiskManager if the file identified by FileId already exists.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 FileIdAlreadyExistDiskMgr()

```
FileIdAlreadyExistDiskMgr::FileIdAlreadyExistDiskMgr (
            FileId file_id )  [explicit]
```

Constructor.

Parameters

file_id.	FileId of the file that already exists.

4.15.3 Member Function Documentation

4.15.3.1 getFileId()

```
FileId FileIdAlreadyExistDiskMgr::getFileId ( ) const throw )
```

Returns the FileId of the file that already exists.

Returns

FileId of the file that already exists.

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

- /home/koh2/swatdb/SwatDB/include/exceptions.h

4.16 FileManager Class Reference

```
#include <filemgr.h>
```

Public Member Functions

- FileManager (Catalog ∗cat, BufferManager ∗buf_mgr, bool load=false)

  Creates SwatDB FileManager, the interface to the File layer.
- ∼FileManager ()

  Destructor.
- FileId createRelation (std::string name, Schema ∗schema, CatType type, std::string file_name)

  Creates a new Relation in the system. Adds an entry for the new file to the Catalog and creates the underlying storage for it with the DiskManager.
- void removeRelation (std::string name)

  Deletes relation from the system and removes its underlying storage from the database.
- void removeFile (FileId file_id)

  Deletes a File or Index from the system and removes its underlying storage from the database.
- File ∗ getRelation (std::string name)

  Retuns File object identified by the relation name.
- File ∗ getFile (FileId file_id)

  Retuns File object identified by the given FileId.

Protected Member Functions

- void _removeAllFiles ()

  Removes all Files in the system and their underlying storage from the database.
- void _closeAllFiles ()

  Flushes and closes all underlying files in the system, saving db state.

Friends

- class **SwatDB**

4.16.1 Detailed Description

SwatDB FileManager Class. The interface to the file layer of the system: manages Files and Indexes.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 FileManager()

```
FileManager::FileManager (
            Catalog * cat,
            BufferManager * buf_mgr,
            bool load = false )
```

Creates SwatDB FileManager, the interface to the File layer.

NOTE: load paramameter has a defaulf vlaue of false, which means that the constructor can be called just passing argument values for the first 2 parameter when a false value for load is desired.

Precondition

Input parameter objects (cat and buf_mgr) are valid.

Postcondition

If load is true, FileManager is constructed for SwatDB tat is loaded from existing files on disk. File objects are created for every relation and index file in the system and corresponding Catalog entries are updated with them. If load is false, FileManager is constructed for a new empty SwatDB.

Parameters

cat	Pointer to the SwatDB Catalog.
buf_mgr	Pointer to the SwatDB BufferManager.
load	bool indicating whether Files shoud be loaded from the existing SwatDB.

4.16.2.2 ∼FileManager()

`FileManager::∼FileManager ( )`

Destructor.

SwatDB controls how all the database state is saved/removed at shut down and deletes the BufferManager and Catalog objects in its destructor

4.16.3 Member Function Documentation

4.16.3.1 _closeAllFiles()

`void FileManager::_closeAllFiles ( ) [protected]`

Flushes and closes all underlying files in the system, saving db state.

NOTE: This method is called on shutdown of the SwatDB, saving the db state.

Precondition

None.

Postcondition

Header page of each File is flushed to disk, but the underlying Unix files are not removed, effectively saving the current db state to disk.

4.16.3.2 _removeAllFiles()

`void FileManager::_removeAllFiles ( ) [protected]`

Removes all Files in the system and their underlying storage from the database.

NOTE: This method is called on shutdown of the SwatDB, removing the db state.

Precondition

None. `@post Underlying files that store the db state of the SwatDB are removed.`

4.16.3.3 createRelation()

```
FileId FileManager::createRelation (
             std::string name,
             Schema * schema,
             CatType type,
             std::string file_name )
```

Creates a new Relation in the system. Adds an entry for the new file to the Catalog and creates the underlying storage for it with the DiskManager.

Precondition

A file with the requested relation or index name and type and underlying file name is not already defined in the system.

Postcondition

A new file or index of the specified type is added to the system. A new file or index object is created, its entry is added to the SwatDB catalog, and a Unix file to store its data is created on disk by the DiskManager.

Parameters

name	The Relation name in the system.
schema	A pointer to a Schema object associated with the file. (for RelationFile types).
type	The specific type of file or index to create.
file_name	The name of the disk file in which the new file object will be stored.

Returns

FileId of the newly created file or index, or INVALID_FILE_ID if an error occurs.

Exceptions

FileAlreadyExistCat	if the file_name already exists in the database.
RelationAlreadyExistCat	if a relation named name already exists in the database.
IndexAlreadyExistsCat	if an index named name already exists in the database.
InvalidFileTypeFileMgr	if the passed CatType is invalid or not supported.

4.16.3.4 getFile()

```
File* FileManager::getFile (
             FileId file_id )
```

Retuns File object identified by the given FileId.

Precondition

The File, identified by the FileId already exists.

Postcondition

File∗ is returned.

Parameters

file←↩ _id	FileId of the file to be returned.

Exceptions

FileNotFoundFileMgr	if the file identified by the FileId does not exist.

4.16.3.5 getRelation()

```
File* FileManager::getRelation (
             std::string name )
```

Retuns File object identified by the relation name.

Precondition

The realtion of the given name already exists.

Postcondition

File∗ is returned.

Parameters

name	Relation ame of the relation file to be returned.

Exceptions

FileNotFoundFileMgr	if the file identified by the name does not exist.

4.16.3.6 removeFile()

```
void FileManager::removeFile (
             FileId file_id )
```

Deletes a File or Index from the system and removes its underlying storage from the database.

Precondition

A valid FileId is provided as input.

Postcondition

The file identified by the given FileId is removed from the database.

Parameters

file_id	FileId of the file to be deleted from the database.

Exceptions

FileNotFoundFileMgr	if the given FileId does not exist in the system.

4.16.3.7 removeRelation()

```
void FileManager::removeRelation (
             std::string name )
```

Deletes relation from the system and removes its underlying storage from the database.

Precondition

A valid relation name is provided as input.

Postcondition

The relation of the given name is removed from the database.

Parameters

name	Name of the relation to be deleted from the database.

Exceptions

FileNotFoundFileMgr	if the file identified by the FileId does not exist.

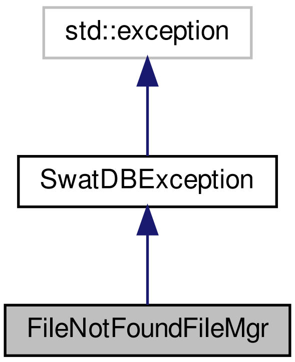
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/filemgr.h

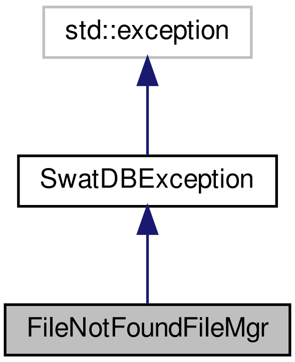
4.17 FileNotFoundFileMgr Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for FileNotFoundFileMgr:



Collaboration diagram for FileNotFoundFileMgr:



Public Member Functions

- FileNotFoundFileMgr ()

  Constructor.
- ∼FileNotFoundFileMgr () throw ()

  Destructor.

Additional Inherited Members

4.17.1 Detailed Description

FileNotFoundFileMgr is thrown by FileManager if file is not found.

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

- /home/koh2/swatdb/SwatDB/include/exceptions.h

4.18 Frame Class Reference

```
#include <bufmgr.h>
```

Public Member Functions

- Frame ()
  Constructor. Calls resetFrame to reset the Frame.
- ∼Frame ()
  Destructor.
- void resetFrame ()
  Resets the metadata of the Frame.
- void loadFrame (PageId page_id)
  Updates the Frame data according to the loaded Page.

Friends

- class BufferManager

4.18.1 Detailed Description

Frame Class which holds metadata about corresponding Page in the buffer pool.

4.18.2 Member Function Documentation

4.18.2.1 loadFrame()

```
void Frame::loadFrame (
            PageId page_id )
```

Updates the Frame data according to the loaded Page.

Precondition

None.

Postcondition

Frame data is updated. pin_count is set to 1, page_id is set to page_id parameter, and valid is set to true.

Parameters

page←↩ _id	PageId of the Page that is loaded to the Frame

4.18.2.2 resetFrame()

```
void Frame::resetFrame ( )
```

Resets the metadata of the Frame.

Precondition

None.

Postcondition

page_id is set to INVALID_PAGE_ID. pin_count is set to 0. valid, dirty, and ref_bit are all set to false.

4.18.3 Friends And Related Function Documentation

4.18.3.1 BufferManager

```
friend class BufferManager  [friend]
```

BufferManager has access to private data members of each Frame.

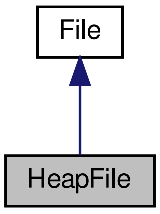
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/bufmgr.h

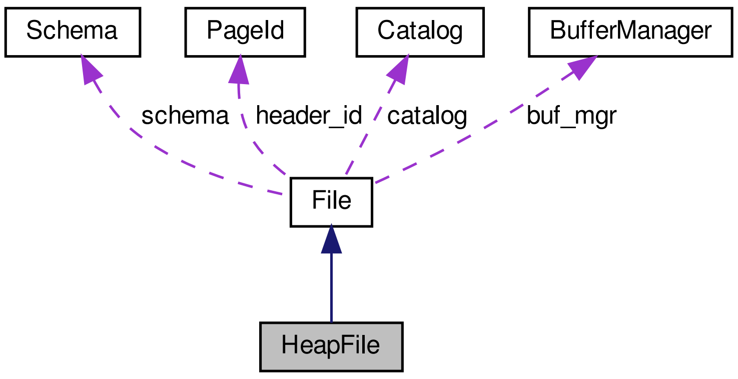
4.19 HeapFile Class Reference

```
#include <heapfile.h>
```

Inheritance diagram for HeapFile:



Collaboration diagram for HeapFile:



Public Member Functions

- HeapFile (Catalog ∗catalog, BufferManager ∗buf_mgr, Schema ∗schema)

  Constructor for HeapFile class.
- ∼HeapFile ()

  HeapFile destructor.
- void createHeader ()

  Allocates and initializes the header Page of the file.
- void flushHeader ()

  Flushes header Page to disk.
- RecordId insertRecord (Record record)

Inserts a Record into the HeapFile.
- void getRecord (RecordId record_id, Record ∗record)

  Sets the record_data of the given Record pointer to the data corresponding to the given RecordId.
- void updateRecord (RecordId record_id, Record record)

  Updates the record data in the HeapFile identified by the given RecordId to the data in the provided Record.
- void deleteRecord (RecordId record_id)

  Deletes the Record identified by the given RecordId.
- HeapFileHeader getHeader ()

  THIS METHOD IS FOR DEBUGGING ONLY. Returns the current HeapFileHeader.

Friends

- class **HeapFileScanner**

Additional Inherited Members

4.19.1 Detailed Description

SwatDB HeapFile Class. Represents heap file in the system. It consists of doubly linked list of HeapPage and is an unsorted collection of records. Provides various methods, including inserting, modifying and retrieving records.

4.19.2 Constructor & Destructor Documentation

4.19.2.1 HeapFile()

```
HeapFile::HeapFile (
            Catalog * catalog,
            BufferManager * buf_mgr,
            Schema * schema )
```

Constructor for HeapFile class.

Precondition

A valid Catalog pointer and BufferManager pointer are provided as inputs.

Postcondition

HeapFile object is constructed. Catalog and BufferManager data members are set to the provided inputs. Other values are initialized after construction.

Parameters

catalog	Pointer to the SwatDB Catalog object.
buf_mgr	Pointer to the SwatDB BufferManager object.
schema	Pointer to this file's Schema.

4.19.2.2 ∼HeapFile()

```
HeapFile::∼HeapFile ( )  [inline]
```

HeapFile destructor.

Precondition

None.

Postcondition

The HeapFile object is destroyed. The file on disk is not removed.

4.19.3 Member Function Documentation

4.19.3.1 createHeader()

```
void HeapFile::createHeader ( )  [virtual]
```

Allocates and initializes the header Page of the file.

Precondition

There is sufficient space for a Page in buffer pool. The file with the corresponding FileId is already created.

Postcondition

A Page is allocated and the header_id field is initialized to the PageId of the allcated Page. The free and full fields are initialized to INVALID_PAGE_NUM and free_size and full_size are initialized to 0.

Exceptions

InsufficientSpaceBufMgr	If there is not enough space in the bufferpool.

Implements File.

4.19.3.2 deleteRecord()

```
void HeapFile::deleteRecord (
             RecordId record_id )
```

Deletes the Record identified by the given RecordId.

Precondition

A valid RecordId is provided.

Postcondition

Deletes the Record identified by the given RecordId. If the Page was in the list of full pages, and it is no longer full after deletion, then the Page is moved to the list of free pages. If the Page is empy after deletion, it is completely removed from any list, released, and deallocated. The header Page is updated appropriately. All pages pinned during the operation are released at the end or before exception is thrown.

Parameters

record←↩ _id	RecordId identifying the record data to be deleted.

Exceptions

InvalidSchemaHeapFile	If the given Record's Schema does not match that of the HeapFile(compare pointers).
InvalidPageIdBufMgr	If the PageNum of the given RecordId is not valid.
InvalidSlotIdHeapPage	If the SlotId of the given RecordId is not valid.

4.19.3.3 flushHeader()

```
void HeapFile::flushHeader ( )  [virtual]
```

Flushes header Page to disk.

Precondition

There is sufficient space for the header Page in buffer pool. header_id is valid.

Postcondition

Header Page of the File is flushed to disk.

Exceptions

InsufficientSpaceBufMgr	If there is not enough space in the bufferpool for the header Page.
InvalidPageIdBufMgr	If the header_id of the File is not valid.

Implements File.

4.19.3.4 getRecord()

```
void HeapFile::getRecord (
            RecordId record_id,
            Record * record )
```

Sets the record_data of the given Record pointer to the data corresponding to the given RecordId.

Precondition

A valid RecordId and a Record object pointer with a Schema matching that of the HeapFile.

Postcondition

The given Record pointer's data field is initialized to the data identified by the given RecordId. All pages pinned during the operation are released at the end or before exeption is thrown.

Parameters

record←_id	RecordId of the record_data to be retrieved.
record	Record pointer to store the retrieved data.

Exceptions

InvalidSchemaHeapFile	If the given Record pointer's Schema does not match that of the HeapFile (compares pointers).
InvalidPageIdBufMgr	If the PageNum of the given RecordId is not valid.
InvalidSlotIdHeapPage	If the SlotId of the given RecordId is not valid.

4.19.3.5 insertRecord()

```
RecordId HeapFile::insertRecord (
            Record record )
```

Inserts a Record into the HeapFile.

Precondition

A valid Record object with a Schema matching that of the HeapFile is provided as input. There is some Page into which the Record can be inserted.

Postcondition

The Record is inserted into some Page that belongs to the HeapFile. If there is enough space on some Page in the list of free pages, then the Record is inserted there. If there is not enough space on any Page in the list of free pages, then a new Page is allocated. The Record is inserted into this Page. If the the Page is full after inserting the Record, the Page is moved to the list of full pages. Otherwise, the Page is added to/remains in free list. The header Page is updated appropriately. All pages pinned during the operation are released at the end or before exeption is thrown.

Parameters

record	Record to be inserted into the HeapFile.

Returns

RecordId RecordId of the inserted Record.

Exceptions

InvalidSchemaHeapFile	If the given Record's Schema does not match that of the HeapFile (compares pointers).
InsufficientSpaceHeapPage	If the given Record's data exceeds the MAXIMUM_RECORD_SIZE.
InsufficientSpaceHeapFile	If the number of pages (including the header) in the HeapFile exceeds MAX_PAGE_NUM.

4.19.3.6 updateRecord()

```
void HeapFile::updateRecord (
            RecordId record_id,
            Record record )
```

Updates the record data in the HeapFile identified by the given RecordId to the data in the provided Record.

Precondition

A valid RecordId and a Record object with a Schema matching that of the HeapFile are provided as inputs. There is enough space in the HeapPage containing the Record identified by RecordId for the data in the provided record.

Postcondition

The Record data of the Record identified by the RecordId is replaced with the data of the provided Record. If the "full" state of the apge changes after the update, it is moved from one list to the other (free to full or full to free). The header Page is updated appropriately. All pages pinned during the operation are released at the end or before exeption is thrown.

Parameters

record↩ *_id*	RecordId identifying the record data to be updated.
record	Record object containing data to overwrite the record data identified by the given RecordId.

Exceptions

InvalidSchemaHeapFile	If the given Record's Schema does not match that of the HeapFile (compares pointers).

Exceptions

InvalidPageIdBufMgr	if the PageNum of the given RecordId is not valid.
InvalidSlotIdHeapPage	if the SlotId of the given RecordId is not valid.
InsufficientSpaceHeapPage	if there is not enough space for the updated record in the corresponding HeapPage.

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

- /home/koh2/swatdb/SwatDB/include/heapfile.h

4.20 HeapFileHeader Struct Reference

```
#include <heapfile.h>
```

Public Attributes

- PageNum free
- PageNum full
- std::uint32_t free_size
- std::uint32_t full_size
- std::uint32_t num_records

4.20.1 Detailed Description

Struct for the header metadata of HeapFile object. The header is casted on top of the first Page allocated to the file.

4.20.2 Member Data Documentation

4.20.2.1 free

```
PageNum HeapFileHeader::free
```

PageNum of the Page at the head of the linked list of free pages.

4.20.2.2 free_size

```
std::uint32_t HeapFileHeader::free_size
```

Number of pages in the linked list of free pages.

4.20.2.3 full

`PageNum HeapFileHeader::full`

PageNum of the Page at the head of the linked list of full pages.

4.20.2.4 full_size

`std::uint32_t HeapFileHeader::full_size`

Number of pages in the linked list of full pages.

4.20.2.5 num_records

`std::uint32_t HeapFileHeader::num_records`

Number of records in the HeapFile.

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

- /home/koh2/swatdb/SwatDB/include/heapfile.h

4.21 HeapFileScanner Class Reference

`#include <heapfile.h>`

Public Member Functions

- HeapFileScanner (HeapFile ∗file)

  Constructor.
- ∼HeapFileScanner ()

  Destructor.
- RecordId getNext (Record ∗record)

  Returns RecordId of the next Record in the HeapFile and initializes the given Record object to the data of the identified Record. Scans for records by iterating through the linked list of full pages, then the linked list of free pages.

4.21.1 Detailed Description

Scanner class for HeapFile. Scanner scans the given HeapFile object and returns the next RecordId and initiliazes the given Record Data whenever getNext is called. Returns INVALID_RECORD_ID if it reaches the end of the file.

4.21.2 Constructor & Destructor Documentation

4.21.2.1 HeapFileScanner()

```
HeapFileScanner::HeapFileScanner (
            HeapFile * file )
```

Constructor.

Precondition

Valid HeapFile∗ is provided as input.

Postcondition

HeapFile object is constructed with initialized data members. The cur_page is pinned if file is not empty.

Parameters

file	HeapFile object to be scanned.

4.21.2.2 ∼HeapFileScanner()

```
HeapFileScanner::∼HeapFileScanner ( )
```

Destructor.

Precondition

If the end of the File has not been reached, cur_page is pinned.

Postcondition

If the end of the File has not been reached, cur_page is released.

4.21.3 Member Function Documentation

4.21.3.1 getNext()

```
RecordId HeapFileScanner::getNext (
            Record * record )
```

Returns RecordId of the next Record in the HeapFile and initializes the given Record object to the data of the identified Record. Scans for records by iterating through the linked list of full pages, then the linked list of free pages.

Precondition

None

Postcondition

RecordId of the next Record is returned and data of the given Record object is initialized to that of the Record identified by the RecordId. If there are no more records in the HeapFile, INVALID_RECORD_ID is returned. cur_page is the Page that is being scanned and is pinned. Once the end of cur_page is reached, it is the next Page in the File is pinned (if there is next Page) and cur_page is unpinned. cur_pid and cur_page are updated accordingly. If the end of the File is reached, no Page is pinned by the HeapFileScanner.

Returns

Next valid RecordId. INVALID_RECORD_ID if the end of the file is reached.

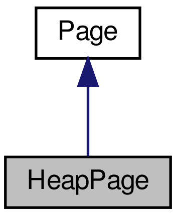
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/heapfile.h

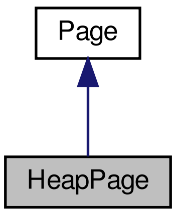
4.22 HeapPage Class Reference

```
#include <heappage.h>
```

Inheritance diagram for HeapPage:



Collaboration diagram for HeapPage:



Public Member Functions

- HeapPage ()=delete

  Constructor.
- ∼HeapPage ()
- void initializeHeader ()

  Initializes header information after the Page is allocated.
- void setNext (PageNum page_num)

  Sets next_page to the given PageNum.
- void setPrev (PageNum page_num)

  Set prev_page to the given PageNum.
- PageNum getNext ()

  Getter for next_page.
- PageNum getPrev ()

  Getter for prev_page.
- std::uint32_t getFreeSpace ()

  Getter for the amount of free space free on the Page.
- bool isFull ()

  bool function indicating whether the Page is full.
- bool isEmpty ()

  bool function indicating whether the Page is empty.
- SlotId insertRecord (Data ∗record_data)

  Inserts given record data to the Page.
- void getRecord (SlotId slot_id, Data ∗data)

  Gets the record identified by SlotId.
- void deleteRecord (SlotId slot_id)

  Deletes Record identified by SlotId.
- void updateRecord (SlotId slot_id, Data ∗record_data)

  Updates Record identified by SlotId.
- HeapPageHeader getHeader ()

  THIS METHOD IS FOR DEBUGGING ONLY. Returns this HeapPage's header information.
- SlotInfo getSlotInfo (SlotId slot_id)

  THIS METHOD IS FOR DEBUGGING ONLY. Returns SlotInfo struct of the given SlotId.
- std::uint32_t getInvalidNum ()

  THIS METHOD IS FOR DEBUGGING ONLY. Returns number of invalid slots in the HeapPage.
- void printHeapPageState ()

  THIS METHOD IS FOR DEBUGGING ONLY. Prints the current state of the HeapPage.

Friends

- class **HeapPageScanner**

Additional Inherited Members

4.22.1 Detailed Description

SwatDB HeapPage Class. HeapPage inherits from base Page class and instantiates heap page, collection of which, form HeapFile.

4.22.2 Constructor & Destructor Documentation

4.22.2.1 HeapPage()

```
HeapPage::HeapPage ( )  [delete]
```

Constructor.

Constructor should never be called. It should be always the case that base class Page constructor is called by the BufferManager when initializing bufferpool. HeapPage Pointer is casted to whatever Page pointer returned by the BufferManager

4.22.2.2 ∼HeapPage()

```
HeapPage::∼HeapPage ( )  [inline]
```

Destructor

4.22.3 Member Function Documentation

4.22.3.1 deleteRecord()

```
void HeapPage::deleteRecord (
            SlotId slot_id )
```

Deletes Record identified by SlotId.

Precondition

A valid SlotId is provided as input

Postcondition

If free_space_end is less than the slot offset of the corresponding slot info, the records to the left of the Record to be deleted are shifted to right by the size of the deleted Record, effectively compactifying space occupied by the records. The slot offset of all shifted Records are updated accordingly. free_space_end is incremented by size of the deleted Record. page header size is decremented. If SlotId is equal to page header capacity - 1, free_space_begin is decremented by sizeof(SlotInfo) and page header capacity is decremented. Else, slot offset of the slot previously occupied by deleted Record is set to INVALID_SLOT_OFFSET.

Parameters

slot←_id	SlotId of the Record to be deleted.

Exceptions

InvalidSlotIdHeapPage	If SlotId is invalid (SlotId is out of range or SlotInfo of the given SlotId has INVALID_SLOT_OFFSET).

4.22.3.2 getFreeSpace()

```
std::uint32_t HeapPage::getFreeSpace ( )
```

Getter for the amount of free space free on the Page.

Precondition

None.

Postcondition

The amount of free space in bytes on the Page is returned. If there is no free slot, sizeof(SlotInfo) is subtracted from the free space returned (accounting extra space occupied if new slot is allocated). If free space is less than sizeof(SlotInfo), 0 is returned.

Returns

Amount of free space in bytes on the Page.

4.22.3.3 getNext()

```
PageNum HeapPage::getNext ( )
```

Getter for next_page.

Precondition

None.

Postcondition

next_page of the page header is returned.

Returns

PageNum of the next Page.

4.22.3.4 getPrev()

```
PageNum HeapPage::getPrev ( )
```

Getter for prev_page.

Precondition

None.

Postcondition

prev_page of the Page header is returned.

Returns

PageNum of the previous Page.

4.22.3.5 getRecord()

```
void HeapPage::getRecord (
             SlotId slot_id,
             Data * data )
```

Gets the record identified by SlotId.

Precondition

A valid SlotId is provided as input and a valid Data∗ with capacity that is greater than maximum size of any Record stored in the Page is provided. The inserted record must not have data size 0.

Postcondition

The data char array of Data object is appropriately initialized by copying the Record stored in Page to the data array of the object and the size of Data object is set to the number of bytes that are copied.

Parameters

slot↩ _id	SlotId of the Record to be retrieved

Exceptions

InvalidSlotIdHeapPage	If SlotId is out of range or SlotInfo of the given SlotId has INVALID_SLOT_OFFSET.
EmptyDataHeapPage	If the size of the given data is 0.

4.22.3.6 initializeHeader()

```
void HeapPage::initializeHeader ( )
```

Initializes header information after the Page is allocated.

Precondition

None

Postcondition

Page header is initialized such that prev_page and next_page are set to INVALID_PAGE_NUM, free_space←
_begin is set to sizeof(HeapPageHeader), free_space_end is set to PAGE_SIZE, size and capacity are set to 0.

4.22.3.7 insertRecord()

```
SlotId HeapPage::insertRecord (
            Data * record_data )
```

Inserts given record data to the Page.

Precondition

There is enough space in the Page and valid Data∗ is provided as input

Postcondition

A slot is allocated for inserting the record data. If there is enough space ((free_space_end - free_space_begin) is greater than size of record data), then data is copied to its size number of bytes in front of free_space_end. Page header size is incremented and capacity is also incremented if a new slot is allocated for the Record. free_space_end is decremented by the size of the record data. SlotInfo offset and length are updated. If new slot was allocated (equal to capacity -1), but there is not enough space for record_data, page_header capacity and free_space_begin are decremented back before throwing an exception (or free space was checked before a slot was allocated).

Parameters

record_data	Data∗ of the record data to be inserted.

Returns

SlotId of the slot the Record is inserted to.

Exceptions

InsufficientSpaceHeapPage	If there is not enough space for the Record where sizeof(SlotInfo) is also conisdered if pageheader size is equal to page header capacity.

4.22.3.8 isEmpty()

```
bool HeapPage::isEmpty ( )
```

bool function indicating whether the Page is empty.

Precondition

None.

Postcondition

true is returned if size is 0. Else false is returned.

Returns

bool indicating whether the Page is empty.

4.22.3.9 isFull()

```
bool HeapPage::isFull ( )
```

bool function indicating whether the Page is full.

Precondition

None.

Postcondition

true is returned if used space/PAGE_SIZE exceeds MAX_HEAP_PAGE_LOAD and false is returned otherwise.

Returns

bool indicating whether the Page is full.

4.22.3.10 setNext()

```
void HeapPage::setNext (
             PageNum page_num )
```

Sets next_page to the given PageNum.

Precondition

None.

Postcondition

next_page is set to the given PageNum.

Parameters

page_num	PageNum of the next Page.

4.22.3.11 setPrev()

```
void HeapPage::setPrev (
            PageNum page_num )
```

Set prev_page to the given PageNum.

Precondition

None.

Postcondition

prev_page is set to the given PageNum.

Parameters

page_num	PageNum of the previous Page.

4.22.3.12 updateRecord()

```
void HeapPage::updateRecord (
            SlotId slot_id,
            Data * record_data )
```

Updates Record identified by SlotId.

Precondition

A valid SlotId is provided as input and there is enough space in the Page for the updated Record. Valid Data∗ is provided as input.

Postcondition

Record stored in the slot identified by the given SlotId is updated to the given Record. Previous Record is not modified if there is not enough space for the updated Record.

Parameters

slot_id	SlotId of the Record to be updated.
record_data	Data∗ of the updated Record.

Exceptions

InsufficientSpaceHeapPage	If there is not enough space for updated Record.
InvalidSlotIdHeapPage	If SlotId is invalid (SlotId is out of range or SlotInfo of the given SlotId has INVALID_SLOT_OFFSET).

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

- /home/koh2/swatdb/SwatDB/include/heappage.h

4.23 HeapPageHeader Struct Reference

```
#include <heappage.h>
```

Public Attributes

- PageNum prev_page
- PageNum next_page
- std::uint32_t free_space_begin
- std::uint32_t free_space_end
- std::uint32_t size
- std::uint32_t capacity

4.23.1 Detailed Description

Struct for the header metadata of HeapPage object. The header is casted on top of the Page data array, from the beginning of the array. Must follow 64bit alignment.

4.23.2 Member Data Documentation

4.23.2.1 capacity

```
std::uint32_t HeapPageHeader::capacity
```

Number of allocated slots.

4.23.2.2 free_space_begin

```
std::uint32_t HeapPageHeader::free_space_begin
```

Offset where free space begins in the Page.

4.23.2.3 free_space_end

```
std::uint32_t HeapPageHeader::free_space_end
```

Offset where free space ends in the Page.

4.23.2.4 next_page

```
PageNum HeapPageHeader::next_page
```

PageNum of next Page in the HeapFile.

4.23.2.5 prev_page

```
PageNum HeapPageHeader::prev_page
```

PageNum of previous Page in the HeapFile.

4.23.2.6 size

```
std::uint32_t HeapPageHeader::size
```

Number of valid/filled slots.

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

- /home/koh2/swatdb/SwatDB/include/heappage.h

4.24 HeapPageScanner Class Reference

```
#include <heappage.h>
```

Public Member Functions

- HeapPageScanner (HeapPage ∗page)

    Constructor.
- ∼HeapPageScanner ()

    Destructor.
- SlotId getNext ()

    Returns SlotId of the next valid slot.
- void reset (HeapPage ∗page)

    Resets the scanner, so it could be used for another Page.

4.24.1 Detailed Description

Scanner class for scanning HeapPage.

4.24.2 Constructor & Destructor Documentation

4.24.2.1 HeapPageScanner()

```
HeapPageScanner::HeapPageScanner (
            HeapPage * page )
```

Constructor.

Precondition

Valid HeapPage∗ is provided as input. The given Page is pinned.

Postcondition

HeapPageScanner is constructed with initialized data members. page is still pinned.

Parameters

page	HeapPage object to be scanned.

4.24.3 Member Function Documentation

4.24.3.1 getNext()

```
SlotId HeapPageScanner::getNext ( )
```

Returns SlotId of the next valid slot.

Precondition

page is pinned.

Postcondition

SlotId of the next valid slot is returned. Current slot field is set to the next SlotId. Current slot field may be incremented by more than one. If the scanner reaches the end of the slot directory, INVALID_SLOT_ID is returned. page is still pinned.

Returns

Next valid SlotId. INVALID_SLOT_ID if the end of the Page is reached. page is still pinned.

4.24.3.2 reset()

```
void HeapPageScanner::reset (
            HeapPage * page )
```

Resets the scanner, so it could be used for another Page.

Precondition

The new Page is pinned.

Postcondition

page is set to the provided HeapPage∗ and current slot is reset to 0. The new Page is still pinned.

Parameters

page	HeapPage object to reset to.

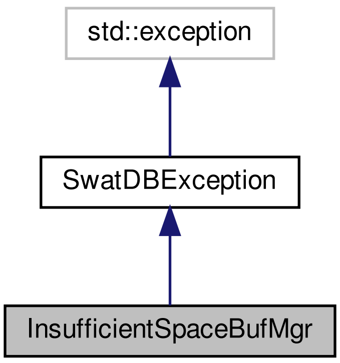
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/heappage.h

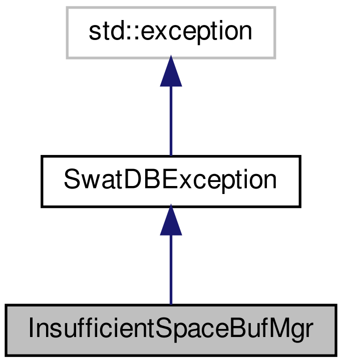
4.25 InsufficientSpaceBufMgr Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for InsufficientSpaceBufMgr:



Collaboration diagram for InsufficientSpaceBufMgr:



Public Member Functions

- InsufficientSpaceBufMgr ()

  Constructor.
- ∼InsufficientSpaceBufMgr () throw ()

  Destructor.

Additional Inherited Members

4.25.1 Detailed Description

InsufficientSpaceBufMgr is thrown by BufferManager if there is not enough space in the buffer pool.

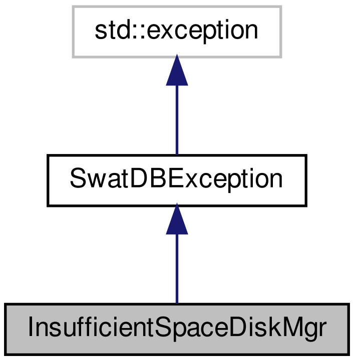
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/exceptions.h

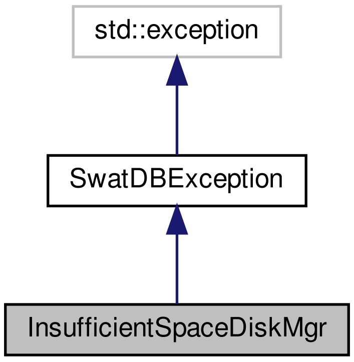
4.26 InsufficientSpaceDiskMgr Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for InsufficientSpaceDiskMgr:



Collaboration diagram for InsufficientSpaceDiskMgr:



Public Member Functions

- InsufficientSpaceDiskMgr (FileId file_id)

  Constructor.
- ∼InsufficientSpaceDiskMgr () throw ()

  Destructor.
- FileId getFileId () const throw ()

  Returns the FileId of the file with insufficient space.

Additional Inherited Members

4.26.1 Detailed Description

InsufficientSpaceDiskMgr is thrown by DiskManager if there is not enough space in file.

4.26.2 Constructor & Destructor Documentation

4.26.2.1 InsufficientSpaceDiskMgr()

```
InsufficientSpaceDiskMgr::InsufficientSpaceDiskMgr (
            FileId file_id )  [explicit]
```

Constructor.

Parameters

file_id.	FileId of the file with insuffucient space.

4.26.3 Member Function Documentation

4.26.3.1 getFileId()

```
FileId InsufficientSpaceDiskMgr::getFileId ( ) const throw )
```

Returns the FileId of the file with insufficient space.

Returns

FileId of the file with insufficient space.

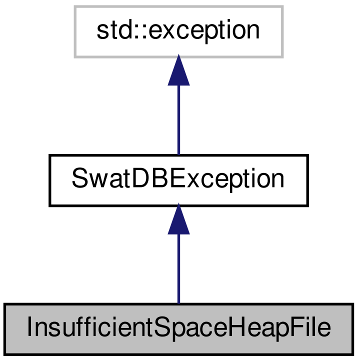
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/exceptions.h

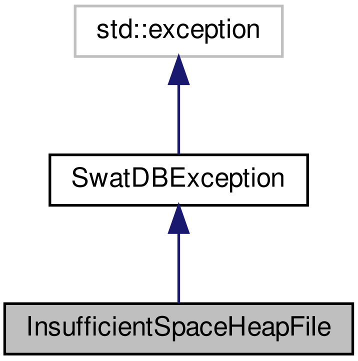
4.27 InsufficientSpaceHeapFile Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for InsufficientSpaceHeapFile:



Collaboration diagram for InsufficientSpaceHeapFile:



Public Member Functions

- InsufficientSpaceHeapFile ()

  Constructor.
- ∼InsufficientSpaceHeapFile () throw ()

  Destructor.

Additional Inherited Members

4.27.1 Detailed Description

InvalidSchemaHeapFile is thrown by HeapFile if the total number of Pages of the HeapFile is MAX_PAGE_NUM.

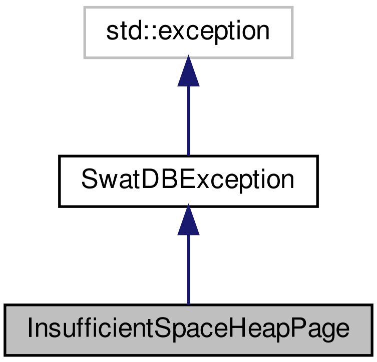
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/exceptions.h

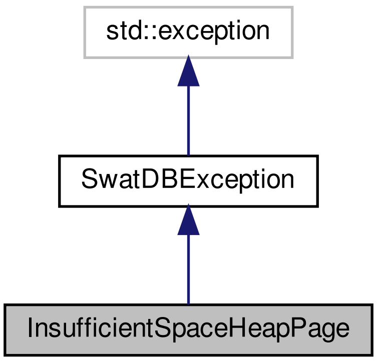
4.28 InsufficientSpaceHeapPage Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for InsufficientSpaceHeapPage:



Collaboration diagram for InsufficientSpaceHeapPage:



Public Member Functions

- InsufficientSpaceHeapPage ()

    Constructor.
- ∼InsufficientSpaceHeapPage () throw ()

    Destructor.

Additional Inherited Members

4.28.1 Detailed Description

InsufficientSpaceHeapPage is thrown by HeapPage if space is insufficient.

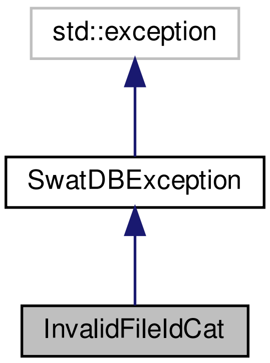
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/exceptions.h

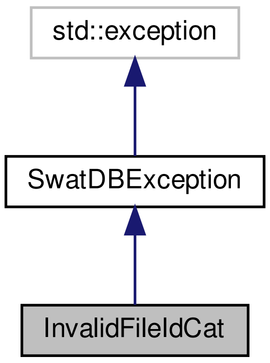
4.29 InvalidFileIdCat Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for InvalidFileIdCat:



Collaboration diagram for InvalidFileIdCat:



Public Member Functions

- InvalidFileIdCat (FileId file_id)

  Constructor.
- ∼InvalidFileIdCat () throw ()

  Destructor.
- FileId getFileId () const throw ()

  Returns the invalid FileId.

Additional Inherited Members

4.29.1 Detailed Description

InvalidFileIdCat is thrown by Catalog if FileId is invalid.

4.29.2 Constructor & Destructor Documentation

4.29.2.1 InvalidFileIdCat()

```
InvalidFileIdCat::InvalidFileIdCat (
            FileId file_id )  [explicit]
```

Constructor.

Parameters

file_id.	Invalid FileId.

4.29.3 Member Function Documentation

4.29.3.1 getFileId()

```
FileId InvalidFileIdCat::getFileId ( ) const throw )
```

Returns the invalid FileId.

Returns

Invalid FileId.

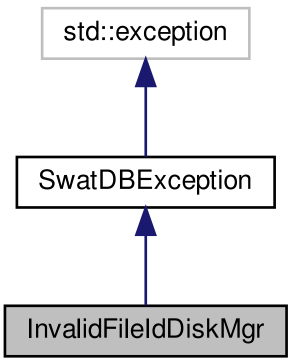
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/exceptions.h

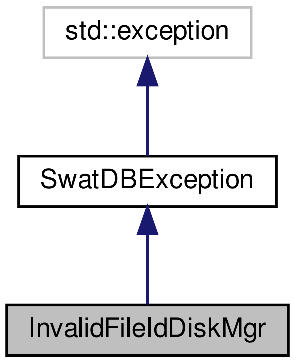
4.30 InvalidFileIdDiskMgr Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for InvalidFileIdDiskMgr:



Collaboration diagram for InvalidFileIdDiskMgr:



Public Member Functions

- InvalidFileIdDiskMgr (FileId file_id)

  Constructor.
- ∼InvalidFileIdDiskMgr () throw ()

  Destructor.
- FileId getFileId () const throw ()

  Returns the invalid FileId.

Additional Inherited Members

4.30.1 Detailed Description

DiskManager Exceptions InvalidFileIdDiskMgr is thrown by DiskManager if FileId is invalid.

4.30.2 Constructor & Destructor Documentation

4.30.2.1 InvalidFileIdDiskMgr()

```
InvalidFileIdDiskMgr::InvalidFileIdDiskMgr (
             FileId file_id )  [explicit]
```

Constructor.

Parameters

file_id.	Invalid FileId.

4.30.3 Member Function Documentation

4.30.3.1 getFileId()

```
FileId InvalidFileIdDiskMgr::getFileId ( ) const throw )
```

Returns the invalid FileId.

Returns

Invalid FileId.

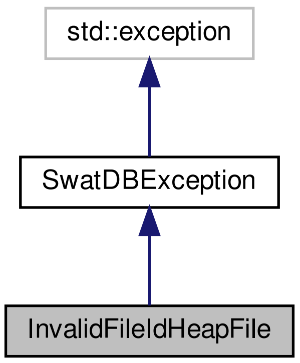
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/exceptions.h

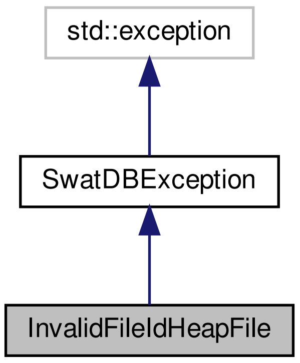
4.31 InvalidFileIdHeapFile Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for InvalidFileIdHeapFile:



Collaboration diagram for InvalidFileIdHeapFile:



Public Member Functions

- InvalidFileIdHeapFile (FileId file_id)

    Constructor.
- ∼InvalidFileIdHeapFile () throw ()

    Destructor.
- FileId getFileId () const throw ()

    Returns the invalid FileId.

Additional Inherited Members

4.31.1 Detailed Description

HeapFile Exceptions. InvalidFileIdHeapFile is thrown by Heapfile if FileId does not match that of the HeapFile.

4.31.2 Constructor & Destructor Documentation

4.31.2.1 InvalidFileIdHeapFile()

```
InvalidFileIdHeapFile::InvalidFileIdHeapFile (
            FileId file_id )  [explicit]
```

Constructor.

Parameters

file_id.	Invalid FileId.

4.31.3 Member Function Documentation

4.31.3.1 getFileId()

```
FileId InvalidFileIdHeapFile::getFileId ( ) const throw )
```

Returns the invalid FileId.

Returns

Invalid FileId.

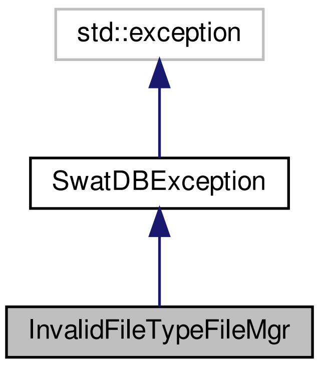
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/exceptions.h

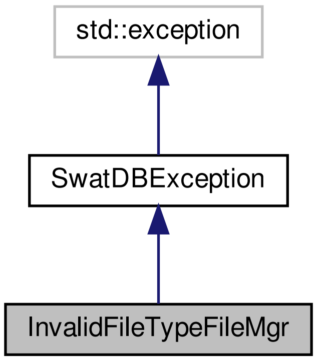
4.32 InvalidFileTypeFileMgr Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for InvalidFileTypeFileMgr:



Collaboration diagram for InvalidFileTypeFileMgr:



Public Member Functions

- InvalidFileTypeFileMgr (CatType file_type)

  Constructor.
- ∼InvalidFileTypeFileMgr () throw ()

  Destructor.
- CatType getFileType () const throw ()

  Returns the PageId of the pinned Page.

Additional Inherited Members

4.32.1 Detailed Description

FileManager Exceptions InvalidFileTypeFileMgr is thrown by FileManager if file type is inconsistent.

4.32.2 Constructor & Destructor Documentation

4.32.2.1 InvalidFileTypeFileMgr()

```
InvalidFileTypeFileMgr::InvalidFileTypeFileMgr (
             CatType file_type )  [explicit]
```

Constructor.

Parameters

file_type.	File type of the file.

4.32.3 Member Function Documentation

4.32.3.1 getFileType()

```
CatType InvalidFileTypeFileMgr::getFileType ( ) const throw )
```

Returns the PageId of the pinned Page.

Returns

PageId of the pinned Page.

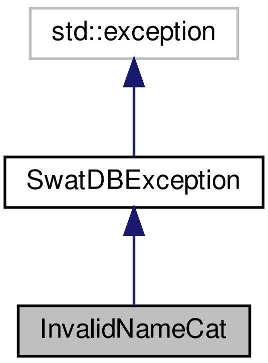
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/exceptions.h

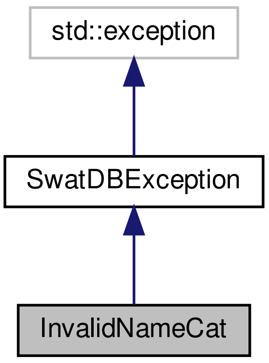
4.33 InvalidNameCat Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for InvalidNameCat:



Collaboration diagram for InvalidNameCat:



Public Member Functions

- InvalidNameCat (const std::string &name)

  Constructor.
- ∼InvalidNameCat () throw ()

  Destructor.
- std::string getName () const throw ()

  Returns the invalid relation name.

Additional Inherited Members

4.33.1 Detailed Description

InvalidNameCat is thrown by Catalog if Relation Name is not valid.

4.33.2 Constructor & Destructor Documentation

4.33.2.1 InvalidNameCat()

```
InvalidNameCat::InvalidNameCat (
            const std::string & name )  [explicit]
```

Constructor.

Parameters

name	Invalid Relation name.

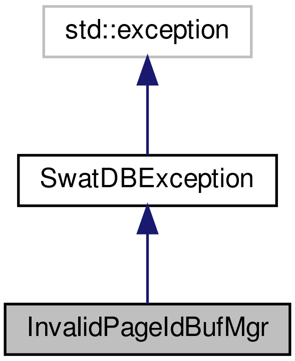
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/exceptions.h

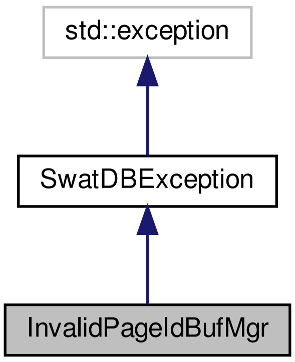
4.34 InvalidPageIdBufMgr Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for InvalidPageIdBufMgr:



Collaboration diagram for InvalidPageIdBufMgr:



Public Member Functions

- InvalidPageIdBufMgr (PageId page_id)

  Constructor.
- ∼InvalidPageIdBufMgr () throw ()

  Destructor.
- PageId getPageId () const throw ()

  Returns the invalid PageId.

Additional Inherited Members

4.34.1 Detailed Description

BufferManager Exceptions InvalidPageIdBufMgr is thrown by BufferManager if PageId is invalid.

4.34.2 Constructor & Destructor Documentation

4.34.2.1 InvalidPageIdBufMgr()

```
InvalidPageIdBufMgr::InvalidPageIdBufMgr (
            PageId page_id )  [explicit]
```

Constructor.

Parameters

page_id.	Invalid PageId.

4.34.3 Member Function Documentation

4.34.3.1 getPageId()

```
PageId InvalidPageIdBufMgr::getPageId ( ) const throw )
```

Returns the invalid PageId.

Returns

Invalid PageId.

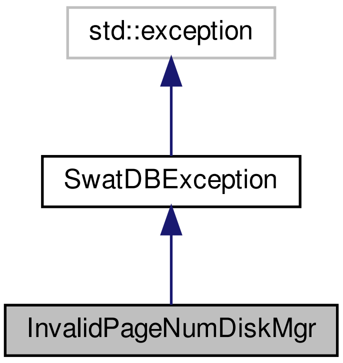
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/exceptions.h

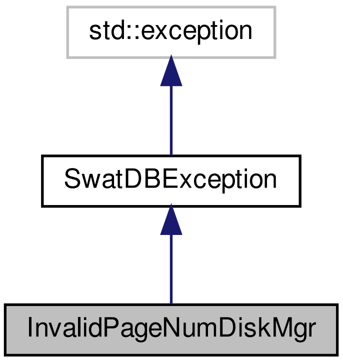
4.35 InvalidPageNumDiskMgr Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for InvalidPageNumDiskMgr:



Collaboration diagram for InvalidPageNumDiskMgr:



Public Member Functions

- InvalidPageNumDiskMgr (PageNum page_num)

  Constructor.
- ∼InvalidPageNumDiskMgr () throw ()

  Destructor.
- PageNum getPageNum () const throw ()

  Returns the invalid PageNum.

Additional Inherited Members

4.35.1 Detailed Description

InvalidPageNumDiskMgr is thrown by DiskManager if PageNum is invalid.

4.35.2 Constructor & Destructor Documentation

4.35.2.1 InvalidPageNumDiskMgr()

```
InvalidPageNumDiskMgr::InvalidPageNumDiskMgr (
            PageNum page_num )  [explicit]
```

Constructor.

Parameters

page_num.	Invalid PageNum.

4.35.3 Member Function Documentation

4.35.3.1 getPageNum()

```
PageNum InvalidPageNumDiskMgr::getPageNum ( ) const throw )
```

Returns the invalid PageNum.

Returns

Invalid PageNum.

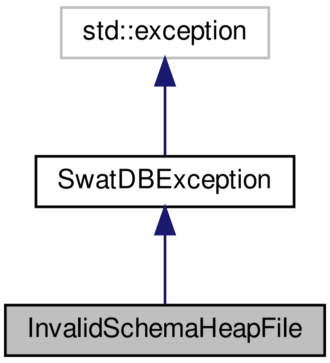
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/exceptions.h

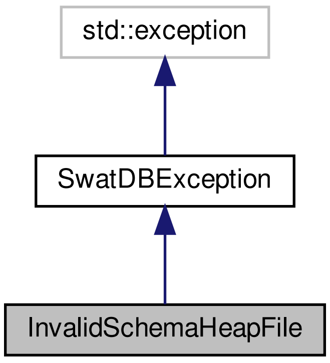
4.36 InvalidSchemaHeapFile Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for InvalidSchemaHeapFile:



Collaboration diagram for InvalidSchemaHeapFile:



Public Member Functions

- InvalidSchemaHeapFile ()

  Constructor.
- ∼InvalidSchemaHeapFile () throw ()

  Destructor.

Additional Inherited Members

4.36.1 Detailed Description

InvalidSchemaHeapFile is thrown by HeapFile if Schema does not match that of the HeapFile.

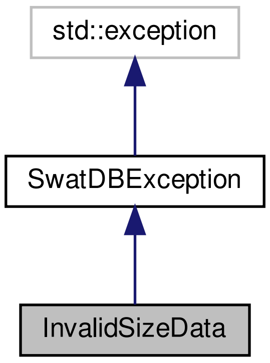
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/exceptions.h

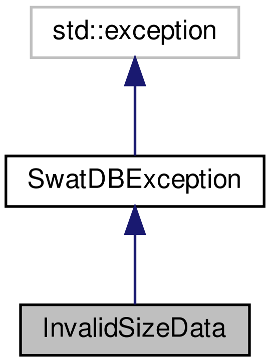
4.37 InvalidSizeData Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for InvalidSizeData:



Collaboration diagram for InvalidSizeData:



Public Member Functions

- InvalidSizeData ()

  Constructor.
- ∼InvalidSizeData () throw ()

  Destructor.

Additional Inherited Members

4.37.1 Detailed Description

Data Exceptions. InvalidSizeData is thrown by Data if size exceeds capacity.

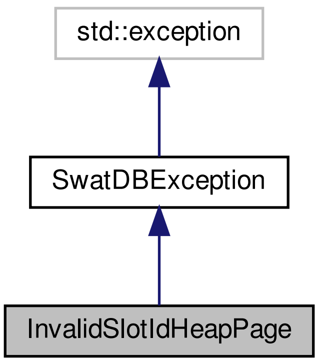
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/exceptions.h

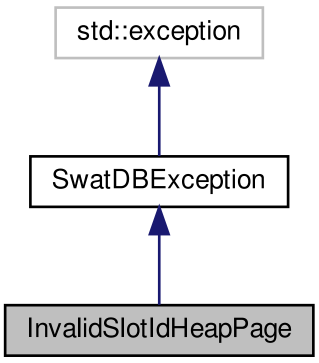
4.38 InvalidSlotIdHeapPage Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for InvalidSlotIdHeapPage:



Collaboration diagram for InvalidSlotIdHeapPage:



Public Member Functions

- InvalidSlotIdHeapPage (SlotId slot_id)

    Constructor.
- ∼InvalidSlotIdHeapPage () throw ()

    Destructor.
- SlotId getSlotId () const throw ()

    Returns the invalid SlotId.

Additional Inherited Members

4.38.1 Detailed Description

HeapPage Exceptions. InvalidSlotIdHeapPage is thrown by HeapPage if SlotId is invalid.

4.38.2 Constructor & Destructor Documentation

4.38.2.1 InvalidSlotIdHeapPage()

```
InvalidSlotIdHeapPage::InvalidSlotIdHeapPage (
            SlotId slot_id )  [explicit]
```

Constructor.

Parameters

slot_id.	Invalid SlotId.

4.38.3 Member Function Documentation

4.38.3.1 getSlotId()

```
SlotId InvalidSlotIdHeapPage::getSlotId ( ) const throw )
```

Returns the invalid SlotId.

Returns

Invalid SlotId.

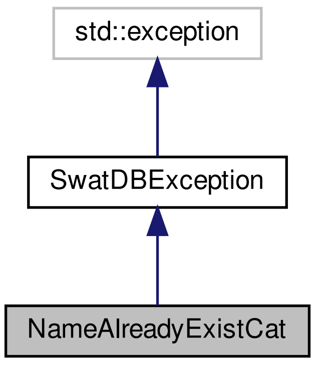
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/exceptions.h

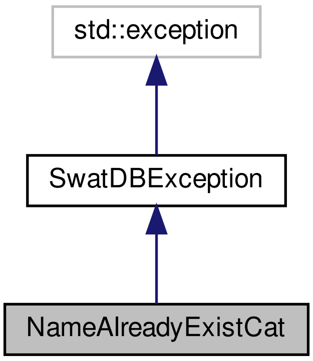
4.39 NameAlreadyExistCat Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for NameAlreadyExistCat:



Collaboration diagram for NameAlreadyExistCat:



Public Member Functions

- NameAlreadyExistCat (const std::string &relname)

  Constructor.
- ∼NameAlreadyExistCat () throw ()

  Destructor.
- std::string getName () const throw ()

  Returns the name of the relation that already exists.

Additional Inherited Members

4.39.1 Detailed Description

NameAlreadyExistCat is thrown by Catalog on a new relation or index addition to the system if the relation name already exits.

4.39.2 Constructor & Destructor Documentation

4.39.2.1 NameAlreadyExistCat()

```
NameAlreadyExistCat::NameAlreadyExistCat (
             const std::string & relname )   [explicit]
```

Constructor.

Parameters

file.	Name of the relation that already exists.

4.39.3 Member Function Documentation

4.39.3.1 getName()

```
std::string NameAlreadyExistCat::getName ( ) const throw )
```

Returns the name of the relation that already exists.

Returns

Name of the relation that already exists.

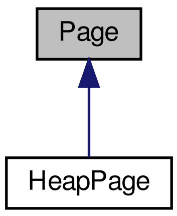
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/exceptions.h

4.40 Page Class Reference

```
#include <page.h>
```

Inheritance diagram for Page:



Public Member Functions

- Page ()

  Constructor.
- ∼Page ()

  Destructor.
- char ∗ getData ()

  Get function for the data char array.

Protected Attributes

- char data [PAGE_SIZE]

    char array that stores the page data. Derived classes could map appropriate structures on it.

Friends

- class **DiskManager**

4.40.1 Detailed Description

SwatDB Page Class. Page is the basic unit of read/write operation on disk in the system. The data member is PAGE_SIZE char array, onto which other derived classes could map layer-specific structures and define access methods.

4.40.2 Member Function Documentation

4.40.2.1 getData()

```
char* Page::getData ( )
```

Get function for the data char array.

Precondition

None.

Postcondition

char∗ to the data char array is returned.

Returns

char∗ to the data char array.

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

- /home/koh2/swatdb/SwatDB/include/page.h

4.41 PageId Struct Reference

```
#include <swatdb_types.h>
```

Public Member Functions

- bool **operator==** (const PageId &other) const
- bool **operator!=** (const PageId &other) const

Public Attributes

- FileId **file_id**
- PageNum **page_num**

4.41.1 Detailed Description

Unique identifier of each file/index page in the system.

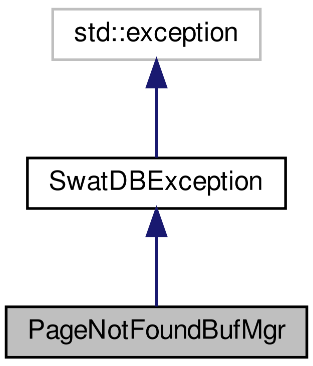
The documentation for this struct was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/swatdb_types.h

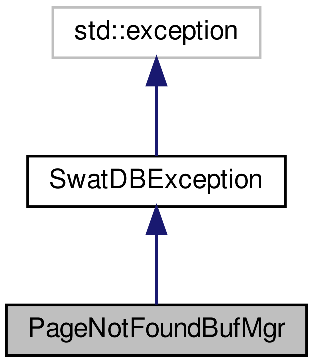
4.42 PageNotFoundBufMgr Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for PageNotFoundBufMgr:



Collaboration diagram for PageNotFoundBufMgr:



Public Member Functions

- PageNotFoundBufMgr (PageId page_id)

  Constructor.
- ∼PageNotFoundBufMgr () throw ()

  Destructor.
- PageId getPageId () const throw ()

  Returns the PageId of the Page not found.

Additional Inherited Members

4.42.1 Detailed Description

PageNotFoundBufMgr is thrown by BufferManager if PageId is not found in bufferpool.

4.42.2 Constructor & Destructor Documentation

4.42.2.1 PageNotFoundBufMgr()

```
PageNotFoundBufMgr::PageNotFoundBufMgr (
            PageId page_id )  [explicit]
```

Constructor.

Parameters

page_id.	PageId of the Page not found.

4.42.3 Member Function Documentation

4.42.3.1 getPageId()

```
PageId PageNotFoundBufMgr::getPageId ( ) const throw )
```

Returns the PageId of the Page not found.

Returns

PageId of the Page not found.

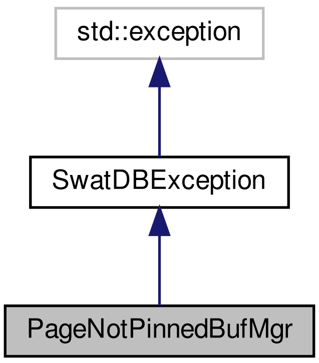
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/exceptions.h

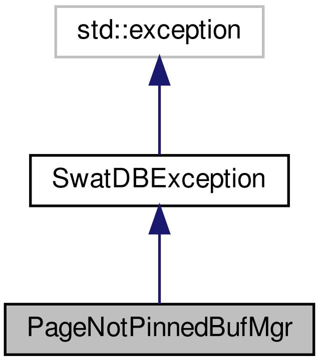
4.43 PageNotPinnedBufMgr Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for PageNotPinnedBufMgr:



Collaboration diagram for PageNotPinnedBufMgr:



Public Member Functions

- PageNotPinnedBufMgr (PageId page_id)

    Constructor.
- ∼PageNotPinnedBufMgr () throw ()

    Destructor.
- PageId getPageId () const throw ()

    Returns the PageId of the Page not pinned.

Additional Inherited Members

4.43.1 Detailed Description

PageNotPinnedBufMgr is thrown by BufferManager if Page is not pinned.

4.43.2 Constructor & Destructor Documentation

4.43.2.1 PageNotPinnedBufMgr()

```
PageNotPinnedBufMgr::PageNotPinnedBufMgr (
            PageId page_id )  [explicit]
```

Constructor.

Parameters

page_id.	PageId of the Page that is not pinned.

4.43.3 Member Function Documentation

4.43.3.1 getPageId()

```
PageId PageNotPinnedBufMgr::getPageId ( ) const throw )
```

Returns the PageId of the Page not pinned.

Returns

PageId of the Page not pinned.

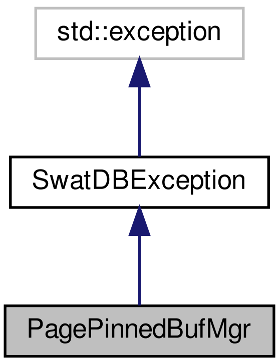
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/exceptions.h

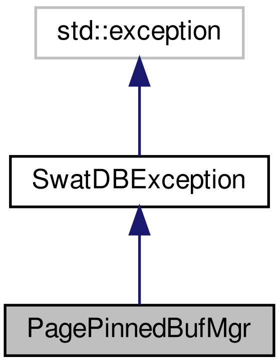
4.44 PagePinnedBufMgr Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for PagePinnedBufMgr:



Collaboration diagram for PagePinnedBufMgr:



Public Member Functions

- PagePinnedBufMgr (PageId page_id)

  Constructor.
- ∼PagePinnedBufMgr () throw ()

  Destructor.
- PageId getPageId () const throw ()

  Returns the PageId of the pinned Page.

Additional Inherited Members

4.44.1 Detailed Description

PagePinnedBufMgr is thrown by BufferManager if Page is pinned.

4.44.2 Constructor & Destructor Documentation

4.44.2.1 PagePinnedBufMgr()

```
PagePinnedBufMgr::PagePinnedBufMgr (
            PageId page_id )  [explicit]
```

Constructor.

Parameters

page_id.	PageId of the pinned Page.

4.44.3 Member Function Documentation

4.44.3.1 getPageId()

```
PageId PagePinnedBufMgr::getPageId ( ) const throw )
```

Returns the PageId of the pinned Page.

Returns

PageId of the pinned Page.

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

- /home/koh2/swatdb/SwatDB/include/exceptions.h

4.45 Record Class Reference

```
#include <record.h>
```

Public Member Functions

- Record ()
  Default constructor. Sets schema and record_data to nullptr.
- Record (Schema ∗schema, Data ∗record_data)
  Constructor given appropriate Data∗ and Schema∗.
- ∼Record ()
  Destructor.
- Schema ∗ getSchema ()
  Getter function for schema.
- Data ∗ getRecordData ()
  Getter for record_data.
- void setSchema (Schema ∗new_schema)
  Setter for schema.
- void setRecordData (Data ∗new_data)
  Setter for record_data.
- std::int32_t compare (Record other)
  Compare function for comparing 2 records.

4.45.1 Detailed Description

SwatDB Record Class. Record instantiates record in SwatDB. It consists of record data Data object, which is serialized in form of variable length record and Schema object, which allows appropriate access to the record data.

4.45.2 Constructor & Destructor Documentation

4.45.2.1 Record()

```
Record::Record (
            Schema * schema,
            Data * record_data )
```

Constructor given appropriate Data∗ and Schema∗.

Precondition

Valid Data∗ and Schema∗ are provided as input. The schema and the record data stored by record_data are consistent.

Postcondition

schema and record_data are set to the provided input.

Parameters

schema	Schema∗ of the Record.
record_data	Data∗ record_data that stores the Record data.

4.45.2.2 ∼Record()

```
Record::∼Record ( )  [inline]
```

Destructor.

Precondition

None.

Postcondition

Neither schema nor record_data is deleted. Both data members have to be deallocated manually by the user.

4.45.3 Member Function Documentation

4.45.3.1 compare()

```
std::int32_t Record::compare (
            Record other )
```

Compare function for comparing 2 records.

Precondition

Valid Record object is provided as input.

Postcondition

If the schemas are different, -1 is returned. If the record_data of the two records do not have the same size or do not have the identical array up to first size number of bytes, -1 is returned. Else, 0 is returned.

Parameters

other	The Record to be compared to.

Returns

-1 if the two records are not the same. 0 if the two records are the same.

4.45.3.2 getRecordData()

```
Data* Record::getRecordData ( )
```

Getter for record_data.

Precondition

None.

Postcondition

record_data is returned.

Returns

Data$*$ to the record_data of the Record.

4.45.3.3 getSchema()

```
Schema* Record::getSchema ( )
```

Getter function for schema.

Precondition

None.

Postcondition

schema is returned.

Returns

Schema∗ to the schema of the Record.

4.45.3.4 setRecordData()

```
void Record::setRecordData (
            Data * new_data )
```

Setter for record_data.

Precondition

Valid Data∗ is provided as an input.

Postcondition

record_data is set to new_data. The previous data is not deleted.

Parameters

new_data	New Data∗ schema is set to.

4.45.3.5 setSchema()

```
void Record::setSchema (
            Schema * new_schema )
```

Setter for schema.

Precondition

Valid Schema∗ is provided as an input.

Postcondition

schema is set to new_schema. The previous schema is not deleted.

Parameters

new_schema	New Schema∗ schema is set to.

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

- /home/koh2/swatdb/SwatDB/include/record.h

4.46 RecordId Struct Reference

```
#include <swatdb_types.h>
```

Public Member Functions

- bool **operator==** (const RecordId &other) const
- bool **operator!=** (const RecordId &other) const

Public Attributes

- PageNum **page_num**
- SlotId **slot_id**

4.46.1 Detailed Description

Unique identifier of each record in a single file.

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

- /home/koh2/swatdb/SwatDB/include/swatdb_types.h

4.47 Schema Class Reference

```
#include <schema.h>
```

Public Member Functions

- Schema ()

    Constructor.
- ~Schema ()

    Destructor.

4.47.1 Detailed Description

SwatDB Schema Class. NOT FULLY IMPLEMENTED. This is a class to represent the schema for a relation. Provides structure to Data of a Record, allowing for various access methods.

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

- /home/koh2/swatdb/SwatDB/include/schema.h

4.48 SlotInfo Struct Reference

```
#include <heappage.h>
```

Public Attributes

- uint32_t offset
- uint32_t length

4.48.1 Detailed Description

Struct for storing metadata of each slot in a Page. Array of SlotInfo forms the slot directory of the Page. Must be 64bit for alignment.

4.48.2 Member Data Documentation

4.48.2.1 length

```
uint32_t SlotInfo::length
```

Length of the record in the slot described by the SlotInfo

4.48.2.2 offset

`uint32_t SlotInfo::offset`

Offset at which slot is located. INVALID_SLOT_OFFSET if slot is not valid

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

- /home/koh2/swatdb/SwatDB/include/heappage.h

4.49 SwatDB Class Reference

`#include <swatdb.h>`

Public Member Functions

- SwatDB ()

  Constructor.
- SwatDB (std::string metadata_filename)

  Constructor to init SwatDB from a metadata file.
- ∼SwatDB ()

  Shutdown SwatDB.
- void setDestroyDB ()

  sets the DB to be destroyed on shutdown.
- void setSaveDB (std::string filename)

  sets the DB to be saved on shutdown.
- Catalog ∗ getCatalog ()

  Gets the SwatDB Catalog.
- DiskManager ∗ getDiskMgr ()

  Gets the SwatDB DiskManager.
- BufferManager ∗ getBufMgr ()

  Gets the SwatDB BufferManager.
- FileManager ∗ getFileMgr ()

  Gets the SwatDB FileManager.

4.49.1 Detailed Description

SwatDB. This is the class definition of the high-level SwatDB object that stores all state about the Swat DBMS system as it runs. This includes:

- The Catalog object of its relations and indices
- The Buffer Manager object
- The Disk Manager object
- The File Manager object
- And possibly others in the future (lock mgr, xact mgr)

It contains methods to boot SwatDB and to save or delete SwatDB, as well as to access layer manager objects for the different parts of the system.

On (or before) shutdown a caller should decide if they want to save or destroy the db and do so before invoking the destructor (the default is to destroy if neither have been invoked):

swatdb->setSaveDB() swatdb->setDestroyDB() delete swatdb delete swatdb

4.49.2 Constructor & Destructor Documentation

4.49.2.1 SwatDB() [1/2]

```
SwatDB::SwatDB ( )
```

Constructor.

Initialize/boot swatDB. Initializes an empty swatDB and creates default manager objects for all the layers

4.49.2.2 SwatDB() [2/2]

```
SwatDB::SwatDB (
            std::string metadata_filename )
```

Constructor to init SwatDB from a metadata file.

This method creates RelationFile and IndexFile objects associated with each stored file listed in the metadata. SwatDB creates the diskmgr before the file manager, and it opens the relation and index files on disk and checks that they exist.

Parameters

metadata_filename	the name of the input file containing information about the DB state to init SwatDB with

Exceptions

exceptions	from system layers with init errors

4.49.2.3 ∼SwatDB()

```
SwatDB::∼SwatDB ( )
```

Shutdown SwatDB.

If saveDB is not called before the destructor is invoked then the destructor calls destroyDB to remove all relation and index files from the system

Precondition

: SwatDB may have some relation and index files

Postcondition

: Based on if saveDB or destroyDB was invoked prior to the destructory either all index and relation files are saved and metadata about them is written out or all files and indexes removed from the system, and their associated storage on "disk" is also removed from the system. Any state created is removed from the system.

4.49.3 Member Function Documentation

4.49.3.1 setDestroyDB()

```
void SwatDB::setDestroyDB ( )
```

sets the DB to be destroyed on shutdown.

This method can be called prior to shuting down swatDB if the current state of swatDB does not want to be saved (destroy on exit is the default). This method does not actually remove db files from the system but sets a flag to trigger the right actions by the destructor on shutdown.

4.49.3.2 setSaveDB()

```
void SwatDB::setSaveDB (
            std::string filename )
```

sets the DB to be saved on shutdown.

This method should be called prior to shuting down swatDB to save the state of the db on shutdown. This method does not actually save the state at the point it is called but set up the system to save the state on shutdown

Parameters

filename	name of swatDB meta data file to which to save db metadata state on shutdown of swatDB

Precondition

: a file with filename does or does not already exist in the system (or an existing one's contents will be replaced)

Postcondition

: flag is set to trigger saving db metadata to file on shutdown

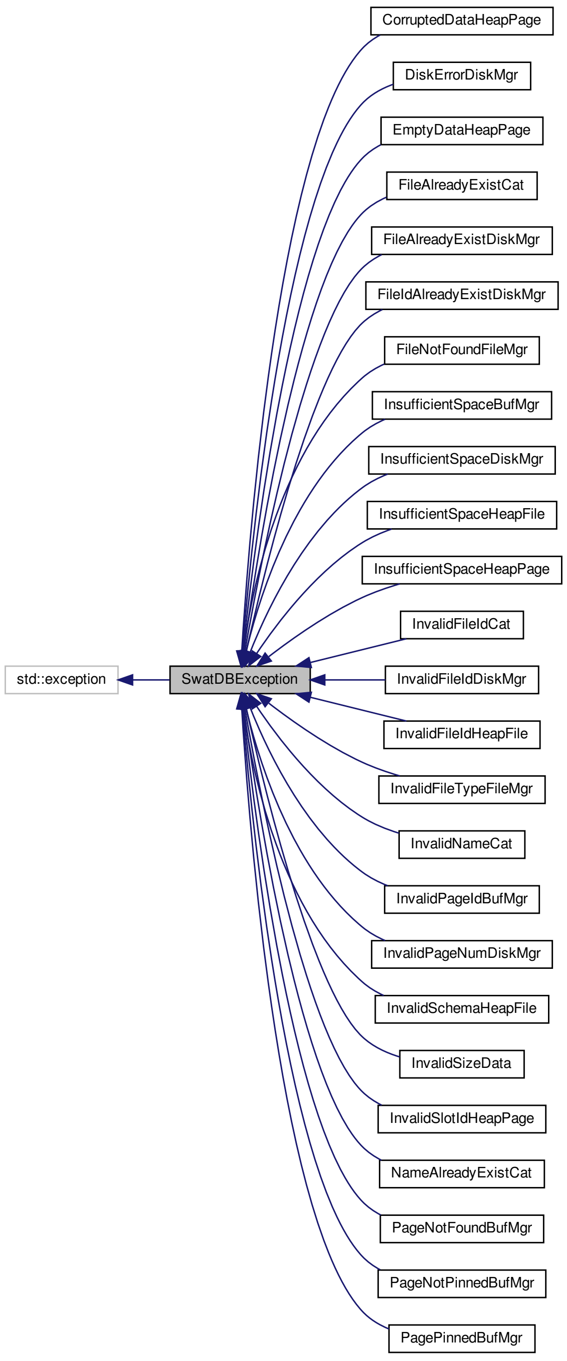
The documentation for this class was generated from the following file:

- /home/koh2/swatdb/SwatDB/include/swatdb.h

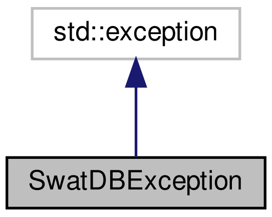
4.50 SwatDBException Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for SwatDBException:



Collaboration diagram for SwatDBException:



Public Member Functions

- SwatDBException (const std::string &msg)
- virtual ∼SwatDBException () throw ()
- virtual const char ∗ what () const throw ()

  Returns the error message of the exception.

Protected Attributes

- std::string message

4.50.1 Detailed Description

SwatDBException. SwatDBException is the base class for all exceptions thrown in SwatDB system. It is based on std::exception and other exceptions defined below inherit from this class.

4.50.2 Constructor & Destructor Documentation

4.50.2.1 SwatDBException()

```
SwatDBException::SwatDBException (
            const std::string & msg )  [inline], [explicit]
```

SwatDBExcpetion constructor. Requires string message

4.50.2.2 ∼SwatDBException()

```
virtual SwatDBException::∼SwatDBException ( ) throw )   [inline], [virtual]
```

SwatDBException destructor.

4.50.3 Member Function Documentation

4.50.3.1 what()

```
virtual const char* SwatDBException::what ( ) const throw )   [inline], [virtual]
```

Returns the error message of the exception.

Precondition

The exception is thrown and caught.

Postcondition

The error message is returned.

Returns

Pointer to the error message.

4.50.4 Member Data Documentation

4.50.4.1 message

```
std::string SwatDBException::message  [protected]
```

Message that this exception will output.

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

- /home/koh2/swatdb/SwatDB/include/exceptions.h

Index