

# **vdsf C++ API Reference Manual**

0.3.0

Generated by Doxygen 1.5.1

Fri Jul 4 22:34:50 2008

## Contents

<a href="#">1 vdsf C++ API Directory Hierarchy</a>	<a href="#">1</a>
<a href="#">2 vdsf C++ API Class Index</a>	<a href="#">1</a>
<a href="#">3 vdsf C++ API File Index</a>	<a href="#">2</a>
<a href="#">4 vdsf C++ API Directory Documentation</a>	<a href="#">2</a>
<a href="#">5 vdsf C++ API Class Documentation</a>	<a href="#">3</a>
<a href="#">6 vdsf C++ API File Documentation</a>	<a href="#">28</a>

## 1 vdsf C++ API Directory Hierarchy

### 1.1 vdsf C++ API Directories

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

src	<a href="#">3</a>
include	<a href="#">2</a>
vdsf	<a href="#">3</a>

## 2 vdsf C++ API Class Index

### 2.1 vdsf C++ API Class List

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

<a href="#">vdsException</a>	<a href="#">3</a>
<a href="#">vdsFolder</a>	<a href="#">5</a>
<a href="#">vdsHashMap</a>	<a href="#">8</a>
<a href="#">vdsProcess</a>	<a href="#">15</a>
<a href="#">vdsQueue</a>	<a href="#">16</a>

[vdsSession](#) 21

## 3 vdsf C++ API File Index

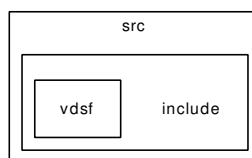
### 3.1 vdsf C++ API File List

Here is a list of all files with brief descriptions:

<a href="#">/home/project/VDSF/vdsf/trunk/src/include/vdsf/vds</a>	28
<a href="#">/home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsException</a>	28
<a href="#">/home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsFolder</a>	29
<a href="#">/home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsHashMap</a>	29
<a href="#">/home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsProcess</a>	30
<a href="#">/home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsQueue</a>	31
<a href="#">/home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsSession</a>	31

## 4 vdsf C++ API Directory Documentation

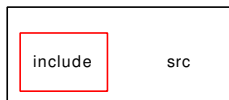
### 4.1 /home/project/VDSF/vdsf/trunk/src/include/ Directory Reference



#### Directories

- directory [vdsf](#)

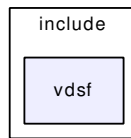
## 4.2 /home/project/VDSF/vdsf/trunk/src/ Directory Reference



### Directories

- directory [include](#)

## 4.3 /home/project/VDSF/vdsf/trunk/src/include/vdsf/ Directory Reference



### Files

- file [vds](#)
- file [vdsException](#)
- file [vdsFolder](#)
- file [vdsHashMap](#)
- file [vdsProcess](#)
- file [vdsQueue](#)
- file [vdsSession](#)

## 5 vdsf C++ API Class Documentation

### 5.1 vdsException Class Reference

#### Public Member Functions

- [vdsException](#) (int theErrorCode, VDS\_HANDLE sessionHandle, const char \*functionName)

*Construct a vdsf exception using the return code from the C function and the name of the function throwing it.*

- virtual `~vdsException ()`
- `std::string & Message (std::string &errorMessage)`

*Return the error message.*

- `std::string & Message ()`
- Return the error message.*

- `int ErrorCode ()`
- Return the error code associated with the exception.*

#### Private Attributes

- `std::string msg`
- `int errcode`

### 5.1.1 Constructor & Destructor Documentation

#### 5.1.1.1 `vdsException::vdsException (int theErrorCode, VDS_HANDLE session-Handle, const char * functionName)`

Construct a vdsf exception using the return code from the C function and the name of the function throwing it.

#### 5.1.1.2 `virtual vdsException::~~vdsException ()` [virtual]

### 5.1.2 Member Function Documentation

#### 5.1.2.1 `std::string& vdsException::Message (std::string & errorMessage)`

Return the error message.

#### 5.1.2.2 `std::string& vdsException::Message ()`

Return the error message.

#### 5.1.2.3 `int vdsException::ErrorCode ()` [inline]

Return the error code associated with the exception.

### 5.1.3 Member Data Documentation

5.1.3.1 `std::string vdsException::msg` [private]

5.1.3.2 `int vdsException::errcode` [private]

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

- `/home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsException`

## 5.2 vdsFolder Class Reference

### Public Member Functions

- `vdsFolder (vdsSession &session)`
- `virtual ~vdsFolder ()`
- `void Close ()`  
*Close a folder.*
- `int GetFirst (vdsFolderEntry *pEntry)`  
*Iterate through the folder - no data items are removed from the folder by this function.*
- `int GetNext (vdsFolderEntry *pEntry)`  
*Iterate through the folder.*
- `void Open (const std::string &folderName)`  
*Open an existing folder (see `vdsSession::CreateObject` to create a new folder).*
- `void Open (const char *folderName, size_t nameLengthInBytes)`  
*Open an existing folder (see `vdsSession::CreateObject` to create a new folder).*
- `void Status (vdsObjStatus *pStatus)`  
*Return the status of the folder.*

### Private Attributes

- `VDS_HANDLE m_objectHandle`  
*Pointer to the `vdsaFolder` struct.*
- `VDS_HANDLE & m_sessionHandle`  
*Reference to the `vdsaSession` struct (we belong to).*

### 5.2.1 Constructor & Destructor Documentation

#### 5.2.1.1 vdsFolder::vdsFolder ([vdsSession](#) & *session*)

#### 5.2.1.2 virtual vdsFolder::~~vdsFolder () [[virtual](#)]

### 5.2.2 Member Function Documentation

#### 5.2.2.1 void vdsFolder::Close ()

Close a folder.

This function terminates the current access to the folder in shared memory (the folder itself is untouched).

#### Exceptions:

[vdsException](#) An abnormal error occurred.

#### 5.2.2.2 int vdsFolder::GetFirst (vdsFolderEntry \* *pEntry*)

Iterate through the folder - no data items are removed from the folder by this function.

Data items which were added by another session and are not yet committed will not be seen by the iterator. Likewise, destroyed data items (even if not yet committed) are invisible.

#### Parameters:

→ *pEntry* The data structure provided by the user to hold the content of each item in the folder. Memory allocation for this buffer is the responsibility of the caller.

#### Returns:

0 on success or VDS\_IS\_EMPTY if the folder is empty.

#### Exceptions:

[vdsException](#) An abnormal error occurred.

#### 5.2.2.3 int vdsFolder::GetNext (vdsFolderEntry \* *pEntry*)

Iterate through the folder.

Data items which were added by another session and are not yet committed will not be seen by the iterator. Likewise, destroyed data items (even if not yet committed) are invisible.

Evidently, you must call [vdsFolder::GetFirst](#) to initialize the iterator.

**Parameters:**

→ *pEntry* The data structure provided by the user to hold the content of each item in the folder. Memory allocation for this buffer is the responsibility of the caller.

**Returns:**

0 on success or VDS\_REACHED\_THE\_END when the iteration reaches the end of the folder.

**Exceptions:**

[vdsException](#) An abnormal error occurred.

**5.2.2.4 void vdsFolder::Open (const std::string & *folderName*)**

Open an existing folder (see [vdsSession::CreateObject](#) to create a new folder).

**Parameters:**

← *folderName* The fully qualified name of the folder.

**Exceptions:**

[vdsException](#) An abnormal error occurred.

**5.2.2.5 void vdsFolder::Open (const char \* *folderName*, size\_t *nameLengthInBytes*)**

Open an existing folder (see [vdsSession::CreateObject](#) to create a new folder).

**Parameters:**

← *folderName* The fully qualified name of the folder.

← *nameLengthInBytes* The length of *folderName* (in bytes) not counting the null terminator.

**Exceptions:**

[vdsException](#) An abnormal error occurred.



**5.2.2.6 void vdsFolder::Status (vdsObjStatus \* *pStatus*)**

Return the status of the folder.

**Parameters:**

→ *pStatus* A pointer to the status structure.

**Exceptions:**

*vdsException* An abnormal error occurred.

**5.2.3 Member Data Documentation****5.2.3.1 VDS\_HANDLE vdsFolder::m\_objectHandle [private]**

Pointer to the vdsFolder struct.

**5.2.3.2 VDS\_HANDLE& vdsFolder::m\_sessionHandle [private]**

Reference to the vdsSession struct (we belong to).

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

- /home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsFolder

**5.3 vdsHashMap Class Reference****Public Member Functions**

- [vdsHashMap](#) ([vdsSession](#) &session)
- virtual [~vdsHashMap](#) ()
- void [Close](#) ()  
*Close a hash map.*
- void [Definition](#) (vdsObjectDefinition \*\*definition)  
*Retrieve the data definition of the hash map.*
- void [Delete](#) (const void \*key, size\_t keyLength)  
*Remove the data item identified by the given key from the hash map.*
- void [Get](#) (const void \*key, size\_t keyLength, void \*buffer, size\_t bufferLength, size\_t \*returnedLength)  
*Retrieve the data item identified by the given key from the hash map.*

- int [GetFirst](#) (void \*key, size\_t keyLength, void \*buffer, size\_t bufferLength, size\_t \*retKeyLength, size\_t \*retDataLength)  
*Iterate through the hash map.*
- int [GetNext](#) (void \*key, size\_t keyLength, void \*buffer, size\_t bufferLength, size\_t \*retKeyLength, size\_t \*retDataLength)  
*Iterate through the hash map.*
- void [Insert](#) (const void \*key, size\_t keyLength, const void \*data, size\_t dataLength)  
*Insert a data element in the hash map.*
- void [Open](#) (const std::string &hashMapName)  
*Open an existing hash map (see [vdsSession::CreateObject](#) to create a new object).*
- void [Open](#) (const char \*hashMapName, size\_t nameLengthInBytes)  
*Open an existing hash map (see [vdsSession::CreateObject](#) to create a new object).*
- void [Replace](#) (const void \*key, size\_t keyLength, const void \*data, size\_t dataLength)  
*Replace a data element in the hash map.*
- void [Status](#) (vdsObjStatus \*pStatus)  
*Return the status of the hash map.*

#### Private Attributes

- VDS\_HANDLE [m\\_objectHandle](#)  
*Pointer to the vdsaHashMap struct.*
- VDS\_HANDLE & [m\\_sessionHandle](#)  
*Reference to the vdsaSession struct (we belong to).*

### 5.3.1 Constructor & Destructor Documentation

#### 5.3.1.1 vdsHashMap::vdsHashMap ([vdsSession](#) & session)

#### 5.3.1.2 virtual vdsHashMap::~vdsHashMap () [virtual]

### 5.3.2 Member Function Documentation

#### 5.3.2.1 void vdsHashMap::Close ()

Close a hash map.

This function terminates the current access to the hash map in shared memory (the hash map itself is untouched).

**Warning:**

Closing an object does not automatically commit or rollback data items that were inserted or removed. You still must use either [vdsSession::Commit](#) or [vdsSession::Rollback](#) to end the current unit of work.

**Exceptions:**

[vdsException](#) An abnormal error occurred.

#### 5.3.2.2 void vdsHashMap::Definition (vdsObjectDefinition \*\* *definition*)

Retrieve the data definition of the hash map.

**Warning:**

This function allocates a buffer to hold the definition (using malloc()). You must free it (with free()) when you no longer need the definition.

**Parameters:**

→ *definition* The buffer allocated by the API to hold the content of the object definition. Freeing the memory (with free()) is the responsibility of the caller.

**Exceptions:**

[vdsException](#) An abnormal error occurred.

#### 5.3.2.3 void vdsHashMap::Delete (const void \* *key*, size\_t *keyLength*)

Remove the data item identified by the given key from the hash map.

Data items which were added by another session and are not yet committed will not be seen by this function and cannot be removed. Likewise, destroyed data items (even if not yet committed) are invisible.

The removals only become permanent after a call to [vdsSession::Commit](#).

**Parameters:**

- ← *key* The key of the item to be removed.
- ← *keyLength* The length of the *key* buffer (in bytes).

**Exceptions:**

- vdsException* An abnormal error occurred.

#### 5.3.2.4 void vdsHashMap::Get (const void \* *key*, size\_t *keyLength*, void \* *buffer*, size\_t *bufferLength*, size\_t \* *returnedLength*)

Retrieve the data item identified by the given key from the hash map.

Data items which were added by another session and are not yet committed will not be seen by this function. Likewise, destroyed data items (even if not yet committed) are invisible.

**Parameters:**

- ← *key* The key of the item to be retrieved.
- ← *keyLength* The length of the *key* buffer (in bytes).
- *buffer* The buffer provided by the user to hold the content of the data item. Memory allocation for this buffer is the responsibility of the caller.
- ← *bufferLength* The length of *buffer* (in bytes).
- *returnedLength* The actual number of bytes in the data item.

**Exceptions:**

- vdsException* An abnormal error occurred.

#### 5.3.2.5 int vdsHashMap::GetFirst (void \* *key*, size\_t *keyLength*, void \* *buffer*, size\_t *bufferLength*, size\_t \* *retKeyLength*, size\_t \* *retDataLength*)

Iterate through the hash map.

Data items which were added by another session and are not yet committed will not be seen by the iterator. Likewise, destroyed data items (even if not yet committed) are invisible.

Data items retrieved this way will not be sorted.

**Parameters:**

- *key* The key buffer provided by the user to hold the content of the key associated with the first element. Memory allocation for this buffer is the responsibility of the caller.

- ← **keyLength** The length of the *key* buffer (in bytes).
- **buffer** The buffer provided by the user to hold the content of the first element.  
Memory allocation for this buffer is the responsibility of the caller.
- ← **bufferLength** The length of *buffer* (in bytes).
- **retKeyLength** The actual number of bytes in the key
- **retDataLength** The actual number of bytes in the data item.

**Returns:**

0 on success or VDS\_IS\_EMPTY if the hash map is empty.

**Exceptions:**

**vdsException** An abnormal error occurred.

#### 5.3.2.6 int vdsHashMap::GetNext (void \* *key*, size\_t *keyLength*, void \* *buffer*, size\_t *bufferLength*, size\_t \* *retKeyLength*, size\_t \* *retDataLength*)

Iterate through the hash map.

Data items which were added by another session and are not yet committed will not be seen by the iterator. Likewise, destroyed data items (even if not yet committed) are invisible.

Evidently, you must call **vdsHashMap::GetFirst** to initialize the iterator. Not so evident - calling **vdsHashMap::Get** will reset the iteration to the data item retrieved by this function (they use the same internal storage). If this cause a problem, please let us know.

Data items retrieved this way will not be sorted.

**Parameters:**

- **key** The key buffer provided by the user to hold the content of the key associated with the data element. Memory allocation for this buffer is the responsibility of the caller.
- ← **keyLength** The length of the *key* buffer (in bytes).
- **buffer** The buffer provided by the user to hold the content of the data element.  
Memory allocation for this buffer is the responsibility of the caller.
- ← **bufferLength** The length of *buffer* (in bytes).
- **retKeyLength** The actual number of bytes in the key
- **retDataLength** The actual number of bytes in the data item.

**Returns:**

0 on success or VDS\_REACHED\_THE\_END when the iteration reaches the end of the hash map.

**Exceptions:**

[\*vdsException\*](#) An abnormal error occurred.

### 5.3.2.7 void vdsHashMap::Insert (const void \* *key*, size\_t *keyLength*, const void \* *data*, size\_t *dataLength*)

Insert a data element in the hash map.

The additions only become permanent after a call to [\*vdsSession::Commit\*](#).

**Parameters:**

- ← *key* The key of the item to be inserted.
- ← *keyLength* The length of the *key* buffer (in bytes).
- ← *data* The data item to be inserted.
- ← *dataLength* The length of *data* (in bytes).

**Exceptions:**

[\*vdsException\*](#) An abnormal error occurred.

### 5.3.2.8 void vdsHashMap::Open (const std::string & *hashMapName*)

Open an existing hash map (see [\*vdsSession::CreateObject\*](#) to create a new object).

**Parameters:**

- ← *hashMapName* The fully qualified name of the hash map.

**Exceptions:**

[\*vdsException\*](#) An abnormal error occurred.

### 5.3.2.9 void vdsHashMap::Open (const char \* *hashMapName*, size\_t *nameLengthInBytes*)

Open an existing hash map (see [\*vdsSession::CreateObject\*](#) to create a new object).

**Parameters:**

- ← *hashMapName* The fully qualified name of the hash map.
- ← *nameLengthInBytes* The length of *hashMapName* (in bytes) not counting the null terminator.

**Exceptions:**

[\*vdsException\*](#) An abnormal error occurred.

**5.3.2.10 void vdsHashMap::Replace (const void \* *key*, size\_t *keyLength*, const void \* *data*, size\_t *dataLength*)**

Replace a data element in the hash map.

The replacements only become permanent after a call to [vdsSession::Commit](#).

**Parameters:**

- ← *key* The key of the item to be replaced.
- ← *keyLength* The length of the *key* buffer (in bytes).
- ← *data* The new data item that will replace the previous data.
- ← *dataLength* The length of *data* (in bytes).

**Exceptions:**

[vdsException](#) An abnormal error occurred.

**5.3.2.11 void vdsHashMap::Status (vdsObjStatus \* *pStatus*)**

Return the status of the hash map.

**Parameters:**

- *pStatus* A pointer to the status structure.

**Exceptions:**

[vdsException](#) An abnormal error occurred.

**5.3.3 Member Data Documentation****5.3.3.1 VDS\_HANDLE [vdsHashMap::m\\_objectHandle](#) [private]**

Pointer to the vdsHashMap struct.

**5.3.3.2 VDS\_HANDLE& [vdsHashMap::m\\_sessionHandle](#) [private]**

Reference to the vdsSession struct (we belong to).

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

- /home/project/VDSF/vdsf/trunk/src/include/vdsf/[vdsHashMap](#)

## 5.4 vdsProcess Class Reference

### Public Member Functions

- [vdsProcess](#) ()
- void [Init](#) (const char \*wdAddress, bool protectionNeeded=false)  
*This function initializes access to a VDS.*
- virtual [~vdsProcess](#) ()  
*The destructor terminates all access to the VDS.*

### 5.4.1 Constructor & Destructor Documentation

#### 5.4.1.1 vdsProcess::vdsProcess ()

#### 5.4.1.2 virtual vdsProcess::~~vdsProcess () [virtual]

The destructor terminates all access to the VDS.

This function will also close all sessions and terminate all accesses to the different objects.

### 5.4.2 Member Function Documentation

#### 5.4.2.1 void vdsProcess::Init (const char \* wdAddress, bool protectionNeeded = false)

This function initializes access to a VDS.

It takes 2 input arguments, the address of the watchdog and a boolean value. This last one indicates if sessions and other objects (Queues, etc) are shared amongst threads (in the current process) and must be protected. Recommendation: always set protectionNeeded to false unless you cannot do otherwise. In other words it is recommended to use one session object for each thread. Also if the same queue needs to be accessed by two threads it is more efficient to have two different objects instead of sharing a single one.

[Additional note: API objects (or C handles) are just proxies for the real objects sitting in shared memory. Proper synchronization is already done in shared memory and it is best to avoid to synchronize these proxy objects.]

#### Parameters:

- ← **wdAddress** The address of the watchdog. Currently a string with the port number ("12345").



← *protectionNeeded* A boolean value indicating if multi-threaded locks are needed or not.

#### Exceptions:

*vdsException* An abnormal error occurred.

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

- /home/project/VDSF/vdsf/trunk/src/include/vdsf/*vdsProcess*

## 5.5 vdsQueue Class Reference

### Public Member Functions

- *vdsQueue* (*vdsSession* &session)
- virtual *~vdsQueue* ()
- void *Close* ()  
*Close a FIFO queue.*
- void *Definition* (*vdsObjectDefinition* \*\*definition)  
*Retrieve the data definition of the queue.*
- int *GetFirst* (void \*buffer, size\_t bufferLength, size\_t \*returnedLength)  
*Iterate through the queue - no data items are removed from the queue by this function.*
- int *GetNext* (void \*buffer, size\_t bufferLength, size\_t \*returnedLength)  
*Iterate through the queue - no data items are removed from the queue by this function.*
- void *Open* (const std::string &queueName)  
*Open an existing FIFO queue (see Session::CreateObject to create a new queue).*
- void *Open* (const char \*queueName, size\_t nameLengthInBytes)  
*Open an existing FIFO queue (see Session::CreateObject to create a new queue).*
- int *Pop* (void \*buffer, size\_t bufferLength, size\_t \*returnedLength)  
*Remove the first item from the beginning of a FIFO queue and return it to the caller.*
- void *Push* (const void \*pItem, size\_t length)  
*Insert a data element at the end of the FIFO queue.*
- void *Status* (*vdsObjStatus* \*pStatus)  
*Return the status of this queue.*

### Private Attributes

- VDS\_HANDLE [m\\_objectHandle](#)  
*Pointer to the vdsaQueue struct.*
- VDS\_HANDLE & [m\\_sessionHandle](#)  
*Reference to the vdsaSession struct (we belong to).*

### 5.5.1 Constructor & Destructor Documentation

#### 5.5.1.1 vdsQueue::vdsQueue ([vdsSession](#) & *session*)

#### 5.5.1.2 virtual vdsQueue::~~vdsQueue () [virtual]

### 5.5.2 Member Function Documentation

#### 5.5.2.1 void vdsQueue::Close ()

Close a FIFO queue.

This function terminates the current access to the queue in shared memory (the queue itself, in shared memory is untouched).

#### Warning:

Closing an object does not automatically commit or rollback data items that were inserted or removed. You still must use either [vdsSession::Commit](#) or [vdsSession::Rollback](#) to end the current unit of work.

#### Exceptions:

[vdsException](#) An abnormal error occurred.

#### 5.5.2.2 void vdsQueue::Definition (vdsObjectDefinition \*\* *definition*)

Retrieve the data definition of the queue.

#### Warning:

This function allocates a buffer to hold the definition (using malloc()). You must free it (with free()) when you no longer need the definition.

**Parameters:**

- *definition* The buffer allocated by the API to hold the content of the object definition. Freeing the memory (with free()) is the responsibility of the caller.

**Exceptions:**

- vdsException* An abnormal error occurred.

**5.5.2.3 int vdsQueue::GetFirst (void \* *buffer*, size\_t *bufferLength*, size\_t \* *returnedLength*)**

Iterate through the queue - no data items are removed from the queue by this function.

Data items which were added by another session and are not yet committed will not be seen by the iterator. Likewise, destroyed data items (even if not yet committed) are invisible.

**Parameters:**

- *buffer* The buffer provided by the user to hold the content of the first element. Memory allocation for this buffer is the responsibility of the caller.
- ← *bufferLength* The length of *buffer* (in bytes).
- *returnedLength* The actual number of bytes in the data item.

**Returns:**

- 0 on success or VDS\_IS\_EMPTY if the queue is empty.

**Exceptions:**

- vdsException* An abnormal error occurred.

**5.5.2.4 int vdsQueue::GetNext (void \* *buffer*, size\_t *bufferLength*, size\_t \* *returnedLength*)**

Iterate through the queue - no data items are removed from the queue by this function.

Data items which were added by another session and are not yet committed will not be seen by the iterator. Likewise, destroyed data items (even if not yet committed) are invisible.

Evidently, you must call GetFirst to initialize the iterator. Not so evident - calling Pop will reset the iteration to the last element (they use the same internal storage). If this cause a problem, please let us know.

**Parameters:**

- *buffer* The buffer provided by the user to hold the content of the next element. Memory allocation for this buffer is the responsibility of the caller.
- ← *bufferLength* The length of *buffer* (in bytes).
- *returnedLength* The actual number of bytes in the data item.

**Returns:**

0 on success or VDS\_REACHED\_THE\_END when the iteration reaches the end of the queue.

**Exceptions:**

*vdsException* An abnormal error occurred.

**5.5.2.5 void vdsQueue::Open (const std::string & queueName)**

Open an existing FIFO queue (see Session::CreateObject to create a new queue).

**Parameters:**

- ← *queueName* The fully qualified name of the queue.

**Exceptions:**

*vdsException* An abnormal error occurred.

**5.5.2.6 void vdsQueue::Open (const char \* queueName, size\_t nameLengthInBytes)**

Open an existing FIFO queue (see Session::CreateObject to create a new queue).

**Parameters:**

- ← *queueName* The fully qualified name of the queue.
- ← *nameLengthInBytes* The length of *queueName* (in bytes) not counting the null terminator.

**Exceptions:**

*vdsException* An abnormal error occurred.

### 5.5.2.7 int vdsQueue::Pop (void \* *buffer*, size\_t *bufferLength*, size\_t \* *returnedLength*)

Remove the first item from the beginning of a FIFO queue and return it to the caller.

Data items which were added by another session and are not yet committed will not be seen by this function. Likewise, destroyed data items (even if not yet committed) are invisible.

The removals only become permanent after a call to [vdsSession::Commit](#).

#### Parameters:

- *buffer* The buffer provided by the user to hold the content of the data item. Memory allocation for this buffer is the responsibility of the caller.
- ← *bufferLength* The length of *buffer* (in bytes).
- *returnedLength* The actual number of bytes in the data item.

#### Returns:

0 on success or VDS\_IS\_EMPTY if the queue is empty or VDS\_ITEM\_IS\_IN\_USE if all existing items are "invisible".

#### Exceptions:

[vdsException](#) An abnormal error occurred.

### 5.5.2.8 void vdsQueue::Push (const void \* *pItem*, size\_t *length*)

Insert a data element at the end of the FIFO queue.

The additions only become permanent after a call to [vdsSession::Commit](#).

#### Parameters:

- ← *pItem* The data item to be inserted.
- ← *length* The length of *pItem* (in bytes).

#### Exceptions:

[vdsException](#) An abnormal error occurred.

### 5.5.2.9 void vdsQueue::Status (vdsObjStatus \* *pStatus*)

Return the status of the queue.

**Parameters:**

→ *pStatus* A pointer to the status structure.

**Exceptions:**

*vdsException* An abnormal error occurred.

**5.5.3 Member Data Documentation****5.5.3.1 VDS\_HANDLE *vdsQueue::m\_objectHandle* [private]**

Pointer to the vdsQueue struct.

**5.5.3.2 VDS\_HANDLE& *vdsQueue::m\_sessionHandle* [private]**

Reference to the vdsSession struct (we belong to).

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

- /home/project/VDSF/vdsf/trunk/src/include/vdsf/*vdsQueue*

**5.6 vdsSession Class Reference****Public Member Functions**

- *vdsSession* ()
- virtual *~vdsSession* ()  
*Terminate the current session and destroy this object.*
- void *Commit* ()  
*Commit all insertions and deletions (of the current session) executed since the previous call to Commit or Rollback.*
- void *CreateObject* (const std::string &objectName, vdsObjectDefinition \*pDefinition)  
*Create a new object in shared memory.*
- void *CreateObject* (const char \*objectName, size\_t nameLengthInBytes, vdsObjectDefinition \*pDefinition)  
*Create a new object in shared memory.*
- void *DestroyObject* (const std::string &objectName)  
*Destroy an existing object in shared memory.*

- void [DestroyObject](#) (const char \*objectName, size\_t nameLengthInBytes)  
*Destroy an existing object in shared memory.*
- void [ErrorMsg](#) (char \*message, size\_t msgLengthInBytes)  
*Return the error message associated with the last error(s).*
- std::string & [ErrorMsg](#) (std::string &message)  
*Return the error message associated with the last error(s).*
- void [GetInfo](#) (vdsInfo \*pInfo)  
*Return information on the current status of the VDS (Virtual Data Space).*
- void [GetStatus](#) (const std::string &objectName, vdsObjStatus \*pStatus)  
*Return the status of the named object.*
- void [GetStatus](#) (const char \*objectName, size\_t nameLengthInBytes, vdsObjStatus \*pStatus)  
*Return the status of the named object.*
- void [Init](#) ()  
*This function initializes a session.*
- int [LastError](#) ()  
*Return the last error seen in previous calls (of the current session).*
- void [Rollback](#) ()  
*Rollback all insertions and deletions (of the current session) executed since the previous call to Commit or Rollback.*

### Private Attributes

- VDS\_HANDLE [m\\_sessionHandle](#)  
*Pointer to the vdsSession struct.*

### Friends

- class [vdsFolder](#)
- class [vdsHashMap](#)
- class [vdsLifo](#)
- class [vdsQueue](#)

## 5.6.1 Constructor & Destructor Documentation

### 5.6.1.1 vdsSession::vdsSession ()

### 5.6.1.2 virtual vdsSession::~~vdsSession () [virtual]

Terminate the current session and destroy this object.

An implicit call to Rollback is executed by this destructor.

## 5.6.2 Member Function Documentation

### 5.6.2.1 void vdsSession::Commit ()

Commit all insertions and deletions (of the current session) executed since the previous call to Commit or Rollback.

Insertions and deletions subjected to this call include both data items inserted and deleted from data containers (maps, etc.) and objects themselves created with CreateObject and/or destroyed with DestroyObject.

Note: the internal calls executed by the engine to satisfy this request cannot fail. As such, you cannot find yourself with an ugly situation where some operations were committed and others not. If this function throws an exception, nothing was committed.

#### Exceptions:

*vdsException* An abnormal error occurred.

### 5.6.2.2 void vdsSession::CreateObject (const std::string & *objectName*, vds-ObjectDefinition \* *pDefinition*)

Create a new object in shared memory.

The creation of the object only becomes permanent after a call to Commit.

#### Parameters:

← *objectName* The fully qualified name of the object.

← *pDefinition* The type of object to create (folder, queue, etc.) and the optional definitions (as needed).

#### Exceptions:

*vdsException* An abnormal error occurred.



### 5.6.2.3 void vdsSession::CreateObject (const char \* *objectName*, size\_t *nameLengthInBytes*, vdsObjectDefinition \* *pDefinition*)

Create a new object in shared memory.

The creation of the object only becomes permanent after a call to Commit.

#### Parameters:

- ← *objectName* The fully qualified name of the object.
- ← *nameLengthInBytes* The length of *objectName* (in bytes) not counting the null terminator.
- ← *pDefinition* The type of object to create (folder, queue, etc.) and the optional definitions (as needed).

#### Exceptions:

*vdsException* An abnormal error occurred.

### 5.6.2.4 void vdsSession::DestroyObject (const std::string & *objectName*)

Destroy an existing object in shared memory.

The destruction of the object only becomes permanent after a call to Commit.

#### Parameters:

- ← *objectName* The fully qualified name of the object.

#### Exceptions:

*vdsException* An abnormal error occurred.

### 5.6.2.5 void vdsSession::DestroyObject (const char \* *objectName*, size\_t *nameLengthInBytes*)

Destroy an existing object in shared memory.

The destruction of the object only becomes permanent after a call to Commit.

#### Parameters:

- ← *objectName* The fully qualified name of the object.
- ← *nameLengthInBytes* The length of *objectName* (in bytes) not counting the null terminator.

#### Exceptions:

*vdsException* An abnormal error occurred.

### 5.6.2.6 void vdsSession::ErrorMsg (char \* *message*, size\_t *msgLengthInBytes*)

Return the error message associated with the last error(s).

If the length of the error message is greater than the length of the provided buffer, the error message will be truncated to fit in the provided buffer.

Caveat, some basic errors cannot be captured, if the provided handles (session handles or object handles) are incorrect (NULL, for example). Without a proper handle, the code cannot know where to store the error...

#### Parameters:

- *message* Buffer for the error message. Memory allocation for this buffer is the responsibility of the caller.
- ← *msgLengthInBytes* The length of *message* (in bytes). Must be at least 32 bytes.

#### Exceptions:

*vdsException* An abnormal error occurred.

### 5.6.2.7 std::string& vdsSession::ErrorMsg (std::string & *message*)

Return the error message associated with the last error(s).

Caveat, some basic errors cannot be captured, if the provided handles (session handles or object handles) are incorrect (NULL, for example). Without a proper handle, the code cannot know where to store the error...

#### Parameters:

- *message* Buffer for the error message. Memory allocation for this buffer is the responsibility of the caller.

#### Exceptions:

*vdsException* An abnormal error occurred.

### 5.6.2.8 void vdsSession::GetInfo (vdsInfo \* *pInfo*)

Return information on the current status of the VDS (Virtual Data Space).

The fetched information is mainly about the current status of the memory allocator.

#### Parameters:

- *pInfo* A pointer to the vdsInfo structure.

#### Exceptions:

*vdsException* An abnormal error occurred.

### 5.6.2.9 void vdsSession::GetStatus (const std::string & *objectName*, vdsObjStatus \* *pStatus*)

Return the status of the named object.

#### Parameters:

- ← *objectName* The fully qualified name of the object.
- *pStatus* A pointer to the vdsObjStatus structure.

#### Exceptions:

*vdsException* An abnormal error occurred.

### 5.6.2.10 void vdsSession::GetStatus (const char \* *objectName*, size\_t *nameLengthInBytes*, vdsObjStatus \* *pStatus*)

Return the status of the named object.

#### Parameters:

- ← *objectName* The fully qualified name of the object.
- ← *nameLengthInBytes* The length of *objectName* (in bytes) not counting the null terminator.
- *pStatus* A pointer to the vdsObjStatus structure.

#### Exceptions:

*vdsException* An abnormal error occurred.

### 5.6.2.11 void vdsSession::Init ()

This function initializes a session.

This function will also initiate a new transaction.

Upon normal termination, the current transaction is rolled back. You MUST explicitly call Commit to save your changes.

#### Exceptions:

*vdsException* An abnormal error occurred.

### 5.6.2.12 int vdsSession::LastError ()

Return the last error seen in previous calls (of the current session).

#### Exceptions:

[\*vdsException\*](#) An abnormal error occurred.

### 5.6.2.13 void vdsSession::Rollback ()

Rollback all insertions and deletions (of the current session) executed since the previous call to Commit or Rollback.

Insertions and deletions subjected to this call include both data items inserted and deleted from data containers (maps, etc.) and objects themselves created with CreateObject and/or destroyed with DestroyObject.

Note: the internal calls executed by the engine to satisfy this request cannot fail. As such, you cannot find yourself with an ugly situation where some operations were rolledback and others not. If this function throws an exception, nothing was rolledback.

#### Exceptions:

[\*vdsException\*](#) An abnormal error occurred.

## 5.6.3 Friends And Related Function Documentation

### 5.6.3.1 friend class [\*vdsFolder\*](#) [friend]

### 5.6.3.2 friend class [\*vdsHashMap\*](#) [friend]

### 5.6.3.3 friend class [\*vdsLifo\*](#) [friend]

### 5.6.3.4 friend class [\*vdsQueue\*](#) [friend]

## 5.6.4 Member Data Documentation

### 5.6.4.1 VDS\_HANDLE [\*vdsSession::m\\_sessionHandle\*](#) [private]

Pointer to the vdsSession struct.

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

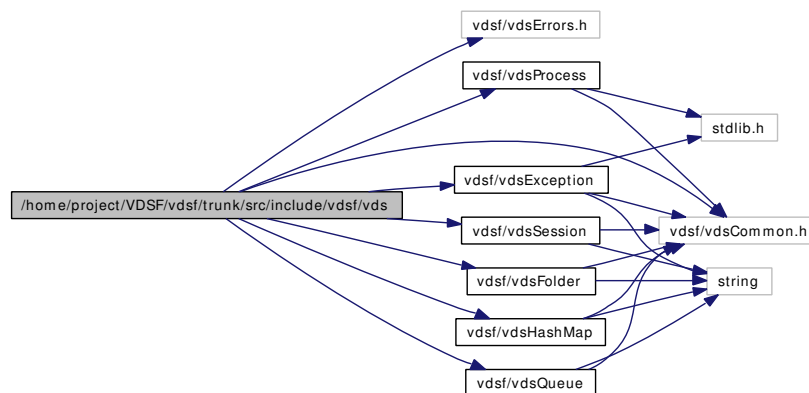
- /home/project/VDSF/vdsf/trunk/src/include/vdsf/[\*vdsSession\*](#)

## 6 vdsf C++ API File Documentation

### 6.1 /home/project/VDSF/vdsf/trunk/src/include/vdsf/vds File Reference

```
#include <vdsf/vdsErrors.h>
#include <vdsf/vdsCommon.h>
#include <vdsf/vdsProcess>
#include <vdsf/vdsSession>
#include <vdsf/vdsFolder>
#include <vdsf/vdsHashMap>
#include <vdsf/vdsQueue>
#include <vdsf/vdsException>
```

Include dependency graph for vds:

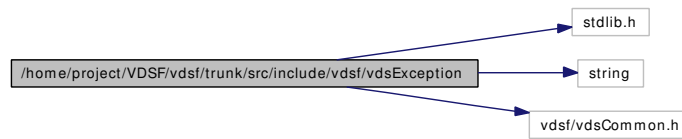


### 6.2 /home/project/VDSF/vdsf/trunk/src/include/vdsf/vds-Exception File Reference

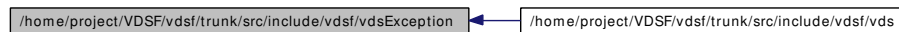
```
#include <stdlib.h>
#include <string>
#include <vdsf/vdsCommon.h>
```

Include dependency graph for vdsException:

### 6.3 /home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsFolder File Reference29



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



#### Classes

- class [vdsException](#)

### 6.3 /home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsFolder File Reference

```
#include <vdsf/vdsCommon.h>
```

```
#include <string>
```

Include dependency graph for vdsFolder:



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



#### Classes

- class [vdsFolder](#)

### 6.4 /home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsHash-Map File Reference

```
#include <vdsf/vdsCommon.h>
```

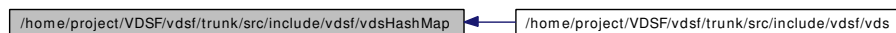
## 6.5 /home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsProcess File Reference

```
#include <string>
```

Include dependency graph for vdsHashMap:



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



### Classes

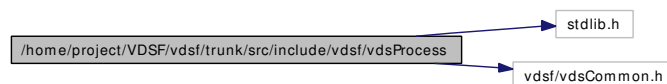
- class [vdsHashMap](#)

## 6.5 /home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsProcess File Reference

```
#include <stdlib.h>
```

```
#include <vdsf/vdsCommon.h>
```

Include dependency graph for vdsProcess:



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



### Classes

- class [vdsProcess](#)

## 6.6 /home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsQueue File Reference

```
#include <vdsf/vdsCommon.h>
```

```
#include <string>
```

Include dependency graph for vdsQueue:



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



### Classes

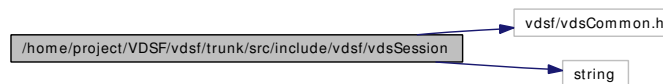
- class [vdsQueue](#)

## 6.7 /home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsSession File Reference

```
#include <vdsf/vdsCommon.h>
```

```
#include <string>
```

Include dependency graph for vdsSession:



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



### Classes

- class [vdsSession](#)



## Index

/home/project/VDSF/vdsf/trunk/src/ vdsHashMap, 10  
Directory Reference, 2 vdsQueue, 17  
/home/project/VDSF/vdsf/trunk/src/include/Delete  
Directory Reference, 2 vdsHashMap, 10  
/home/project/VDSF/vdsf/trunk/src/include/DeleteObject  
Directory Reference, 3 vdsSession, 24  
/home/project/VDSF/vdsf/trunk/src/include/vdsf/vds,  
27 errcode  
/home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsException, 4  
28 ErrorCode  
/home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsException, 4  
29 ErrorMessage  
/home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsHashMap, 24, 25  
29  
/home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsProcess,  
30 vdsHashMap, 10  
/home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsQueue,  
30 vdsFolder, 6  
/home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsSession, 11  
31 vdsQueue, 17  
~vdsException  
vdsException, 4  
~vdsFolder  
vdsFolder, 5  
~vdsHashMap  
vdsHashMap, 9  
~vdsProcess  
vdsProcess, 15  
~vdsQueue  
vdsQueue, 17  
~vdsSession  
vdsSession, 22  
Close  
vdsFolder, 6  
vdsHashMap, 9  
vdsQueue, 17  
Commit  
vdsSession, 23  
CreateObject  
vdsSession, 23  
Definition  
vdsHashMap, 10  
vdsQueue, 17  
vdsException, 4  
ErrorCode  
ErrorMessage  
vdsSession, 24, 25  
Get  
vdsProcess, 10  
vdsHashMap, 10  
GetFirst  
vdsQueue, 6  
vdsFolder, 6  
vdsHashMap, 11  
vdsSession, 11  
vdsQueue, 17  
GetInfo  
vdsSession, 25  
GetNext  
vdsFolder, 6  
vdsHashMap, 11  
vdsQueue, 18  
GetStatus  
vdsSession, 25, 26  
Init  
vdsProcess, 15  
vdsSession, 26  
Insert  
vdsHashMap, 12  
LastError  
vdsSession, 26  
m\_objectHandle  
vdsFolder, 8  
vdsHashMap, 14  
vdsQueue, 20  
m\_sessionHandle  
vdsFolder, 8

- vdsHashMap, 14
  - vdsQueue, 20
  - vdsSession, 27
- Message
  - vdsException, 4
- msg
  - vdsException, 4
- Open
  - vdsFolder, 7
  - vdsHashMap, 13
  - vdsQueue, 19
- Pop
  - vdsQueue, 19
- Push
  - vdsQueue, 20
- Replace
  - vdsHashMap, 13
- Rollback
  - vdsSession, 26
- Status
  - vdsFolder, 7
  - vdsHashMap, 14
  - vdsQueue, 20
- vdsException, 3
  - vdsException, 4
- vdsException
  - ~vdsException, 4
  - errcode, 4
  - ErrorCode, 4
  - Message, 4
  - msg, 4
  - vdsException, 4
- vdsFolder, 5
  - vdsFolder, 5
  - vdsSession, 27
- vdsFolder
  - ~vdsFolder, 5
  - Close, 6
  - GetFirst, 6
  - GetNext, 6
  - m\_objectHandle, 8
  - m\_sessionHandle, 8
- Open, 7
- Status, 7
- vdsFolder, 5
- vdsHashMap, 8
  - vdsHashMap, 9
  - vdsSession, 27
- vdsHashMap
  - ~vdsHashMap, 9
  - Close, 9
  - Definition, 10
  - Delete, 10
  - Get, 10
  - GetFirst, 11
  - GetNext, 11
  - Insert, 12
  - m\_objectHandle, 14
  - m\_sessionHandle, 14
  - Open, 13
  - Replace, 13
  - Status, 14
  - vdsHashMap, 9
- vdsLifo
  - vdsSession, 27
- vdsProcess, 14
  - vdsProcess, 15
- vdsProcess
  - ~vdsProcess, 15
  - Init, 15
  - vdsProcess, 15
- vdsQueue, 16
  - vdsQueue, 17
  - vdsSession, 27
- vdsQueue
  - ~vdsQueue, 17
  - Close, 17
  - Definition, 17
  - GetFirst, 17
  - GetNext, 18
  - m\_objectHandle, 20
  - m\_sessionHandle, 20
  - Open, 19
  - Pop, 19
  - Push, 20
  - Status, 20
  - vdsQueue, 17
- vdsSession, 21

---

- [vdsSession](#), [22](#)
- [vdsSession](#)
  - [~vdsSession](#), [22](#)
  - [Commit](#), [23](#)
  - [CreateObject](#), [23](#)
  - [DestroyObject](#), [24](#)
  - [ErrorMsg](#), [24](#), [25](#)
  - [GetInfo](#), [25](#)
  - [GetStatus](#), [25](#), [26](#)
  - [Init](#), [26](#)
  - [LastError](#), [26](#)
  - [m\\_sessionHandle](#), [27](#)
  - [Rollback](#), [26](#)
  - [vdsFolder](#), [27](#)
  - [vdsHashMap](#), [27](#)
  - [vdsLifo](#), [27](#)
  - [vdsQueue](#), [27](#)
  - [vdsSession](#), [22](#)