2

Storage Device API Description

Connection Functions

CCODE
NWSMListSDIs -- Data Requestor API

```
                    ( char *pattern,
                      NWSM_NAME_LIST **nameList);
```

Parameters

pattern	(INPUT) Passes the pattern to search for. *pattern* can be set to "*", a string with "*", or an exact match.
nameList	(OUTPUT) Passes the address of a pointer and returns a list of available local SDIs (see Appendix B for more information). **Note:** Do not pass a structure.

Completion Codes

0x0	Successful
0xFFFEFFFD	NWSMDR_OUT_OF_MEMORY

Type

Waiting

Prerequisites

None

Remarks

This function returns the specified available local SDIs. If an SDI is not found, Successful is returned and *nameList* is set to null. To select an SDI, select a name from the list and call **NWSMSDConnectToSDI**.

Example

See Also

NWSMSDConnectToSDI

CCODE
NWSMSDConnectToSDI -- Data Requestor API

```
( STRING sdiName,
  STRING sdiUserName,
  void *reserved,
  UINT32 *connection);
```

Parameters

sdiName	(INPUT) Passes a name that **NWSMListSDIs** returned.
sdiUserName	(INPUT) Passes a user's name.
reserved	Reserved
connection	(OUTPUT) Returns a connection handle that is used for all subsequent SDI function calls.

Completion Codes

0x0	Successful
0xFFFCFFD4	NWSMSD_TIME_OUT
0xFFFCFFDA	NWSMSD_OUT_OF_MEMORY
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFD	NWSMDR_OUT_OF_MEMORY
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Type

Waiting

Prerequisites

None

Remarks

This function connects to the specified SDI.

Note: Under SMS for NetWare v4.0, SDI and SME must be in the same file service.

See Also

NWSMSDReleaseSDI

CCODE
NWSMSDReleaseSDI -- Data Requestor API

```
( UINT32 *connection);
```

Parameters

connection	(INPUT) Passes a handle that **NWSMSDConnectToSDI** returned.

Completion Codes

0x0	Successful
0xFFFCFFD4	NWSMSD_TIME_OUT
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Type

Waiting

Remarks

On return, *connection* is set to zero.

Example

See Also

NWSMSDConnectToSDI

Initialization Functions

CCODE
NWSMSDListDevices

```
( UINT32 connection,
  NWSMSD_DEVICE_LIST *deviceList);
```

Parameters

connection	(INPUT) Passes a handle that **NWSMSDConnectToSDI** returned.
deviceList	(OUTPUT) Passes a pointer to an NWSMSD_DEVICE_LIST structure and returns a list of devices available to the SDI. See Appendix B for more information.

Completion Codes

0x0	Successful. If no devices are available, 0x0 is still returned and *deviceList* is null.
0xFFFCFFF1	NWSMSD_INVALID_PARAMETER
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFCFFFA	NWSMSD_DEVICE_LIST_CHANGED
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Type

Waiting

Prerequisites

None

Remarks

This function returns a list of devices available through the SDI. Some local devices may not be available to SDI, because they may have been assigned to other engines. Set *deviceList.deviceID* to 0 if the engine wants to retrieve all the devices or set it to the desired device ID that the list should begin with.

See Also

NWSMSDConnectToSDI
NWSMSDSubjugateDevice

CCODE
NWSMSDListMedia

```
( UINT32 connection,
  NWSMSD_DEVICE_HANDLE deviceHandle,
  NWBOOLEAN reScan,
  void *reserved0,
  NWSMSD_MEDIA_LIST *mediaList,
  CCODE *completionStatus);
```

Parameters

connection	(INPUT) Passes a handle that **NWSMSDConnectToSDI** returned.
deviceHandle	(INPUT) Passes a device handle that **NWSMSDMountMedia** or **NWSMSDSubjugateDevice** returned. If *deviceHandle* is zeroed, SDI returns a list of all available media on all devices. If this parameter is nonzero, SDI returns the media information available on the specified device. For more information about NWSMSD_DEVICE_HANDLE, see **NWSMSDMountMedia**.
reScan	(INPUT) If set to TRUE, SDI updates its media list by rescanning all available media. If set to FALSE, SDI returns its media list without updating it.
reserved0	Reserved
mediaList	(INPUT/OUTPUT) Returns a list of available media. mediaList->totalCount is 0 if the list is empty (see Appendix B for more information).
completionStatus	(OUTPUT) Returns the same value as the function's return value.

Completion Codes

0x0	Successful
0xFFFCFFF1	NWSMSD_INVALID_PARAMETER
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFCFFFB	NWSMSD_DEVICE_H_INVALID
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Type

Waiting

Prerequisites

None

Remarks

This function returns a list of accessible media. The list includes media mounted in any device that is listed by **NWSMSDListDevices**. If no media are available, the **NWSMSDListMedia** returns successfully.

SDI may not be informed of new media, removed media, etc. Therefore, the engine must tell SDI to rescan if it suspects there is a change in SDI's environment.

See Also

NWSMSDConnectToSDI
NWSMSDListMedia
NWSMSDSubjugateMedia
NWSMSDSubjugateDevice
NWSMSDMountMedia

CCODE
NWSMSDSubjugateDevice

```
( UINT32 connection,
  NWSMSD_DEVICE_ID *deviceDesc,
  UINT32 deviceReadWriteMode,
  NWSMSD_DEVICE_HANDLE *deviceHandle,
  CCODE *completionStatus);
```

Parameters

connection	(INPUT) Passes a handle that was returned by **NWSMSDConnectToSDI**.
deviceDesc	(INPUT) Passes the type of device the engine wants to use. If a null is passed, SDI uses any available device. To select a specific device, pass a pointer to an NWSMSD_DEVICE_ID structure that was returned by **NWSMSDListDevices**.
deviceReadWriteMode	(INPUT) Passes the desired access rights to the specified device. These access rights are enforced on all operations associated with *connection*, until the engine changes it by calling this function again. For access rights definitions, see the read/write mode parameter description of **NWSMSDMountMedia**.
deviceHandle	(OUTPUT) *deviceHandle* is valid if the call is successful.
completionStatus	(OUTPUT) Returns the same value as the function's return value.

Completion Codes

0x0	Successful
0xFFFCFFD0	NWSMSD_UNSUPPORTED_SERVICE
0xFFFCFFDA	NWSMSD_OUT_OF_MEMORY
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFCFFF8	NWSMSD_DEVICE_NOT_EXIST
0xFFFCFFF9	NWSMSD_DEVICE_NOT_AVAIL
0xFFFCFFFB	NWSMSD_DEVICE_H_INVALID
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Type

Waiting

Prerequisites

None

Remarks

This function performs an "open" and a "reserve" on the device for the requesting engine. This function is optional and is used when an engine wants a specific device for future operations.

See Also

NWSMSDSubjugateMedia
NWSMSDMountMedia
NWSMSDEmancipateDevice

CCODE
NWSMSDEmancipateDevice

```
( UINT32 connection,
  NWSMSD_DEVICE_HANDLE *deviceHandle);
```

Parameters

connection	(INPUT) Passes a handle that **NWSMSDConnectToSDI** returned.
deviceHandle	(INPUT) Passes a device handle that **NWSMSDMountMedia** or **NWSMSDSubjugateDevice** returned.

Completion Codes

0x0	Successful
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFCFFFB	NWSMSD_DEVICE_H_INVALID
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Type

Waiting

Prerequisites

The engine must subjugate the device with **NWSMSDSubjugateDevice** or **NWSMSDMountMedia**. Also, the session must be closed and the media dismounted before calling this function.

Remarks

This function releases and closes the device for the engine. *deviceHandle* is invalid when this call returns.

See Also

NWSMSDDismountMedia
NWSMSDEmancipateMedia

CCODE
NWSMSDSubjugateMedia

```
( UINT32 connection,
  NWSMSD_MEDIA_ID *mediaDesc,
  UINT32 mediaReadWriteMode,
  void *reserved,
  NWSMSD_MEDIA_HANDLE *mediaHandle,
  CCODE *completionStatus);
```

Parameters

connection	(INPUT) Passes a handle that **NWSMSDConnectToSDI** returned.
mediaDesc	(INPUT) Passes a description of a media that **NWSMSDListMedia** returned. The media is described by setting *mediaDesc*'s fields to the appropriate values. For more information about this structure, see Appendix B.
mediaReadWriteMode	(INPUT) Passes the access modes. These can be ORed together. The access modes are enforced on all operations associated with *connection* until this function is called again. If the media overflows, the next media inherits the mode. For a description of the modes, see the read/write mode parameter of **NWSMSDMountMedia.**
reserved	Reserved
mediaHandle	(OUTPUT) Returns a media handle. *mediaHandle* is valid if the call is successful.
completionStatus	(OUTPUT) Returns the same value as the function's return value.

Completion Codes

0x0	Successful
0xFFFCFFD0	NWSMSD_UNSUPPORTED_SERVICE
0xFFFCFFDA	NWSMSD_OUT_OF_MEMORY
0xFFFCFFE4	NWSMSD_MEDIA_NOT_EXIST
0xFFFCFFE5	NWSMSD_MEDIA_NOT_AVAIL
0xFFFCFFEB	NWSMSD_MEDIA_H_INVALID
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Type

Waiting

Prerequisites

None

Remarks

This function reserves the media for the engine and returns a media handle for future media actions (e.g., mounting and moving). The function or **NWSMSDMountMedia** must be called before performing any action on the media (this function is called by **NWSMSDMountMedia** if necessary).

Note: If no match is found, this call fails.

See Also

NWSMSDSubjugateDevice
NWSMSDMountMedia
NWSMSDEmancipateMedia

CCODE
NWSMSDEmancipateMedia

```
( UINT32 connection,
  NWSMSD_MEDIA_HANDLE *mediaHandle);
```

Parameters

connection	(INPUT) Passes a handle that **NWSMSDConnectToSDI** returned.
mediaHandle	(INPUT) Passes the media handle that **NWSMSDMountMedia** or **NWSMSDSubjugateMedia** returned.

Completion Codes

0x0	Successful
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION!
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION

Type

Waiting

Prerequisites

The media was subjugated by **NWSMSDSubjugateMedia** or **NWSMSDMountMedia**.

Remarks

The function releases and closes the media reserved by the engine. *mediaHandle* is invalid after the call returns.

See Also

NWSMSDDismountMedia
NWSMSDEmancipateMedia

CCODE
NWSMSDMountMedia

```
( UINT32 connection,
  NWSMSD_DEVICE_ID *deviceDesc,
  UINT32 deviceReadWriteMode,
  NWSMSD_MEDIA_ID *mediaDesc,
  UINT32 mediaReadWriteMode,
  void *reserved,
  NWSMSD_DEVICE_HANDLE *deviceHandle,
  NWSMSD_MEDIA_HANDLE *mediaHandle,
  CCODE *completionStatus);
```

Parameters

connection	(INPUT) Passes a handle that was returned by **NWSMSDConnectToSDI**.
deviceDesc	(INPUT) Passes a description of the device to reserve. This parameter is used if *deviceHandle* is zeroed. If *deviceDesc* is null and *deviceHandle* is zero, SDI uses the default device specified by **NWSMSDSetReadSDIDefaults**. If no default device is specified, the first device that can perform the required operations is used. If *deviceDesc* is not null, SDI passes it to **NWSMSDSubjugateDevice**, which reserves the device.
deviceReadWriteMode	(INPUT) Passes a device open mode bit map for the device to be mounted. The modes can be ORed together. See "Remarks" for more information.
mediaDesc	(INPUT) This parameter is used only if *mediaHandle* is zeroed. If *mediaHandle* is zeroed and *mediaDesc* is null, SDI uses the default media specified by **NWSMSDSetReadSDIDefaults**. If no default media is specified, the first media that can perform the required operations is used. If *mediaDesc* is not null, SDI passes it to **NWSMSDSubjugateMedia**, which reserves the media. **Note:** See Appendix B for more information about this structure.
mediaReadWriteMode	(INPUT) Passes a media open mode bit map for the media to be mounted. The modes can be ORed together. If the media overflows, the next media inherits these mode. See "Remarks" for more information.
reserved	Not Used
deviceHandle	(INPUT/OUTPUT) Passes a device handle. If *deviceHandle* refers to a device (handle is not zero), *deviceDesc* is ignored and the device is mounted. If *deviceHandle* is zero, this function uses *deviceDesc* to determine the device to use. The resulting device handle is placed into this parameter. **Note:** This field requires a pointer to a handle. Do not pass a null pointer.

mediaHandle	(INPUT/OUTPUT) Passes a media handle. If *mediaHandle* refers to a media (handle is not zero), this function ignores *mediaDesc* and mounts the media. If *mediaHandle* is zeroed, this function uses *mediaDesc* to determine the media to use. The resulting media handle is placed into this parameter. **Note:** This field requires a pointer to a handle. Do not pass a null pointer.
completionStatus	(OUTPUT) Not Used.

Completion Codes

0x0	Successful
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFF1	NWSMSD_INVALID_PARAMETER
0xFFFCFFDA	NWSMSD_OUT_OF_MEMORY
0xFFFCFFF9	NWSMSD_DEVICE_NOT_AVAIL
0xFFFCFFD0	NWSMSD_UNSUPPORTED_SERVICE
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFCFFE5	NWSMSD_MEDIA_NOT_AVAIL
0xFFFCFFF8	NWSMSD_DEVICE_NOT_EXIST
0xFFFCFFD4	NWSMSD_TIME_OUT
0xFFFCFFCB	NWSMSD_DRIVER_UNSUPPORT_FUNC
0xFFFCFFCD	NWSMSD_IO_ABORT_SUCCESSFUL
0xFFFCFFEE	NWSMSD_LOCATION_INVALID
0xFFFCFFFF	NWSMSD_ACCESS_DENIED
0xFFFCFFE6	NWSMSD_MEDIA_MOUNTED
0xFFFCFFFB	NWSMSD_DEVICE_H_INVALID
0xFFFCFFEB	NWSMSD_MEDIA_H_INVALID
0xFFFCFFE4	NWSMSD_MEDIA_NOT_EXIST
0xFFFFFFFF	NWSMSD_WAIT_PENDING
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION

Type

Waiting

Prerequisites

None

Remarks

This function performs all the remaining operations necessary to make the media ready for other functions such as opening a session, repositioning, labeling, deleting, etc. By using this function, an engine can reserve the media and device as follows:

- If the desired device/media is already subjugated, the engine passes the appropriate handle(s) to this function.
- If the engine sets all descriptions to null and handles to zero, SDI uses the device and/or media specified by **NWSMSDSetReadSDIDefaults**. If no defaults were set, SDI uses the first device and/or media that can perform the requested operation.

Set *mediaReadWriteMode* and/or *deviceReadWriteMode* to one or more of the following:

NWSMSD_WRITE_MODE	Opens the media or device as write only.
NWSMSD_READ_MODE	Opens the media or device as read only.

See Also

NWSMSDSubjugateMedia
NWSMSDSubjugateDevice
NWSMSDCloseSession
NWSMSDDismountMedia

CCODE
NWSMSDDismountMedia

```
( UINT32 connection,
  NWSMSD_DEVICE_HANDLE *deviceHandle,
  NWSMSD_MEDIA_HANDLE *mediaHandle,
  UINT32 dismountMode,
  NWSMSD_HEADER_BUFFER *mediaTrailerInfo,
  CCODE *completionStatus);
```

Parameters

connection	(INPUT) Passes a handle that **NWSMSDConnectToSDI** returned.
deviceHandle	(INPUT) Passes a handle that **NWSMSDMountMedia** or **NWSMSDSubjugateDevice** returned.
mediaHandle	(INPUT) Passes a handle that **NWSMSDMountMedia** or **NWSMSDSubjugateMedia** returned.
dismountMode	(INPUT) Passes an emancipation bit map that tells SDI how to release a device/media. One device flag and one media flag must be ORed together (see "Remarks" for more information).
mediaTrailerInfo	(INPUT) Passes the SIDF data to be placed into the media trailer. If this parameter is null, SDI places its own media trailer onto the media. The engine decides what it wants to put into the media trailer; SDI will fill in the rest. For more information about the media trailer, see the *System Independent Data Format* document. **Note:** SDI may reallocate this buffer.
completionStatus	(OUTPUT) Returns NWSMSD_WAIT_PENDING.

Completion Codes

0x0	Successful
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFEC	NWSMSD_MEDIA_FAILED
0xFFFCFFF1	NWSMSD_INVALID_PARAMETER
0xFFFCFFDA	NWSMSD_OUT_OF_MEMORY
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFFFFFF	NWSMSD_WAIT_PENDING
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION

Type

Waiting

Prerequisites

The media was mounted by using **NWSMSDMountMedia**. The session must be closed before this function is called.

Remarks

This function releases the devices/media according to the emancipation modes (the media is not ejected). This function also causes SDI to write a media index if the engine wrote to the media via **NWSMSDWriteSessionData**.

dismountMode must have one device mode flag and one media mode flag set, and an optional trailer flag set. The flags are:

Device flags (choose only one):

NWSMSD_AUTO_EMANCIPATE_DEVICE
Emancipate only the device(s) that are subjugated by **NWSMSDMountMedia**. This flag's value is zero.

NWSMSD_DONT_EMANCIPATE_DEVICE
Do not emancipate any device.

NWSMSD_UNCOND_EMANCIPATE_DEVICE
Emancipate all devices.

Media flags (choose only one):

NWSMSD_AUTO_EMANCIPATE_MEDIA
Emancipate only the media that are subjugated by **NWSMSDMountMedia**. This flag's value is 0.

NWSMSD_DONT_EMANCIPATE_MEDIA
Do not emancipate any media.

NWSMSD_UNCOND_EMANCIPATE_MEDIA
Emancipate all media.

Trailer flag (optional):

NWSMSD_WRITE_TRAILER
Write the media trailer before dismounting the media. This is usually done by SDI when the media overflows. We do not recommend that the engine do this. If the engine writes the trailer, it should only be done if the media set is being closed permanently.

See Also

NWSMSDMountMedia
NWSMSDCloseSession
NWSMSDEmancipateMedia
NWSMSDEmancipateDevice

Read/Write Functions

CCODE
NWSMSDOpenSessionForWriting

```
( UINT32 connection,
  NWSMSD_MEDIA_HANDLE mediaHandle,
  NWSMSD_HEADER_BUFFER *sessionHeaderInfo,
  NWSMSD_TRANSFER_BUF_INFO *transferBufferInfo,
  NWSMSD_SESSION_HANDLE *sessionHandle,
  CCODE *completionStatus);
```

Parameters

connection	(INPUT) Passes the handle that **NWSMSDConnectToSDI** returned.
mediaHandle	(INPUT) Passes the media handle that was return by **NWSMSDMountMedia** or **NWSMSDSubjugateMedia**.
sessionHeaderInfo	(INPUT) (Optional) Passes SIDF-formatted session header information. If no session header is passed, SDI writes its own session header onto the media. **NWSMSetSessionHeaderInfo** can be used to format the data (see *Storage Management Services Utilities Library*). For more information about FIDs and session headers see *System Independent Data Format*. **Note:** The engine decides what it needs to put into the header; SDI will fill in the rest.
transferBufferInfo	(INPUT/OUTPUT) Passes the engine's specifications for the transfer buffer's header sizes, offsets, and sector sizes. On return, this parameter contains SDI's final word on these values (i.e., this parameter is used for buffer size negotiation). The buffer size returned by SDI is a maximum size, not an absolute size; however, the header area must be the exact size specified. This means that the engine can give to SDI a transfer buffer whose size is anywhere between the header size (only a header is being passed) and the maximum size. For a discussion about transfer buffers, see Chapter One "Introduction" and *Storage Management Services Architecture*.
sessionHandle	(OUTPUT) Returns a session handle, but it is not valid until the function returns successfully.
completionStatus	(OUTPUT) See "Types" section for more information.

Completion Codes

0x0	Successful
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFD4	NWSMSD_TIME_OUT
0xFFFCFFEB	NWSMSD_MEDIA_H_INVALID
0xFFFCFFF1	NWSMSD_INVALID_PARAMETER
0xFFFCFFC7	NWSMSD_HEADER_TOO_LARGE
0xFFFCFFDA	NWSMSD_OUT_OF_MEMORY
0xFFFCFFFE	NWSMSD_BUFFER_INCORRECT
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFFFFFF	NWSMSD_WAIT_PENDING
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION

Type

Nonwaiting. When the function returns, the engine must poll *completionStatus* until it is not NWSMSD_WAIT_PENDING. *completionStatus* will be set to one of the above completion codes.

Prerequisites

NWSMSDMountMedia

Remarks

This function behaves like a file system's open function. It performs all the remaining operations necessary to make the media ready for the write functions. If successful, a valid handle is returned that is used for all subsequent calls.

This function repositions the media to the end of recorded data and writes the session header contained in the transfer buffer. The engine fills in the fields particular to it, and SDI fills in the rest to make it SIDF-compliant.

NWSMSD_BUFFER_INCORRECT is returned when the session header information:

- Is formatted incorrectly
- May contain the one or more of the following fields:

  session header
  offset to end
  crc type

transfer buffer size

- May not contain the required fields (see *System Independent Data Format* for more information).

See Also

NWSMSDMountMedia
NWSMSDWriteSessionData

CCODE
NWSMSDWriteSessionData

```
( UINT32 connection,
  NWSMSD_CONTROL_BLOCK *controlBlock);
```

Parameters

connection	(INPUT) Passes the handle that was returned by **NWSMSDConnectToSDI**.
controlBlock	(INPUT) Passes a control block and returns the location of the transfer buffer on the media. The transfer buffer contains the data to be written to the media. The engine must allocate memory for the control block and transfer buffer (**NWSMSDOpenSessionForWriting** returns the sizes for these buffers). See Appendix B for more information.

Completion Codes

0x0	Successful
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFF1	NWSMSD_INVALID_PARAMETER
0xFFFCFFDA	NWSMSD_OUT_OF_MEMORY
0xFFFCFFFE	NWSMSD_BUFFER_INCORRECT
0xFFFCFFEC	NWSMSD_MEDIA_FAILED
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFCFFD4	NWSMSD_TIME_OUT
0xFFFCFFD6	NWSMSD_SESSION_H_INVALID
0xFFFCFFCD	NWSMSD_IO_ABORT_SUCCESSFUL
0xFFFCFFCB	NWSMSD_DRIVER_UNSUPPORT_FUNC
0xFFFCFFCF	NWSMSD_OS_ERROR
0xFFFFFFFF	NWSMSD_WAIT_PENDING
0xFFFCFFD0	NWSMSD_UNSUPPORTED_SERVICE
0xFFFCFFF7	NWSMSD_EARLY_WARNING
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION

Type

Nonwaiting. When this function returns, the engine must loop until *controlBlock->completionStatus* is not NWSMSD_WAIT_PENDING. *controlBlock->completionStatus* will then be set to one of the above completion codes.

Prerequisites

NWSMSDOpenSessionForWriting

Remarks

This function writes the data onto the media. The engine must set *controlBlock->sessionDataType* to the transfer buffer's data type. See Appendix B for more information.

Before putting the data into the transfer buffer's data area, the engine must put the data set information and data set data into a record (see *System Independent Data Format* for more details).

> **Caution**: Under SDI 1.0 only, the maximum transfer buffer is 256kb.

See Also

NWSMSDOpenSessionForWriting
NWSMSDCloseSession

CCODE
NWSMSDOpenSessionForReading

```
( UINT32 connection,
  NWSMSD_MEDIA_HANDLE mediaHandle,
  NWSMSD_SESSION_ID *sessionDesc,
  void *reserved0,
  NWSMSD_HEADER_BUFFER *sessionHeader,
  UINT32 *sectorSize,
  UINT32 *transferBufferSize.
  NWSMSD_SESSION_HANDLE *sessionHandle,
  CCODE *completionStatus);
```

Parameters

connection	(INPUT) Passes a handle that was returned by **NWSMSDConnectToSDI**.
mediaHandle	(INPUT) Passes a media handle that was returned by **NWSMSDMountMedia** or **NWSMSDSubjugateMedia**.
sessionDesc	(INPUT) (Optional) Passes a pointer to an NWSMSD_SESSION_ID structure that describes the session to open. If it is null, the next session is opened.
reserved0	Reserved
sessionHeader	(OUTPUT) (Optional) SDI copies the session header into this buffer. If a null pointer is passed, no information is returned. **NWSMGetSessionHeaderInfo** can be used to deformat the session header (see *Storage Management Services Utilities Library*). **Note:** SDI may reallocate this buffer.
sectorSize	(OUTPUT) Returns the smallest readable unit that SDI may put into the transfer buffer. The engine memory allocation must be a multiple of this size.
transferBufferSize	(OUTPUT) Returns the maximum size of the transfer buffer (the buffer size is a multiple of *sectorSize*). The engine must allocate a transfer buffer of this size.
sessionHandle	(OUTPUT) Returns a session handle, but it is not valid until the engine's callback function returns successfully.
completionStatus	(OUTPUT) See "Type" section for more information.

Completion Codes

0x0	Successful
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFEB	NWSMSD_MEDIA_H_INVALID
0xFFFCFFF1	NWSMSD_INVALID_PARAMETER
0xFFFCFFDA	NWSMSD_OUT_OF_MEMORY
0xFFFCFFD4	NWSMSD_TIME_OUT
0xFFFCFFCD	NWSMSD_IO_ABORT_SUCCESSFUL
0xFFFCFFD0	NWSMSD_UNSUPPORTED_SERVICE
0xFFFCFFC7	NWSMSD_HEADER_TOO_LARGE
0xFFFCFFCB	NWSMSD_DRIVER_UNSUPPORT_FUNC
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFFFFFF	NWSMSD_WAIT_PENDING
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Type

Nonwaiting. When this function returns, the engine must loop on *completionStatus* until it is not NWSMSD_WAIT_PENDING. *completionStatus* will be set to one of the above completion codes.

Prerequisites

The media must be mounted by **NWSMSDMountMedia**.

Remarks

This function behaves like a file system's open function. It performs all the remaining operations necessary to make the media ready for the read function. If successful, this function returns a valid handle that is used for all subsequent calls.

This function physically positions the media at the beginning of the session by using *sessionHeader* to identify the recorded session.

Example

See Also

NWSMSDMountMedia
NWSMSDReadSessionData

CCODE
NWSMSDReadSessionData
(UINT32 connection,
NWSMSD_CONTROL_BLOCK *controlBlock);

Parameters

connection	(Input) Passes the handle that was returned by **NWSMSDConnectToSDI**.
controlBlock	(INPUT/OUTPUT) Points to a control block. The engine must allocate memory for the control block and the transfer buffer. The size of the transfer buffer was returned by **NWSMSDOpenSessionForReading**.

Completion Codes

0x0	Successful
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFFD	NWSMSD_BUFFER_SIZE_INVALID
0xFFFCFFDA	NWSMSD_OUT_OF_MEMORY
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFCFFD4	NWSMSD_TIME_OUT
0xFFFCFFD6	NWSMSD_SESSION_H_INVALID
0xFFFCFFEB	NWSMSD_MEDIA_H_INVALID
0xFFFCFFD0	NWSMSD_UNSUPPORTED_SERVICE
0xFFFCFFCF	NWSMSD_OS_ERROR
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION

Type

Nonwaiting. When this function returns, the engine must poll *controlBlock->completionStatus* until it is not NWSMSD_WAIT_PENDING. *controlBlock->completionStatus* will then be set to one of the above completion codes.

Prerequisites

NWSMSDOpenSessionForReading.

Remarks

This function reads data from the session associated with *connection* into *controlBlock*'s transfer buffer. Before returning the completion status to the engine, SDI ensures that it has the correct buffer by comparing the block header against a buffer identification structure.

> **Note:** Before the engine sends the data set data back to the TSA, the data must be taken out of the record or subrecord sections. For more information, see *System Independent Data Format*.
>
> **Caution**: Under SDI 1.0 only, the maximum transfer buffer is 256kb.

SDI sets *controlBlock->sessionDataType* to the type of data being read. The data type was set by the engine when **NWSMSDWriteSessionData** was called. For more information about the data types, see NWSMSD_CONTROL_BLOCK in Appendix B.

The engine gets the session trailer, session index, etc., by continually issuing read calls as shown in the following statements. If the last block of data is about to be read from the media, the following read sequences show what is retrieved as the engine issues read calls (each numbered line represents one **NWSMSDReadSessionData** function call).

> **Note:** Some sections shown below are not required and consequently will not be seen when a read call is issued. *sessionDataType*, *sectorsNotTransfer*, and *transferBuffer* are fields in *controlBlock*.

1 File system data in the transfer buffer
 sessionDataType = NWSMSD_TSA_DATA
 sectorsNotTransfer = 0

2 File system data in the transfer buffer
 sessionDataType = NWSMSD_END_OF_TSA_DATA
 sectorsNotTransfer = a nonzero value

4 Session trailer data in the transfer buffer
 sessionDataType = NWSMSD_SESSION_TRAILER
 sectorsNotTransfer = 0

6 Session index data in the transfer buffer
 sessionDataType = NWSMSD_SESSION_INDEX
 sectorsNotTransfer = 0

7 Session index data in the transfer buffer
 sessionDataType = NWSMSD_SESSION_INDEX
 sectorsNotTransfer = a nonzero value

9 Media index data in the transfer buffer
sessionDataType = NWSMSD_MEDIA_INDEX
sectorsNotTransfer = 0

10 Media index data in the transfer buffer
sessionDataType = NWSMSD_MEDIA_INDEX
sectorsNotTransfer = a nonzero value

11 No data in the transfer buffer
sessionDataType = NWSMSD_END_OF_SESSION
sectorsNotTransfer = 0

See Also

NWSMSDOpenSessionForReading
NWSMSDCloseSession

Termination Functions

```
CCODE
NWSMSDCloseSession
                ( UINT32 connection,
                  NWSMSD_SESSION_HANDLE *sessionHandle,
                  CCODE *completionStatus);
```

Parameters

connection	(INPUT) Passes the handle that was returned by **NWSMSDConnectToSDI**.
sessionHandle	(INPUT) Passes a session handle to close and invalidate.
completionStatus	(OUTPUT) See section "Types" for more information.

Completion Codes

0x0	Successful
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFF1	NWSMSD_INVALID_PARAMETER
0xFFFCFFD6	NWSMSD_SESSION_H_INVALID
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFCFFDA	NWSMSD_OUT_OF_MEMORY
0xFFFCFFEB	NWSMSD_MEDIA_H_INVALID
0xFFFFFFFF	NWSMSD_WAITING_PENDING
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION

Type

Nonwaiting. When this function returns, the engine must poll *completionStatus* until it is not NWSMSD_WAIT_PENDING. *completionStatus* will be set to one of the above completion codes.

Remarks

This function closes a read or write session. For a write session, if the session index is already written, this function writes a media index onto the media, then closes the session (at SDI's discretion). If the session index has not been written, this function writes a session trailer, writes the media index, and then closes the session (see *System Independent Data Format* for more information about media index, session index, and session trailer).

> **Note:** Once the session index is written, SDI prevents any action on the medium from any engine, until this function is called.

If NWSMSD_INVALID_PARAMETER is returned, it could be that *sessionHandle* was never initialized.

See Also

NWSMSDDismountMedia

CCODE

NWSMSDCancelDataTransfer

```
( UINT32 connection,
  NWSMSD_CONTROL_BLOCK *controlBlock);
```

Parameters

connection	(INPUT) Passes the handle that was returned by **NWSMSDConnectToSDI**.
controlBlock	(INPUT) Passes a control block that is used by **NWSMSDReadSessionData** or **NWSMSDWriteSessionData**.

Completion Codes

0x0	Successful
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFF1	NWSMSD_INVALID_PARAMETER
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFCFFEB	NWSMSD_MEDIA_H_INVALID
0xFFFFFFFF	NWSMSD_WAIT_PENDING
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION

Type

Waiting

Prerequisites

The session was opened by **NWSMSDOpenSessionForReading** or **NWSMSDOpenSessionForWriting**.

Remarks

This function cancels the data transfer associated with *conttrolBlock*.

Miscellaneous Functions

CCODE
NWSMSDFormatMedia(NEW)

```
( UINT32 connection,
  NWSMSD_MEDIA_HANDLE mediaHandle,
  UINT32 operationType,
  UINT32 numberOfPartitions,
  CAPACITY *partitionSizeArray,
  CCODE *completionStatus);
```

Parameters

connection	(INPUT) Passes a handle that was returned by **NWSMSDConnectToSDI**.
mediaHandle	(INPUT) Passes a handle that was returned by **NWSMSDMountMedia** or **NWSMSDSubjugateMedia**.
operationType	(INPUT) Passes one or more of the following bit mapped commands: NWSMSD_FORMAT_MEDIA (0x00000001) Perform a low-level format on the media. NWSMSD_PARTITION_MEDIA (0x00000002) Partition the media. Partitioning occurs after formatting if formatting is also specified.
numberOfPartitions	(INPUT) Number of partitions specified in *partitionSizeArray*.
partitionSizeArray	(INPUT) Passes an array of partition sizes to create on the media. The position of each element corresponds to a logical partition number (i.e., element 0 is logical partition 0 and element 1 is logical partition 1). The last partition *must* contain a -1 (see remarks for more information). **Note:** The order of the logical partitions, may *not* be the physical layout of the partitions on the media.
completionStatus	(OUTPUT) Returns the same value as the function's return value.

Completion Codes

0x0	Successful
0xFFFCFFF1	NWSMSD_INVALID_PARAMETER
0xFFFCFFEB	NWSMSD_MEDIA_H_INVALID
0xFFFCFFE2	NWSMSD_MEDIA_NOT_MOUNTED
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFD0	NWSMSD_UNSUPPORTED_SERVICE

0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFCFFCD	NWSMSD_IO_ABORT_SUCCESSFUL
0xFFFFFFFF	NWSMSD_WAIT_PENDING
0xFFFCFFCB	NWSMSD_DRIVER_UNSUPPORT_FUNC
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION

Type

Waiting

Prerequisites

Call **NWSMSDMountMedia** with the mode set to write access.

Remarks

This function formats a medium. The engine must know how the physical partitions are arranged on the media, because the -1 for *partitionSizeArray* must be in the array element that represents the last physical partition.

For example, if medium started with physical partition 2 and ended with physical partition 0, the partition size array should be set as follows:

```
partitionSizeArray[0] = -1;
partitionSizeArray[1] = desired size;
partitionSizeArray[2] = desired size;
```

If the medium started with physical partition 0 and ended with physical partition 2, the array should be set as follows:

```
partitionSizeArray[0] = desired size;
partitionSizeArray[1] = desired size;
partitionSizeArray[2] = -1;
```

Note: The size of the last partition (represented by the element with -1) is the medium's size minus the combined sizes of the other partitions.

See Also

NWSMSDLabelMedia

CCODE
NWSMSDLabelMedia

```
( UINT32 connection,
  NWSMSD_MEDIA_HANDLE mediaHandle,
  NWSMSD_HEADER_BUFFER *mediaHeaderInfo,
  CCODE *completionStatus);
```

Parameters

connection	(INPUT) Passes a handle that was returned by **NWSMSDConnectToSDI**.
mediaHandle	(INPUT) Passes a media handle that was returned by **NWSMSDMountMedia** or **NWSMSDSubjugateMedia**.
mediaHeaderInfo	(INPUT) Passes the media header information. The engine is required to pass a media label and media number (all other information is optional). This data must be formatted according to SIDF's specifications. **NWSMSetMediaHeaderInfo** can format the media header information (see *Storage Management Services Library*). For more information about media headers, see *System Independent Data Format*.
completionStatus	(OUTPUT) Returns the same value as the function's return value.

Completion Codes

0x0	Successful
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFE2	NWSMSD_MEDIA_NOT_MOUNTED
0xFFFCFFF1	NWSMSD_INVALID_PARAMETER
0xFFFCFFFE	NWSMSD_BUFFER_INCORRECT
0xFFFCFFDA	NWSMSD_OUT_OF_MEMORY
0xFFFCFFDC	NWSMSD_NO_WRITE_MODE
0xFFFCFFD4	NWSMSD_TIME_OUT
0xFFFCFFEB	NWSMSD_MEDIA_H_INVALID
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFCFFC7	NWSMSD_HEADER_TOO_LARGE
0xFFFCFFD0	NWSMSD_UNSUPPORTED_SERVICE
0xFFFCFFCD	NWSMSD_IO_ABORT_SUCCESSFUL
0xFFFCFFCB	NWSMSD_DRIVER_UNSUPPORT_FUNC
0xFFFFFFFF	NWSMSD_WAIT_PENDING

0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION

Type

Waiting

Prerequisites

Call **NWSMSDMountMedia** with the mode set to NWSMSD_WRITE_MODE.

Remarks

This function places a media header onto an empty media. Any previous media header must be erased, or this function will fail. The transfer buffer contains the media header. SDI will add any necessary FIDs to make the media comply with the SIDF specifications.

NWSMSD_BUFFER_INCORRECT will be returned if *mediaHeaderInfo*:

- Is formatted incorrectly
- Contains the one or more of the following fields:
  media header
  offset to end
  revision level
  physical sector size
- Does not contain the required fields (see *System Independent Data Format* for more information).

See Also

NWSMSDMountMedia
NWSMSDDeleteMedia
NWSMSDLabelDevice

CCODE
NWSMSDDeleteMedia

```
( UINT32 connection,
  NWSMSD_MEDIA_HANDLE mediaHandle,
  UINT32 deleteMode,
  CCODE *completionStatus);
```

Parameters

connection	(INPUT) Passes a handle that was returned by **NWSMSDConnectToSDI**.
mediaHandle	(INPUT) Passes a media handle that was returned by **NWSMSDMountMedia** or **NWSMSDSubjugateMedia**.
deleteMode	(INPUT) This parameter is not currently implemented, but will be used in future releases of SDI.
completionStatus	(OUTPUT) Returns the same value as the function’s return value.

Completion Codes

0x0	Successful
0xFFFCFFEB	NWSMSD_MEDIA_H_INVALID
0xFFFCFFDC	NWSMSD_NO_WRITE_MODE
0xFFFCFFD4	NWSMSD_TIME_OUT
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFE2	NWSMSD_MEDIA_NOT_MOUNTED
0xFFFCFFD0	NWSMSD_UNSUPPORTED_SERVICE
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFCFFCD	NWSMSD_IO_ABORT_SUCCESSFUL
0xFFFCFFCB	NWSMSD_DRIVER_UNSUPPORT_FUNC
0xFFFFFFFF	NWSMSD_WAIT_PENDING
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION

Type

Waiting

Prerequisites

Call **NWSMSDMountMedia** with the mode set to write access.

Remarks

This function removes the media header from the media, which makes it an empty media.

See Also

NWSMSDLabelMedia

CCODE
NWSMSDReturnMediaHeader

```
( UINT32 connection,
  NWSMSD_MEDIA_HANDLE mediaHandle,
  NWBOOLEAN verifyHeader,
  NWSMSD_HEADER_BUFFER *mediaHeader,
  CCODE *completionStatus);
```

Parameters

connection	(INPUT) Passes a handle that was returned by **NWSMSDConnectToSDI**.
mediaHandle	(INPUT) Passes a media handle that was returned by **NWSMSDMountMedia** or **NWSMSDSubjugateMedia**.
verifyHeader	(INPUT) If *verifyHeader* is set to TRUE, SDI rereads and verifies the media header. If the media header does not match SDI's copy, NWSMSD_HEADER_MISMATCH is returned.
mediaHeader	(OUTPUT) Returns the media header. If no media header is present, this function returns successfully and sets *mediaHeader->headerSize* to 0.
completionStatus	(OUTPUT) Returns one of the completion codes.

Completion Codes

0x0	Successful
0xFFFCFFEB	NWSMSD_MEDIA_H_INVALID
0xFFFCFFE2	NWSMSD_MEDIA_NOT_MOUNTED
0xFFFCFFF1	NWSMSD_INVALID_PARAMETER
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFDA	NWSMSD_OUT_OF_MEMORY
0xFFFCFFE5	NWSMSD_MEDIA_NOT_AVAIL
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFCFFCD	NWSMSD_IO_ABORT_SUCCESSFUL
0xFFFCFFCB	NWSMSD_DRIVER_UNSUPPORT_FUNC
0xFFFCFFD9	NWSMSD_NOT_SMS_MEDIA
0xFFFCFFF5	NWSMSD_MEDIA_CHANGED
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION!
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION

Type

Waiting

Prerequisites

The media must be subjugated if *verifyHeader* is TRUE.

Remarks

This function returns a media header.

See Also

NWSMSDLabelMedia

CCODE
NWSMSDPositionMedia

```
( UINT32 connection,
  NWSMSD_MEDIA_HANDLE mediaHandle,
  NWSMSD_MEDIA_POSITION *mediaPosition,
  UINT32 positionMode,
  void *reserved0,
  CCODE *completionStatus);
```

Parameters

connection	(INPUT) Passes a handle that was returned by **NWSMSDConnectToSDI**.
mediaHandle	(INPUT) Passes a media handle that was returned by **NWSMSDMountMedia** or **NWSMSDSubjugateMedia**.
mediaPosition	(INPUT/OUTPUT) Passes the media positioning values. *positionMode* determines whether this is an input or output structure (see the Remarks section. Also see Appendix B for more information).
positionMode	(INPUT) Passes a positioning mode (see the Remarks section).
reserved	Reserved
completionStatus	(OUTPUT) Returns the same value as the function's return value.

Completion Codes

0x0	Successful
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFF1	NWSMSD_INVALID_PARAMETER
0xFFFCFFD7	NWSMSD_POSITION_NOT_FOUND
0xFFFCFFD0	NWSMSD_UNSUPPORTED_SERVICE
0xFFFCFFDA	NWSMSD_OUT_OF_MEMORY
0xFFFCFFEC	NWSMSD_MEDIA_FAILED
0xFFFCFFE5	NWSMSD_MEDIA_NOT_AVAIL
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFCFFEB	NWSMSD_MEDIA_H_INVALID
0xFFFCFFD8	NWSMSD_POSITION_INVALID
0xFFFCFFCD	NWSMSD_IO_ABORT_SUCCESSFUL
0xFFFCFFCB	NWSMSD_DRIVER_UNSUPPORT_FUNC
0xFFFFFFFF	NWSMSD_WAIT_PENDING
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION

Type

Waiting

Prerequisites

NWSMSDMountMedia.

Remarks

This function positions or reports the current position of the media. NWSMSD_POSITION_INQUIRE is the only mode that does not reposition the media. All other modes attempt to reposition the media. The following list shows the media positioning modes:

NWSMSD_POSITION_INQUIRE (0x00000001)
Places the current media position into *mediaPosition*. All values use the absolute member of the union (i.e., C's union).

NWSMSD_POSITION_SECTOR_ABS (0x00000007)
Positions the media forward *mediaPosition.sectorAddress.absolute* sectors from the beginning of the session.

NWSMSD_POSITION_PARTITION_ABS (0x00000008)
Positions the media at *mediaPosition->partitionNumber*. This is an absolute value and is zero-indexed (i.e., starts from zero).

NWSMSD_REWIND_MEDIA (0x0000000A)
Rewind media to the beginning of the partition.

NWSMSD_POSITION_END_OF_MEDIA (0x0000000C)
Positions the media at the end of SIDF recorded media (logical end of media). If the device cannot sense logical end of media, NWSMSD_UNSUPPORTED_FUNCTION is returned.

Example

CCODE
NWSMSDMoveMedia

```
( UINT32 connection,
  NWSMSD_MEDIA_HANDLE mediaHandle,
  UINT32 moveMode,
  NWSMSD_MEDIA_LOCATION *mediaLocation,
  CCODE *completionStatus);
```

Parameters

connection	(INPUT) Passes a handle that was returned by **NWSMSDConnectToSDI**.
mediaHandle	(INPUT) Passes a media handle that was returned by **NWSMSDMountMedia** or **NWSMSDSubjugateMedia**.
moveMode	(INPUT) Get the media's current position or move the media (see the Remarks section for more information).
mediaLocation	(INPUT/OUTPUT) Passes the new media location or returns the location of the media.
completionStatus	(OUTPUT) Returns the same value as the function's return value.

Completion Codes

0x0	Successful
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFEB	NWSMSD_MEDIA_H_INVALID
0xFFFCFFFF	NWSMSD_ACCESS_DENIED
0xFFFCFFE6	NWSMSD_MEDIA_MOUNTED
0xFFFCFFEE	NWSMSD_LOCATION_INVALID
0xFFFCFFF1	NWSMSD_INVALID_PARAMETER
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFCFFD0	NWSMSD_UNSUPPORTED_SERVICE
0xFFFCFFE5	NWSMSD_MEDIA_NOT_AVAIL
0xFFFCFFCD	NWSMSD_IO_ABORT_SUCCESSFUL
0xFFFCFFCB	NWSMSD_DRIVER_UNSUPPORT_FUNC
0xFFFCFFE4	NWSMSD_MEDIA_NOT_EXIST
0xFFFFFFFF	NWSMSD_WAIT_PENDING
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION

Type

Waiting

Prerequisites

None

Remarks

Under SMS for NetWare v4.0 this function only ejects the media. Set *moveMode* to:

NWSMSD_MOVE_EJECT (0x00000004)
: If the device was subjugated or mounted with a relation type[1] of NWSMSD_DEVICE_SINGLE_MEDIA, the media is ejected.

If the engine tries to move a mounted medium, this function returns NWSMSD_MEDIA_MOUNTED. If the media is not subjugated, NWSMSD_ACCESS_DENIED is returned.

[1] Relation type is the relation of the device to the media. See structure NWSMSD_DEVICE_ID for more information.

CCODE
NWSMSDGetDeviceStatus

```
( UINT32 connection,
  NWSMSD_DEVICE_HANDLE deviceHandle,
  NWSMSD_DEVICE_ID *deviceDesc,
  NWSMSD_DEVICE_STATUS *deviceStatus);
```

Parameters

connection	(INPUT) Passes a handle that was returned by **NWSMSDConnectToSDI**.
deviceHandle	(INPUT) Passes a device handle that was returned by **NWSMSDMountMedia** or **NWSMSDSubjugateDevice**. If *deviceHandle* is not zeroed, SDI returns the status of the device specified by *deviceHandle*. If *deviceHandle* is zero, SDI returns the status of the device described by *deviceDesc*.
deviceDesc	(INPUT) If *deviceHandle* is zeroed, the function searches for a device that matches uniqueDeviceID of NWSMSD_DEVICE_ID. See Appendix B for more information.
deviceStatus	(OUTPUT) Returns the device's status (see "NWSMSD_DEVICE_STATUS" in Appendix B for more information).

Completion Codes

0x0	Successful
0xFFFCFFFB	NWSMSD_DEVICE_H_INVALID
0xFFFCFFF1	NWSMSD_INVALID_PARAMETER
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION

Type

Waiting

Prerequisites

If *deviceHandle* is not zero, the device must be subjugated by **NWSMSDSubjugateDevice** or **NWSMSDMountMedia**.

Remarks

This function returns the status of the requested device.

CCODE
NWSMSDGetMediaStatus

```
( UINT32 connection,
  NWSMSD_MEDIA_HANDLE mediaHandle,
  NWSMSD_MEDIA_ID *mediaDesc,
  NWSMSD_MEDIA_STATUS *mediaStatus);
```

Parameters

connection	(INPUT) Passes a handle that was returned by **NWSMSDConnectToSDI**.
mediaHandle	(INPUT) Passes a media handle that was returned by **NWSMSDMountMedia** or **NWSMSDSubjugateMedia**. If *mediaHandle* is not zero, SDI returns the status of the media specified by *mediaHandle*. If *mediaHandle* is zero, SDI returns the status of the device described by *mediaDesc*.
mediaDesc	(INPUT) Passes a description of the type of media to look for. If *mediaHandle* is zeroed, the function searches for a media that matches the non-NWSMSD_DONT_CARE values of *mediaDesc*. If the description applies to more than one media, the first medium is returned. For more information about this structure, see Appendix B.
mediaStatus	(OUTPUT) Returns the device's status (see Appendix B for more information).

Completion Codes

0x0	Successful
0xFFFCFFF1	NWSMSD_INVALID_PARAMETER
0xFFFCFFEB	NWSMSD_MEDIA_H_INVALID
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION

Type

Waiting

Prerequisites

If *mediaHandle* is not zero, subjugate the media with **NWSMSDSubjugateMedia** or **NWSMSDMountMedia**.

Remarks

This function returns the status of the requested media.

CCODE
NWSMSDGetDeviceCharacteristics

```
( UINT32 connection,
  NWSMSD_DEVICE_HANDLE deviceHandle,
  NWSMSD_DEVICE_ID *deviceCharacteristics);
```

Parameters

connection	(INPUT) Passes a handle that was returned by **NWSMSDConnectToSDI**.
deviceHandle	(INPUT) Passes a device handle that was returned by **NWSMSDMountMedia** or **NWSMSDSubjugateDevice**.
deviceCharacteristics	(OUTPUT) Passes a pointer to a structure and returns the device's characteristics (see the Appendix B for more information).

Completion Codes

0x0	Successful
0xFFFCFFFB	NWSMSD_DEVICE_H_INVALID
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Type

Waiting

Prerequisites

The device was subjugated with **NWSMSDSubjugateDevice** or **NWSMSDMountMedia**.

Remarks

This function returns the characteristics of the specified device.

CCODE
NWSMSDGetMediaCharacteristics

```
( UINT32 connection,
  NWSMSD_MEDIA_HANDLE mediaHandle,
  NWSMSD_MEDIA_ID *mediaCharacteristics);
```

Parameters

connection	(INPUT) Passes a handle that was returned by **NWSMSDConnectToSDI**.
mediaHandle	(INPUT) Passes a media handle that was returned by **NWSMSDMountMedia** or **NWSMSDSubjugateMedia**.
mediaCharacteristics	(OUTPUT) Passes a pointer to a structure and returns the media's characteristics.

Completion Codes

0x0	Successful
0xFFFCFFEB	NWSMSD_MEDIA_H_INVALID
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Type

Waiting

Prerequisites

The engine must subjugate the media with **NWSMSDSubjugateMedia** or **NWSMSDMountMedia**.

Remarks

This function returns the characteristics of a specified media.

CCODE
NWSMSDLabelDevice

```
( UINT32 connection,
  NWSMSD_MEDIA_HANDLE deviceHandle,
  BUFFERPTR deviceLabel,);
```

Parameters

connection	(INPUT) Passes a handle that was returned by **NWSMSDConnectToSDI**.
deviceHandle	(INPUT) Passes a device handle that was returned by **NWSMSDMountMedia** or **NWSMSDSubjugateDevice**.
deviceLabel	(INPUT/OUTPUT) Passes a NWSM_MAX_DEVICE_LABEL_LEN buffer containing the device's label. After the device is renamed, SDI will query the device for its name and copy it into this buffer for verification by the engine.

Completion Codes

0x0	Successful
0xFFFCFFF1	NWSMSD_INVALID_PARAMETER
0xFFFCFFFF	NWSMSD_ACCESS_DENIED
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFFB	NWSMSD_DEVICE_H_INVALID
0xFFFCFFD4	NWSMSD_TIME_OUT
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER

Type

Waiting

Prerequisites

None

Remarks

This function provides a means of giving the device a user friendly name. Unless this function is called, SDI uses the name it retrieved from the device via the Media Manager. This name is usually not very "user-friendly."

CCODE
NWSMSDSetReadSDIDefaults

```
( UINT32 connection,
  NWSMSD_SDI_DEFAULTS *sdiDefaults,
  NWBOOLEAN setReadMode);
```

Parameters

connection	(INPUT) Passes the handle that was returned by **NWSMSDConnectToSDI**.
sdiDefaults	(INPUT/OUTPUT) Passes the new defaults, or returns the current default settings for each SDI-nonwaiting function.
setReadMode	(INPUT) If set to TRUE, SDI reads *sdiDefaults* into its default settings. If set to FALSE, SDI writes its default settings into *sdiDefaults* .

Completion Codes

0x0	Successful
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Type

Waiting

Prerequisites

None

Remarks

This function provides a way for the engine to specify or discover:

- The currently defined default devices.
- The timeout values for each SDI waiting function (see the NWSMSD_SDI_DEFAULTS structure).
- The default media.

The defaults apply only to the current connection.

CCODE
NWSMSDRegisterAlertRoutine

```
( UINT32 connection,
  UINT32 alertType,
  NWSMSDAlertRoutine *alertRoutine);
```

Parameters

connection	(INPUT) Passes the handle that was returned by **NWSMSDConnectToSDI**.
alertType	(INPUT) Passes a bit map that describes the alert message types the engine wants to know about (see "Remarks" for more information).
alertRoutine	(INPUT) Passes a function pointer that SDI calls when a qualifying alert occurs (see "NWSMSDAlertRoutine" in this chapter for more information). **Note:** Pass a null function pointer to permanently disable previously set alerts.

Completion Codes

0x0	Successful
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Type

Waiting

Prerequisites

None

Remarks

This function registers an engine's alert routine. *alertType* indicates the type of messages filters an engine wants to receive.

> **Note:** Alerts are not multiplexed (i.e., the engine is notified of only one alert at a time).

The alert types are:

NWSMSD_NEW_MEDIA (0x00000001)
: SDI retrieved the next media during a read or write session. **AlertRoutine** should not call **NWSMSDAlertResponse**, because this is only a notification.

NWSMSD_DELETED_MEDIA (0x00000002)
The media for this *mediaHandle* was deleted from underneath SDI. The mediaHandle is no longer valid.

NWSMSD_NEW_MEDIA_NEEDED (0x00000004)
SDI ran out of media and needs user input. When a medium is inserted, alertRoutine must call **NWSMSDAlertResponse** to tell SDI of the event.

NWSMSD_NEW_MEDIA_NOT_BLANK (0x00000008)
The new media has a media header. A media without a header is needed.

NWSMSD_NEW_MEDIA_INCORRECT (0x00000010)
The media being spanned to is not the correct media. **AlertRoutine** should not call **NWSMSDAlertResponse**. **AlertRoutine** will be called again with NWSMSD_NEW_MEDIA_NEEDED.

NWSMSD_DEVICE_ONLINE (0x00000080)

NWSMSD_DEVICE_OFFLINE (0x00000100)

See Also

AlertRoutine
NWSMSDAlertResponse

NWSMSDAlertRoutine*

AlertRoutine(Engine Provided Function)

```
( UINT32 alertHandle,
  UINT32 alertType,
  UINT32 uniqueID,
  UINT32 alertNumber,
  STRING alertString);
```

Parameters

alertHandle	(INPUT) SDI passes this handle into this function when calling this function.
alertType	(OUTPUT) Returns the engine's alert bit map to **AlertRoutine**.
uniqueID	(OUTPUT) Returns the device's ID that caused the alert.
alertNumber	(OUTPUT) Returns a number that should be used with alertString (see the Remarks section for more information).
alertString	(OUTPUT) Returns a double-byte string[2] that the engine may display as part of its alert handling. Sometimes a null string is returned. **Note:** It is assumed that when **AlertRoutine** returns to SDI, that *string* 's buffer is available for reuse or removal by SDI. Therefore, we advise that **AlertRoutine** copies *string* before returning to SDI.

Type

Waiting

Prerequisites

None

Remarks

SDI calls this function to notify the engine of an alert (this function is written by the engine developer). The engine should call **NWSMSDAlertResponse** to inform SDI of the engine's reaction to the alert.

alertNumber and *alertString* are used to form a complete message. For example, if *alertString* is "Device hardware failure," *alertNumber* returns the device's number. Both parameters could be used as follows:

```
printf("%s, Device Number: 0x%x\n", alertString,
   alertNumber);
```

[2] Double byte strings have characters that use two bytes instead of one. This is needed to be NetWare v4.0 compliant. For more information, see the CLIB documentation.

See Also

NWSMSDRegisterAlertRoutine
NWSMSDAlertResponse

CCODE
NWSMSDAlertResponse

```
( UINT32 connection,
  UINT32 alertHandle,
  UINT32 alertType,
  UINT32 alertResponseValue);
```

Parameters

connection	(INPUT) Passes the handle that was returned by **NWSMSDConnectToSDI**.
alertHandle	(INPUT) Passes the handle returned by **AlertRoutine**.
alertType	(INPUT) This is the same bit map defined in by **NWSMSDRegisterAlertRoutine**. It informs the calling module of the alert type being responded to.
alertResponseValue	(INPUT) Passes the engine's response to the alert (see the remarks for more information).

Completion Codes

0x0	Successful
0xFFFCFFF1	NWSMSD_INVALID_PARAMETER
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION

Type

Waiting

Prerequisites

None

Remarks

The engine calls this function to tell SDI what SDI needs to do in response to an alert type. The following responses are defined:

Alert Type	Response (*alertResponseValue*)
NWSMSD_NEW_MEDIA_NEEDED	NWSMSD_NEW_MEDIA_CONTINUE (0x00000000) A new media has been inserted, and SDI needs to mount it. NWSMSD_NEW_MEDIA_ABORT (0x00000001) No media are available, abort spanning.
NWSMSD_MEDIA_NOT_BLANK	NWSMSD_NEW_MEDIA_CONTINUE (0x00000000) A new media has been inserted; SDI needs to mount it. NWSMSD_NEW_MEDIA_ABORT (0x00000001) Do not use this medium; dismount it and request a new one.

CODE
NWSMSDConvertValueToMessage

```
( UINT32 connection,
  UINT32 valueType,
  UINT32 value,
  UINT32 stringLen,
  STRING string);
```

Parameters

connection	(INPUT) Passes the handle that was returned by **NWSMSDConnectToSDI**.
valueType	(INPUT) Passes the value's type to be converted (e.g., media type). See the Remarks section for more information.
value	(INPUT) Passes the value to be converted to a string.
stringLen	(INPUT) Passes *string*'s buffer size. If the message is longer than the buffer, it is truncated.
string	(OUTPUT) Passes a *stringLen* byte buffer and returns the string associated with *value*.

Completion Codes

0x0	Successful
0xFFFCFFF3	NWSMSD_INVALID_CONNECTION
0xFFFCFFF4	NWSMSD_INTERNAL_ERROR
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Type

Waiting

Prerequisites

None

Remarks

This function translates an SDI value (i.e., a constant beginning with NWSMSD_) into a language-enabled (double byte) string[3] that can be sent to a console, an error log, etc.

Set *valueType* to one of the following:

NWSMSD_VALUE_TYPE_MEDIA (0x00000001)
Set *value* to a media type.

[3] Double byte strings have characters that use two bytes instead of one. This is needed to be compliant with NetWare's current environment requirements. For more information, see your CLIB documentation.

NWSMSD_VALUE_TYPE_DEVICE (0x00000002)
Set *value* to a device type.

NWSMSD_VALUE_TYPE_OBJECT (0x00000003)
Use this type to get strings for NWSMSD_MEDIA_LOCATION.objectType.

NWSMSD_VALUE_TYPE_RELATION (0x00000004)
Use this type to get strings for NWSMSD_DEVICE_ID.deviceRelation.

NWSMSD_VALUE_TYPE_RESERVED (0x00000005)
Use this type to get strings for NWSMSD_DEVICE_ID.reservedStatus or NWSMSD_MEDIA_ID.reservedStatus.

NWSMSD_VALUE_TYPE_MODE (0x00000006)
Use this type to get strings for NWSMSD_OBJECT_STATUS.objectMode (subjugation mode).

NWSMSD_VALUE_TYPE_MOUNTED (0x00000007)
Use this type to get string for NWSMSD_MEDIA_STATUS.mediaMounted.

NWSMSD_VALUE_TYPE_OWNER (0x00000008)
Use this type to get strings for NWSMSD_MEDIA_ID.mediaOwner.

NWSMSD_VALUE_TYPE_CAPACITY (0x00000009)
Use this type to get strings for CAPACITY.factor.

NWSMSD_VALUE_TYPE_OPERATION (0x0000000A)
Use this type to get strings for NWSMSD_OBJECT_STATUS.objectOperation.