vdsf Reference Manual

0.1

Generated by Doxygen 1.5.1

Wed Nov 7 16:30:02 2007

Contents

Chapter 1

vdsf Directory Hierarchy

1.1 vdsf Directories

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

Chapter 2

vdsf Data Structure Index

2.1 vdsf Data Structures

Here are the data structures with brief descriptions:

Chapter 3

vdsf File Index

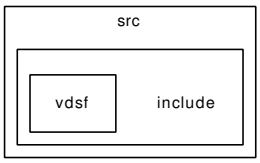
3.1 vdsf File List

Here is a list of all files with brief descriptions:

Chapter 4

vdsf Directory Documentation

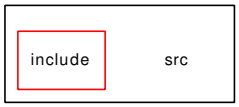
4.1 /home/project/VDSF/vdsf/trunk/src/include/ Directory Reference



Directories

- directory vdsf

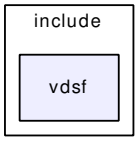
4.2 /home/project/VDSF/vdsf/trunk/src/ Directory Reference



Directories

- directory include

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



Files

- file vds.h
- file vdsCommon.h
- file vdsErrors.h
- file vdsFolder.h
- file vdsHashMap.h
- file vdsProcess.h
- file vdsQueue.h

  This files provides the API to access a VDSF FIFO queue.

- file vdsSession.h

Chapter 5

vdsf Data Structure Documentation

5.1 vdsFolderEntry Struct Reference

`#include <vdsCommon.h>`

5.1.1 Detailed Description

Definition at line 96 of file vdsCommon.h.

Data Fields

- vdsObjectType type
- size_t nameLengthInBytes
- char name [VDS_MAX_NAME_LENGTH *4]

5.1.2 Field Documentation

5.1.2.1 vdsObjectType vdsFolderEntry::type

Definition at line 98 of file vdsCommon.h.

5.1.2.2 size_t vdsFolderEntry::nameLengthInBytes

Definition at line 100 of file vdsCommon.h.

5.1.2.3 char vdsFolderEntry::name[VDS_MAX_NAME_LENGTH *4]

Definition at line 102 of file vdsCommon.h.

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

- /home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsCommon.h

5.2 vdsInfo Struct Reference

`#include <vdsCommon.h>`

5.2.1 Detailed Description

Definition at line 124 of file vdsCommon.h.

Data Fields

- size_t totalSizeInBytes
- size_t allocatedSizeInBytes
- size_t numObjects
- size_t numGroups
- size_t numMallocs
- size_t numFrees
- size_t largestFreeInBytes

5.2.2 Field Documentation

5.2.2.1 size_t vdsInfo::totalSizeInBytes

Definition at line 126 of file vdsCommon.h.

5.2.2.2 size_t vdsInfo::allocatedSizeInBytes

Definition at line 128 of file vdsCommon.h.

5.2.2.3 size_t vdsInfo::numObjects

Definition at line 130 of file vdsCommon.h.

5.2.2.4 size_t vdsInfo::numGroups

Definition at line 132 of file vdsCommon.h.

5.2.2.5 size_t vdsInfo::numMallocs

Definition at line 134 of file vdsCommon.h.

5.2.2.6 size_t vdsInfo::numFrees

Definition at line 136 of file vdsCommon.h.

5.2.2.7 size_t vdsInfo::largestFreeInBytes

Definition at line 138 of file vdsCommon.h.

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

- /home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsCommon.h

5.3 vdsObjStatus Struct Reference

`#include <vdsCommon.h>`

5.3.1 Detailed Description

Definition at line 108 of file vdsCommon.h.

Data Fields

- vdsObjectType type
- size_t numBlocks
- size_t numBlockGroup
- size_t numDataItem
- size_t freeBytes

5.3.2 Field Documentation

5.3.2.1 vdsObjectType vdsObjStatus::type

Definition at line 110 of file vdsCommon.h.

5.3.2.2 size_t vdsObjStatus::numBlocks

Definition at line 112 of file vdsCommon.h.

5.3.2.3 size_t vdsObjStatus::numBlockGroup

Definition at line 114 of file vdsCommon.h.

5.3.2.4 size_t vdsObjStatus::numDataItem

Definition at line 116 of file vdsCommon.h.

5.3.2.5 size_t vdsObjStatus::freeBytes

Definition at line 118 of file vdsCommon.h.

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

- /home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsCommon.h

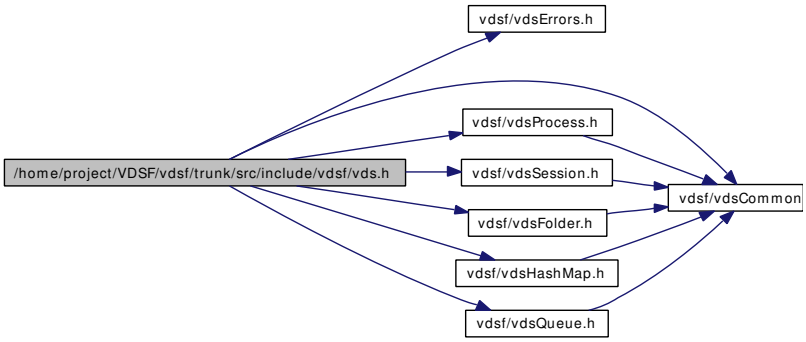
Chapter 6

vdsf File Documentation

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

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

Include dependency graph for vds.h:

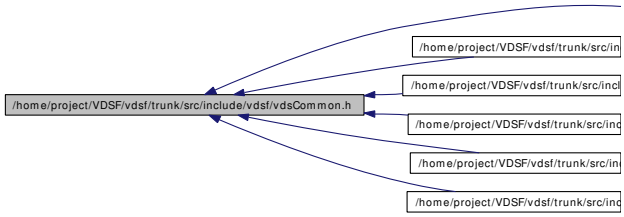

vdsf/vdsErrors.h
vdsf/vdsProcess.h
/home/project/VDSF/vdsf/trunk/src/include/vdsf/vds.h
vdsf/vdsSession.h
vdsf/vdsCommon
vdsf/vdsFolder.h
vdsf/vdsHashMap.h
vdsf/vdsQueue.h

6.2 /home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsCommon.h File Reference

`#include <stdlib.h>`

Include dependency graph for vdsCommon.h:

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



Data Structures

- struct vdsFolderEntry
- struct vdsObjStatus
- struct vdsInfo

Defines

- #define VDSF_EXPORT
- #define VDS_MAX_NAME_LENGTH 256

  Maximum number of characters (or bytes if not supporting i18n) of the name of a vds object (not counting the name of the parent folder(s)).

- #define VDS_MAX_FULL_NAME_LENGTH 1024

  Maximum number of characters (or bytes if not supporting i18n) of the fully qualified name of a vds object (including the name(s) of its parent folder(s)).

Typedefs

- typedef void * VDS_HANDLE

Enumerations

- enum vdsObjectType { VDS_FOLDER = 1, VDS_QUEUE = 2, VDS_HASH_MAP = 3, VDS_LAST_OBJECT_TYPE }
- enum vdsIteratorType { VDS_FIRST = 1, VDS_NEXT = 2 }

6.2.1 Define Documentation

6.2.1.1 #define VDS_MAX_FULL_NAME_LENGTH 1024

Maximum number of characters (or bytes if not supporting i18n) of the fully qualified name of a vds object (including the name(s) of its parent folder(s)).

If the software was compiled with i18n, this maximum is the number of wide characters (4 bytes). Otherwise it is the number of bytes (which should equal the number of characters unless something funny is going on like using UTF-8 as locale and using —disable-i18n with configure...).

Note: setting this value eliminates a possible loophole since some heap memory must be allocated to hold the wide characters string for the duration of the operation (open, close, create or destroy).

Definition at line 75 of file vdsCommon.h.

6.2.1.2 #define VDS_MAX_NAME_LENGTH 256

Maximum number of characters (or bytes if not supporting i18n) of the name of a vds object (not counting the name of the parent folder(s)).

If the software was compiled with i18n, this maximum is the number of wide characters (4 bytes). Otherwise it is the number of bytes (which should equal the number of characters unless something funny is going on like using UTF-8 as locale and using —disable-i18n with configure...).

Definition at line 58 of file vdsCommon.h.

6.2.1.3 #define VDSF_EXPORT

Definition at line 32 of file vdsCommon.h.

6.2.2 Typedef Documentation

6.2.2.1 typedef void* VDS_HANDLE

Definition at line 43 of file vdsCommon.h.

6.2.3 Enumeration Type Documentation

6.2.3.1 enum vdsIteratorType

Enumerator:

VDS_FIRST
VDS_NEXT

Definition at line 87 of file vdsCommon.h.

6.2.3.2 enum vdsObjectType

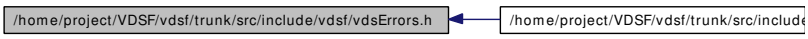
Enumerator:

VDS_FOLDER
VDS_QUEUE
VDS_HASH_MAP
VDS_LAST_OBJECT_TYPE

Definition at line 79 of file vdsCommon.h.

6.3 /home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsErrors.h File Reference

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



Enumerations

- enum vdsErrors {

  VDS_OK = 0, VDS_INTERNAL_ERROR = 666, VDS_ENGINE_BUSY = 1, VDS_NOT_ENOUGH_VDS_MEMORY = 2,

  VDS_NOT_ENOUGH_HEAP_MEMORY = 3, VDS_NOT_ENOUGH_RESOURCES = 4, VDS_WRONG_TYPE_HANDLE = 5, VDS_NULL_HANDLE = 6,

  VDS_NULL_POINTER = 7, VDS_INVALID_LENGTH = 8, VDS_PROCESS_ALREADY_INITIALIZED = 21, VDS_PROCESS_NOT_INITIALIZED = 22,

  VDS_INVALID_WATCHDOG_ADDRESS = 23, VDS_INCOMPATIBLE_VERSIONS = 24, VDS_SOCKET_ERROR = 25, VDS_CONNECT_ERROR = 26,

  VDS_SEND_ERROR = 27, VDS_RECEIVE_ERROR = 28, VDS_BACKSTORE_FILE_MISSING = 29, VDS_ERROR_OPENING_VDS = 30,

  VDS_LOGFILE_ERROR = 41, VDS_SESSION_CANNOT_GET_LOCK = 42, VDS_SESSION_IS_TERMINATED = 43, VDS_INVALID_OBJECT_NAME = 51,

  VDS_NO_SUCH_OBJECT = 52, VDS_NO_SUCH_FOLDER = 53, VDS_OBJECT_ALREADY_PRESENT = 54, VDS_IS_EMPTY = 55,

  VDS_WRONG_OBJECT_TYPE = 56, VDS_OBJECT_CANNOT_GET_LOCK = 57, VDS_REACHED_THE_END = 58, VDS_INVALID_ITERATOR = 59,

  VDS_OBJECT_NAME_TOO_LONG = 60, VDS_FOLDER_IS_NOT_EMPTY = 61,

VDS_ITEM_ALREADY_PRESENT = 62, VDS_NO_SUCH_ITEM = 63,

VDS_OBJECT_IS_DELETED = 64, VDS_OBJECT_NOT_INITIALIZED = 65 }

6.3.1 Enumeration Type Documentation

6.3.1.1 enum vdsErrors

Enumerator:

VDS_OK No error.

..

VDS_INTERNAL_ERROR Abnormal internal error - it should not happen!

VDS_ENGINE_BUSY Cannot get a lock on a system object, the engine is "busy".

This might be the result of either a very busy system where unused cpu cycles are rare or a lock might be held by a crashed process.

VDS_NOT_ENOUGH_VDS_MEMORY Not enough memory in the VDS.

VDS_NOT_ENOUGH_HEAP_MEMORY Not enough heap memory (non-VDS memory).

VDS_NOT_ENOUGH_RESOURCES There are not enough resources to correctly process the call.

This might be due to a lack of POSIX semaphores on systems where locks are implemented that way or a failure in initializing a pthread_mutex (or on Windows, a critical section).

VDS_WRONG_TYPE_HANDLE The provided handle is of the wrong type.

VDS_NULL_HANDLE The provided handle is NULL (zero).

VDS_NULL_POINTER One of the arguments of an API function is an invalid NULL pointer.

VDS_INVALID_LENGTH An invalid length was provided (it will usually indicate that the length value is set to zero.

VDS_PROCESS_ALREADY_INITIALIZED The process was already initialized.

Was vdsInit() called for a second time?

VDS_PROCESS_NOT_INITIALIZED The process was not properly initialized.

Was vdsInit() called?

VDS_INVALID_WATCHDOG_ADDRESS The watchdog address is invalid (empty string, NULL pointer, etc.

)

VDS_INCOMPATIBLE_VERSIONS API - memory-file version mismatch.

VDS_SOCKET_ERROR Generic socket error.

VDS_CONNECT_ERROR Socket error when trying to connect to the watchdog.

VDS_SEND_ERROR Socket error when trying to send a request to the watchdog.

VDS_RECEIVE_ERROR Socket error when trying to receive a reply from the watchdog.

VDS_BACKSTORE_FILE_MISSING The vds backstore file is missing (the name of this file is provided by the watchdog).

VDS_ERROR_OPENING_VDS Generic i/o error when attempting to open the vds.

VDS_LOGFILE_ERROR Error accessing the directory for the log files or error opening the log file itself.

VDS_SESSION_CANNOT_GET_LOCK Cannot get a lock on the session (a pthread_mutex or a critical section on Windows).

VDS_SESSION_IS_TERMINATED An attempt was made to use a session object (a session handle) after this session was terminated.

VDS_INVALID_OBJECT_NAME Permitted characters for names are alphanumerics, spaces (' '), dashes ('-') and underlines ('_').

The first character must be alphanumeric.

VDS_NO_SUCH_OBJECT The object was not found (but its folder does exist).

VDS_NO_SUCH_FOLDER One of the parent folder of an object does not exist.

VDS_OBJECT_ALREADY_PRESENT Attempt to create an object which already exists.

VDS_IS_EMPTY The object (data container) is empty.

VDS_WRONG_OBJECT_TYPE Attempt to create an object of an unknown object type.

VDS_OBJECT_CANNOT_GET_LOCK Cannot get lock on the object.

This might be the result of either a very busy system where unused cpu cycles are rare or a lock might be held by a crashed process.

VDS_REACHED_THE_END The search/iteration reached the end without finding a new item/record.

VDS_INVALID_ITERATOR An invalid value was used for a vds-IteratorType parameter.

VDS_OBJECT_NAME_TOO_LONG The name of the object is too long.

The maximum length of a name cannot be more than VDS_MAX_-NAME_LENGTH (or VDS_MAX_FULL_NAME_LENGTH for the fully qualified name).

VDS_FOLDER_IS_NOT_EMPTY You cannot delete a folder if there are still undeleted objects in it.

Technical: a folder does not need to be empty to be deleted but all objects in it must be "marked as deleted" by the current session. This enables writing recursive deletions

VDS_ITEM_ALREADY_PRESENT An item with the same key was found.

VDS_NO_SUCH_ITEM The item was not found in the hash map.

VDS_OBJECT_IS_DELETED The object is scheduled to be deleted soon.

Operations on this data container are not permitted at this time.

VDS_OBJECT_NOT_INITIALIZED Object must be open first before you can access them.

Definition at line 27 of file vdsErrors.h.

6.4 /home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsFolder.h File Reference

`#include <vdsf/vdsCommon.h>`

Include dependency graph for vdsFolder.h:

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

Functions

- VDSF_EXPORT int vdsFolderClose (VDS_HANDLE objectHandle)
- VDSF_EXPORT int vdsFolderGetFirst (VDS_HANDLE objectHandle, vdsFolderEntry *pEntry)
- VDSF_EXPORT int vdsFolderGetNext (VDS_HANDLE objectHandle, vdsFolderEntry *pEntry)
- VDSF_EXPORT int vdsFolderOpen (VDS_HANDLE sessionHandle, const char *folderName, size_t nameLengthInBytes, VDS_HANDLE *objectHandle)
- VDSF_EXPORT int vdsFolderStatus (VDS_HANDLE objectHandle, vdsObjStatus *pStatus)

6.4.1 Function Documentation

6.4.1.1 VDSF_EXPORT int vdsFolderClose (VDS_HANDLE *objectHandle*)

6.4.1.2 VDSF_EXPORT int vdsFolderGetFirst (VDS_HANDLE *objectHandle*, vdsFolderEntry * *pEntry*)

6.4.1.3 VDSF_EXPORT int vdsFolderGetNext (VDS_HANDLE *objectHandle*, vdsFolderEntry * *pEntry*)

6.4.1.4 VDSF_EXPORT int vdsFolderOpen (VDS_HANDLE *sessionHandle*, const char * *folderName*, size_t *nameLengthInBytes*, VDS_HANDLE * *objectHandle*)

6.4.1.5 VDSF_EXPORT int vdsFolderStatus (VDS_HANDLE *objectHandle*, vdsObjStatus * *pStatus*)

6.5 /home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsHashMap.h File Reference

`#include <vdsf/vdsCommon.h>`

Include dependency graph for vdsHashMap.h:

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

Functions

- VDSF_EXPORT int vdsHashMapClose (VDS_HANDLE objectHandle)
- VDSF_EXPORT int vdsHashMapDelete (VDS_HANDLE objectHandle, const void *key, size_t keyLength)
- VDSF_EXPORT int vdsHashMapGet (VDS_HANDLE objectHandle, const void *key, size_t keyLength, void *buffer, size_t bufferLength, size_t *returnedLength)
- VDSF_EXPORT int vdsHashMapGetFirst (VDS_HANDLE objectHandle, void *buffer, size_t bufferLength, size_t *returnedLength)
- VDSF_EXPORT int vdsHashMapGetNext (VDS_HANDLE objectHandle, void *buffer, size_t bufferLength, size_t *returnedLength)
- VDSF_EXPORT int vdsHashMapInsert (VDS_HANDLE objectHandle, const void *key, size_t keyLength, const void *data, size_t dataLength)
- VDSF_EXPORT int vdsHashMapOpen (VDS_HANDLE sessionHandle, const char *hashMapName, size_t nameLengthInBytes, VDS_HANDLE *objectHandle)
- VDSF_EXPORT int vdsHashMapStatus (VDS_HANDLE objectHandle, vdsObjStatus *pStatus)

6.5.1 Function Documentation

6.5.1.1 VDSF_EXPORT int vdsHashMapClose (VDS_HANDLE *objectHandle*)

6.5.1.2 VDSF_EXPORT int vdsHashMapDelete (VDS_HANDLE *objectHandle*, const void * *key*, size_t *keyLength*)

6.5.1.3 VDSF_EXPORT int vdsHashMapGet (VDS_HANDLE *objectHandle*, const void * *key*, size_t *keyLength*, void * *buffer*, size_t *bufferLength*, size_t * *returnedLength*)

6.5.1.4 VDSF_EXPORT int vdsHashMapGetFirst (VDS_HANDLE *objectHandle*, void * *buffer*, size_t *bufferLength*, size_t * *returnedLength*)

6.5.1.5 VDSF_EXPORT int vdsHashMapGetNext (VDS_HANDLE *objectHandle*, void * *buffer*, size_t *bufferLength*, size_t * *returnedLength*)

6.5.1.6 VDSF_EXPORT int vdsHashMapInsert (VDS_HANDLE *objectHandle*, const void * *key*, size_t *keyLength*, const void * *data*, size_t *dataLength*)

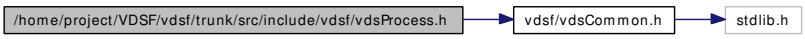
6.5.1.7 VDSF_EXPORT int vdsHashMapOpen (VDS_HANDLE *sessionHandle*, const char * *hashMapName*, size_t *nameLengthInBytes*, VDS_HANDLE * *objectHandle*)

6.5.1.8 VDSF_EXPORT int vdsHashMapStatus (VDS_HANDLE *objectHandle*, vdsObjStatus * *pStatus*)

6.6 /home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsProcess.h File Reference

`#include <vdsf/vdsCommon.h>`

Include dependency graph for vdsProcess.h:



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

Functions

- VDSF_EXPORT void vdsExit (VDS_HANDLE processHandle)

  This function terminates all access to the VDS.

- VDSF_EXPORT int vdsInit (const char *wdAddress, int protectionNeeded, VDS_HANDLE *processHandle)

  This function initializes access to a VDS.

6.6.1 Function Documentation

6.6.1.1 VDSF_EXPORT void vdsExit (VDS_HANDLE *processHandle*)

This function terminates all access to the VDS.

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

This function takes a single argument, the handle to the process object and always end successfully.

6.6.1.2 VDSF_EXPORT int vdsInit (const char * *wdAddress*, int *protectionNeeded*, VDS_HANDLE * *processHandle*)

This function initializes access to a VDS.

It takes 2 input arguments, the address of the watchdog and an integer (used as a boolean, 0 for false, 1 for true) to indicate if sessions and other objects (Queues, etc) are shared amongst threads (in the current process) and must be protected. Recommendation: always set protectionNeeded to 0 (false) unless you cannot do it otherwise. In other words it is recommended to use one session handle for each thread. Also if the same queue needs to be accessed by two threads it is more efficient to have two different handles 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.]

Upon successful completion, the process handle is set. Otherwise the error code is returned.

6.7 /home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsQueue.h File Reference

6.7.1 Detailed Description

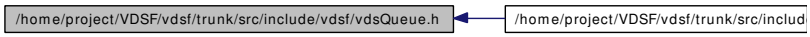
This files provides the API to access a VDSF FIFO queue.

Definition in file vdsQueue.h.

`#include <vdsf/vdsCommon.h>`

Include dependency graph for vdsQueue.h:

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



Functions

- VDSF_EXPORT int vdsQueueClose (VDS_HANDLE objectHandle)

  Close a FIFO queue.

- VDSF_EXPORT int vdsQueueGetFirst (VDS_HANDLE objectHandle, void *buffer, size_t bufferLength, size_t *returnedLength)

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

- VDSF_EXPORT int vdsQueueGetNext (VDS_HANDLE objectHandle, void *buffer, size_t bufferLength, size_t *returnedLength)

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

- VDSF_EXPORT int vdsQueueOpen (VDS_HANDLE sessionHandle, const char *queueName, size_t nameLengthInBytes, VDS_HANDLE *objectHandle)

  Open an existing FIFO queue (see vdsCreateObject to create a new queue).

- VDSF_EXPORT int vdsQueuePop (VDS_HANDLE objectHandle, 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.

- VDSF_EXPORT int vdsQueuePush (VDS_HANDLE objectHandle, const void *pItem, size_t length)

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

- VDSF_EXPORT int vdsQueueStatus (VDS_HANDLE objectHandle, vdsObjStatus *pStatus)

  Return the status of the queue.

6.7.2 Function Documentation

6.7.2.1 VDSF_EXPORT int vdsQueueClose (VDS_HANDLE *objectHandle*)

Close a FIFO queue.

This function terminates the current access to the queue in shared memory (the queue 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 vdsCommit or vdsRollback to end the current unit of work.

Parameters:

← ***objectHandle*** The handle to the queue (see vdsQueueOpen).

Returns:

0 on success or a vdsErrors on error.

6.7.2.2 VDSF_EXPORT int vdsQueueGetFirst (VDS_HANDLE *objectHandle*, 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:

- ← ***objectHandle*** The handle to the queue (see vdsQueueOpen).
- ↔ ***buffer*** The buffer provided by the user to hold the content of the first element. Memory allocation for this buffer is the responsability 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 a vdsErrors on error.

6.7.2.3 VDSF_EXPORT int vdsQueueGetNext (VDS_HANDLE *objectHandle*, 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 vdsQueueGetFirst to initialize the iterator. Not so evident - calling vdsQueuePop will reset the iteration to the last element (they use the same internal storage). If this cause a problem, please let us know.

Parameters:

- ← ***objectHandle*** The handle to the queue (see vdsQueueOpen).
- ↔ ***buffer*** The buffer provided by the user to hold the content of the first element. Memory allocation for this buffer is the responsability 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 a vdsErrors on error.

6.7.2.4 VDSF_EXPORT int vdsQueueOpen (VDS_HANDLE *sessionHandle*, const char * *queueName*, size_t *nameLengthInBytes*, VDS_HANDLE * *objectHandle*)

Open an existing FIFO queue (see vdsCreateObject to create a new queue).

Parameters:

← ***sessionHandle*** The handle to the current session.

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

← ***nameLengthInBytes*** The length of *queueName* (in bytes) not counting the null terminator (null-terminators are not used by the vdsf engine).

→ ***objectHandle*** The handle to the queue, allowing us access to the queue in shared memory. On error, this handle will be set to zero (NULL) unless the objectHandle pointer itself is NULL.

Returns:

0 on success or a vdsErrors on error.

6.7.2.5 VDSF_EXPORT int vdsQueuePop (VDS_HANDLE *objectHandle*, 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 vdsCommit.

Parameters:

- ← ***objectHandle*** The handle to the queue (see vdsQueueOpen).
- → ***buffer*** The buffer provided by the user to hold the content of the data item. Memory allocation for this buffer is the responsability 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 a vdsErrors on error.

6.7.2.6 VDSF_EXPORT int vdsQueuePush (VDS_HANDLE *objectHandle*, 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 vdsCommit.

Parameters:

- ← ***objectHandle*** The handle to the queue (see vdsQueueOpen).
- ← ***pItem*** The data item to be inserted.
- ← ***length*** The length of *pItem* (in bytes).

Returns:

0 on success or a vdsErrors on error.

6.7.2.7 VDSF_EXPORT int vdsQueueStatus (VDS_HANDLE *objectHandle*, vdsObjStatus * *pStatus*)

Return the status of the queue.

Parameters:

- ← ***objectHandle*** The handle to the queue (see vdsQueueOpen).
- → ***pStatus*** A pointer to the status structure.

Returns:

0 on success or a vdsErrors on error.

6.8 /home/project/VDSF/vdsf/trunk/src/include/vdsf/vdsSession.h File Reference

`#include <vdsf/vdsCommon.h>`

Include dependency graph for vdsSession.h:

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

Functions

- VDSF_EXPORT int vdsInitSession (VDS_HANDLE *sessionHandle)
  This function initializes a session.
- VDSF_EXPORT int vdsCommit (VDS_HANDLE handle)
- VDSF_EXPORT int vdsCreateObject (VDS_HANDLE handle, const char *objectName, size_t nameLengthInBytes, vdsObjectType objectType)
- VDSF_EXPORT int vdsDestroyObject (VDS_HANDLE handle, const char *objectName, size_t nameLengthInBytes)
- VDSF_EXPORT int vdsErrorMsg (VDS_HANDLE sessionHandle, char *message, size_t msgLengthInBytes)
- VDSF_EXPORT int vdsExitSession (VDS_HANDLE handle)
- VDSF_EXPORT int vdsGetInfo (VDS_HANDLE sessionHandle, vdsInfo *pInfo)
- VDSF_EXPORT int vdsGetStatus (VDS_HANDLE handle, const char *objectName, size_t nameLengthInBytes, vdsObjStatus *pStatus)
- VDSF_EXPORT int vdsLastError (VDS_HANDLE sessionHandle)
- VDSF_EXPORT int vdsRollback (VDS_HANDLE handle)

6.8.1 Function Documentation

6.8.1.1 VDSF_EXPORT int vdsCommit (VDS_HANDLE *handle*)

6.8.1.2 VDSF_EXPORT int vdsCreateObject (VDS_HANDLE *handle*, const char * *objectName*, size_t *nameLengthInBytes*, vdsObjectType *objectType*)

6.8.1.3 VDSF_EXPORT int vdsDestroyObject (VDS_HANDLE *handle*, const char * *objectName*, size_t *nameLengthInBytes*)

6.8.1.4 VDSF_EXPORT int vdsErrorMsg (VDS_HANDLE *sessionHandle*, char * *message*, size_t *msgLengthInBytes*)

6.8.1.5 VDSF_EXPORT int vdsExitSession (VDS_HANDLE *handle*)

6.8.1.6 VDSF_EXPORT int vdsGetInfo (VDS_HANDLE *sessionHandle*, vdsInfo * *pInfo*)

6.8.1.7 VDSF_EXPORT int vdsGetStatus (VDS_HANDLE *handle*, const char * *objectName*, size_t *nameLengthInBytes*, vdsObjStatus * *pStatus*)

6.8.1.8 VDSF_EXPORT int vdsInitSession (VDS_HANDLE * *sessionHandle*)

This function initializes a session.

It takes one output argument, the session handle.

Upon successful completion, the session handle is set and the function returns zero. Otherwise the error code is returned and the handle is set to NULL.

This function will also initiate a new transaction:

Contrary to some other transaction management software, almost every call made is part of a transaction. Even viewing data (for example deleting the data by another session will be delayed until the current session terminates its access).

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

6.8.1.9 VDSF_EXPORT int vdsLastError (VDS_HANDLE *sessionHandle*)

6.8.1.10 VDSF_EXPORT int vdsRollback (VDS_HANDLE *handle*)