B

Data Types and Structures

Data Types

BUFFER	unsigned char
BUFFERPTR	unsigned char *
CCODE	unsigned long
NWSMSD_DEVICE_HANDLE	UINT32
NWSMSD_MEDIA_HANDLE	UINT32
NWSMSD_SESSION_HANDLE	UINT32
STRING	unsigned char
UINT16	16-bit unsigned integer
UINT32	32-bit unsigned long

Data Structures
NWSMSD_MEDIA_LOCATION

```
typedef struct
{
   UINT32 objectType;
   UINT32 uniqueDeviceID;
   UINT32 reserved0;
   UINT32 reserved1;
} NWSMSD_OBJECT_LOCATION;
```

objectType is the type of object that contains or will contain the media:

NWSMSD_DEVICE
NWSMSD_STORAGE_BAY

uniqueDeviceID passes or returns the device's ID. This value is contained in the NWSMSD_DEVICE_ID structure, which is set by **NWSMSDSubjugateMedia**, **NWSMSDMountMedia**, or **NWSMSDGetDeviceCharacteristics**.

reserved0

Location is not used in this release of SDI.

reserved1

NWSM_DEVICE_STATUS

```
typedef struct
{
   UINT32   numberOfSiblings;
   UINT32   reserved0;
   UINT32   reserved1;
   UINT32   reserved2;
   NWSMSD_OBJECT_STATUS status;
} NWSMSD_DEVICE_STATUS;
```

numberOfSiblings returns the number of sibling devices the device has. This value is returned for each sibling.

status returns the current state of the device. See NWSMSD_DEVICE_STATUS for more information.

NWSM_OBJECT_STATUS

```
typedef struct
{
   UINT32   objectStatus;
   UINT32   objectOperation;
   UINT32   objectMode;
   UINT32   reserved;
} NWSMSD_OBJECT_STATUS;
```

objectStatus returns the object's status as reported by Media Manager:

```
OBJECT_ACTIVATED           0x00000001
OBJECT_CREATED             0x00000002

OBJECT_RESERVED            0x00000010
OBJECT_BEING_IDENTIFIED    0x00000020
OBJECT_FAILURE             0x00000080

OBJECT_REMOVABLE           0x00000100
OBJECT_READ_ONLY           0x00000200

OBJECT_IN_DEVICE           0x00010000
OBJECT_LOADABLE            0x00080000

OBJECT_BEING_LOADED        0x00080000
OBJECT_DEVICE_LOCK         0x01000000
OBJECT_REMIRRORING         0x04000000
OBJECT_SELECTED            0x08000000
```

objectOperation returns the object's current operating mode. This indicator, in combination with *objectStatus*, gives the complete state of the object. The modes are:

```
NWSMSD_OPERATION_NONE          0x00000000
NWSMSD_OPERATION_WRITING       0x00000001
NWSMSD_OPERATION_READING       0x00000002
NWSMSD_OPERATION_FORMATTING    0x00000003
```

objectMode returns a bit map of the object's current access mode. The modes are:

```
NWSMSD_NOT_SUBJUGATED
NWSMSD_READ_MODE
NWSMSD_WRITE_MODE
NWSMSD_SHARE_READ_MODE
NWSMSD_SHARE_WRITE_MODE
```

reserved is scheduled for future use

NWSMSD_SESSION_ID

```
typedef struct
{
   UINT32   sessionDateAndTime;
   char     sessionDescription[NWSM_MAX_DESCRIPTION_LEN];
   char     sourceName[NWSM_MAX_TARGET_SRVC_NAME_LEN];
   char     sourceType[NWSM_MAX_TARGET_SRVC_TYPE_LEN];
   char     sourceVersion[NWSM_MAX_TARGET_SRVC_VER_LEN];
} NWSMSD_SESSION_ID;
```

sessionDataAndTime passes or returns the session's date and time. To pack/unpack this data see "DOS Date and Time Functions" in the utilities document.

sessionDescription passes a user-supplied session identification string.

sourceName passes the target's name. This data is formatted according to the SIDF specifications.

sourceType passes or returns the target type. This data is formatted according to the SIDF specifications.

sourceVersion passes or returns the target's version. This data is formatted according to the SIDF specifications.

NWSMSD_DEVICE_ID

```
typedef struct
{
   UINT32                  uniqueDeviceID;
   UINT32                  siblingUniqueID;
   UINT32                  deviceType;
   UINT32                  deviceRelation;
   char                    deviceName[NWSM_MAX_DEVICE_LABEL_LEN];
   NWSMSD_DEVICE_STATUS    deviceStatus;
   UINT32                  reservedStatus;
   NWBOOLEAN               sequential;
   NWBOOLEAN               removable;
   CAPACITY                deviceCapacity;
   UINT32                  unitSize;
   UINT32                  reserved0;
   UINT32                  reserved1;
   UINT32                  reserved2;
   UINT32                  reserved3;
   UINT32                  reserved4;
} NWSMSD_DEVICE_ID;
```

uniqueDeviceID passes or returns the device's ID.

siblingUniqueID returns the next device's ID. If *siblingUniqueID* is 0, there is only one device. If it returns the first device ID received, all the devices have been identified.

deviceType returns the device's type as defined by the OS (e.g., 4 mm, or 8mm)

deviceRelation returns a bit map that shows the device's ability to handle media (only one relationship type is supported under SMS for NetWare v4.0):

```
NWSMSD_DEVICE_SINGLE_MEDIA    0x00000000
   A device qualifies as a single media device if
   every media must be manually inserted.
```

deviceName returns the device's name as reported by the device driver via Media Manager.

Note: The name can be set by **NWSMSDLabelDevice**.

deviceStatus returns the device's status. See NWSMSD_DEVICE_STATUS for more information.

reservedStatus returns the device's assignment. The reserved statuses are:

```
NWSMSD_RESERVED_TO_THIS_SDI      0x00000001
NWSMSD_RESERVED_TO_OTHER_APP     0x00000002
NWSMSD_UNRESERVED                0x00000003
```

sequential is TRUE if the device is sequentially accessed, and FALSE if it is randomly accessed.

removable is TRUE if the media is removable and FALSE if the media is fixed.

deviceCapacity returns the usual capacity of the device. This value may vary depending on the media used, and should only be used as a guideline.

unitSize returns the device's basic unit size; usually, this is the sector size.

CAPACITY

```
typedef struct
{
   UINT32 factor;
   UINT32 value;
} CAPACITY;
```

factor shows *value*'s unit measurement. The following units are defined:

NWSMSD_CAPACITY_BYTE
NWSMSD_CAPACITY_KILO
NWSMSD_CAPACITY_MEGA
NWSMSD_CAPACITY_GIGA
NWSMSD_CAPACITY_TERA

value shows the number of units on an object (i.e., the total capacity of the object). For example, if the fields were set to the following values:

value = 24
factor = NWSMSD_CAPACITY_GIGA

The total capacity of the object is 24 gigabytes.

Note: SDI always reports the lowest factor possible to provide the maximum size resolution.

NWSMSD_MEDIA_ID

```
List Media uses only uniqueID
   typedef struct
   {
      UINT32                  uniqueMediaID;
      UINT32                  mediaSetDateAndTime;
      UINT32                  mediaDateAndTime;
      UINT32                  mediaNumber;
      BUFFER                  mediaLabel[NWSM_MAX_MEDIA_LABEL_LEN];
      NWSMSD_MEDIA_STATUS     mediaStatus;
      UINT32                  reserved0[3];
      NWSMSD_MEDIA_LOCATION   mediaLocation;
      UINT32                  mediaOwner;
      UINT32                  reservedStatus;
      UINT32                  mediaType;
      NWBOOLEAN               sequential;
      NWBOOLEAN               removable;
      UINT32                  unitSize;
      UINT32                  reserved1[3];
      CAPACITY                totalCapacity;
      CAPACITY                reserved2;
} NWSMSD_MEDIA_ID;
```

uinqueMediaID passes or returns the media's ID. This is returned by **NWSMSDListMedia**.

mediaSetDateAndTime passes or returns the creation date and time of the media set (this is set by SDI). The value is the same for every medium in the set, and uses the NetWare's calendar format (the number of seconds since 1/1/70).

mediaDateAndTime returns the medium's creation date and time. The first medium's date and time is the same as *mediaSetDateAndTime*. The following media contains the date and time when it was created.

mediaNumber passes or returns the medium's sequence number within the media set (the value starts from 1). If *mediaSetDateAndTime* is used as input and is set to 0, the first medium in the media set is returned. If it is set to NWSMSD_END_MEDIA, SDI locates the last known medium in the media set. SDI has no knowledge of how many media are in the media set or if the last known media for the media set is really the last one.

mediaLabel returns the media's label; this can be a null string. Each medium in a media set has the same label, but each medium is made unique through a media (sequence) number. During read operations, if SDI encounters end-of-media, it will search for another medium with the same label and a media number that is one higher than the current number. During write operations, if SDI encounters end of media, it will mount an empty media, give the media the

same label, and give it a media number that is one higher than the previous media.

mediaStatus returns the media's status. See NWSMSD_MEDIA_STATUS for more information.

mediaLocation passes or returns the media's location. See NWSMSD_MEDIA_LOCATION for more information.

mediaOwner passes or returns an owner value. There are two media owner groups: Novell and third-party developers. The Novell-defined owners are:

```
UNIDENTIFIABLE_MEDIA        0x00000001
HIGH_SIERRA_CDROM_MEDIA     0x00000002
ISO_CDROM_MEDIA             0x00000003
MAC_CDROM_MEDIA             0x00000004
NETWARE_FILE_SYSTEM_MEDIA   0x00000005
INTERNAL_IDENTIFY_TYPE      0x00000007
SMS_MEDIA_TYPE              0x00000008
```

The third-party-defined owner IDs have the high nibble of the high byte set to 0xF. If the owner is not known or the media is unlabeled, *mediaOwner* is set to 0 and *mediaLabel* returns a null string (first character is '\0'). If the completion code is not 0 and not SMS_MEDIA_TYPE, the media is labeled by a non-SMS engine.

reservedStatus returns the media's assignment. The following reserved status are defined:

```
NWSMSD_RESERVED_TO_THIS_SDI   0x00000001
NWSMSD_RESERVED_TO_OTHER_APP  0x00000002
NWSMSD_UNRESERVED             0x00000003
```

Note: "Other App" refers to non-engine.

mediaType returns the media's type. The following types are defined:

```
NULL_DEVICE             0x00000001
TAPE_4MM                0x00000002
TAPE_8MM                0x00000003
DISK_PARTITION          0x00000004
WORM                    0x00000005
NWSMSD_MEDIA_TYPE_LAST  0x00000006
```

sequential returns TRUE if the media is accessed sequentially, and FALSE if the media is accessed randomly.

removable returns TRUE if the media is removable, and FALSE if media is fixed.

unitSize returns the medium's basic logical sector size in bytes (this value cannot be set by the engine). See NWSMSD_DEVICE_ID for more information.

totalCapacity returns the media's total capacity in sectors. If this value is unknown, NWSMSD_UNKNOWN is returned.

NWSMSD_HEADER_BUFFER

```
typedef struct
{
   UINT32      bufferSize;
   UINT32      headerSize;
   NWBOOLEAN   reallocateOk;
   UINT32      overflowSize;
   BUFFER      headerBuffer[1];
} NWSMSD_HEADER_BUFFER;
```

bufferSize indicates the buffer's size as allocated by the engine.

headerSize indicates this buffer' header size.

reallocateOk. Set this to TRUE to allow SDI to realloc memory if the buffer cannot hold the header.

overflowSize is the number of header bytes that could not fit into the buffer.

headerBuffer returns the header buffer.

NWSMSD_TRANS_BUF_POSITION

```
typedef struct
{
   UINT16   mediaNumber;
   UINT16   partitionNumber;
   UINT32   sectorAddress;
} NWSMSD_TRANS_BUF_POSITION;
```

mediaNumber indicates which medium in the media set the transfer buffer resides in.

partitionNumber is the partition number the transfer buffer resides in.

sectorAddress is the absolute sector address of the transfer buffer.

NWSMSD_MEDIA_POSITION

```
typedef struct
{
   UINT32                  partitionNumber;
   union
   {
      int                  relative;
      UINT32               absolute;
   } sectorAddress;

   union
   {
      int                  sessionRelative;
      UINT32               sessionAbsolute;
      UINT32               mediaIndex;
   } number;

   NWSMSD_SESSION_ID       sessionDesc;
   NWSMSD_SESSION_HANDLE   sessionHandle;
   UINT32                  mediaNumber;
} NWSMSD_MEDIA_POSITION;
```

partitionNumber returns the partition numbers.

> **Note:** For position command and position inquire, this field contains the absolute partition number.

sectorAddress.relative contains an offset relative to the current sector. The value is in sectors and can be positive or negative.

sectorAddress.absolute returns or passes the session physical sector address (for more information about this type of addressing, see *System Independent Data Format*).

number.sessionRelative returns or passes the number of sessions relative to the current position.

number.sessionAbsolute is the absolute session number from the beginning of the partition.

number.mediaIndex finds the next media index if the portioning mode is NWSMSD_MEDIA_INDEX and *mediaIndex* is set to:

0 SDI positions the head to the media index at the end of the current session. If it does not exist, SDI fails.

1 SDI finds the next media index unconditionally.

sessionDesc. If a reposition command is issued (not a position inquiry), this field contains the session description to search for. If a position inquiry is issued, this field returns the current session's description. This field is ignored if *sessionHandle* is used.

sessionHandle is used if the position mode is NWSMSD_POSITION_SECTOR_ABS. This field passes a session handle that was set by **NWSMSDOpenSessionForReading** and **NWSMSDOpenSessionForWriting**. *sessionDesc* is ignored if this field is used.

mediaNumber passes the desired media number. If the media is not mounted, SDI will mount it.

NWSMSD_CONTROL_BLOCK

```
typedef struct
{
   UINT32                     transferBufferState;
   NWSMSD_SESSION_HANDLE      sessionHandle;
   UINT32                     transferBufferSequence;
   NWBOOLEAN                  finalTransferBuffer;
   UINT16                     reservedVariable;
   BUFFERPTR                  transferBuffer;
   UINT32                     transferBufferSizeAllocated;
   UINT32                     transferBufferSizeData;
   UINT32                     sessionDataType;
   UINT32                     transferBufferDataOffset;
   UINT32                     bytesNotTransfered;
   UINT32                     bytesSpanned;
   NWSMSD_TRANS_BUF_POSITION  beginningPosition;
   NWSMSD_TRANS_BUF_POSITION  endingPosition;
   UINT32                     completionStatus;
} NWSMSD_CONTROL_BLOCK;
```

transferBufferState is set and used by the engine to track the transfer buffer(s). SDI updates this value before returning it to the engine. The statuses are:

NWSMSD_UNASSIGNED (0x00000000)
The transfer buffer is allocated and not in use. The engine sets this.

NWSMSD_AVAILABLE (0x00000001)
The transfer buffer was allocated and available for the engine to use.

NWSMSD_READY_TO_TRANSFER (0x00000002)
The transfer buffer is ready for SDI to use. This is set by the engine. Set the field to this value when calling **NWSMSDReadSessionData** or **NWSMSDWriteSessionData**.

NWSMSD_TRANSFER_IN_PROGRESS (0x00000003)
SDI is transferring data to the transfer buffer.

NWSMSD_TRANSFER_COMPLETE (0x00000004)
SDI has completed the data transfer.

NWSMSD_TRANSFER_STATUS_LAST (0x00000005)
This value marks the last status and is not used for anything else.

sessionHandle passes the session handle. The engine sets this field before calling SDI.

transferBufferSequence specifies the sequence in which the transfer buffers are read or written. This field is initially set

by the engine (e.g., SME) for either writes or reads. Before calling **NWSMSDWriteSessionData**, the engine initially sets this field to 1. After making the first call, the engine increments it before making the next call.

For a read session (calling **NWSMSDReadSessionData**), the initial value of this field can be one of two values:

- 0xFFFFFFFF, the function returns the sequence number of the transfer buffer read. Reset this field to 0xFFFFFFFF each time **NWSMSDReadSessionData** is called.

- Any value except 0xFFFFFFFF. SDI retrieves the specified transfer buffer. The actual sequence found is returned with the transfer buffer (i.e.., if the specified transfer buffer is not found, the transfer buffer with the next higher sequence number is returned).

finalTransferBuffer is TRUE if the last transfer buffer was read. For a write session, the engine must set this to TRUE if it is the last transfer buffer.

reservedVariable is used internally by SDI.

transferBuffer points to the transfer buffer. The engine allocates memory for the transfer buffer and is responsible for releasing it. **NWSMSDOpenSessionForReading** and **NWSMSDOpenSessionForWriting** determines the size of the transfer buffer.

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

transferBufferSizeAllocated passes the transfer buffer's size. This is used to determine if the transfer buffer is large enough to hold the requested data. The engine sets this field before calling SDI.

transferBufferSizeData is the transfer buffer header size and data size. This value is set by the engine. For write operations, the field specifies the amount of data to be written. For read operations, it shows the maximum amount of data or header and data that can be read into the transferBuffer. If the field is set to 0, *transferBufferSizeAllocated* is used. SDI updates this field to specify the number of bytes actually read.

sessionDataType specifies the type of data in the transfer buffer as shown below:

NWSMSD_TSA_DATA (0x00000001): The transfer buffer contains the data from the TSA.

NWSMSD_END_OF_TSA_DATA (0x00000002)

NWSMSD_SESSION_TRAILER (0x00000003)

NWSMSD_SESSION_INDEX (0x00000004)

NWSMSD_MEDIA_INDEX (0x00000005)

NWSMSD_END_OF_SESSION (0x00000006)
The transfer buffer is empty.

Session Index
Once an engine starts writing the session index, the media cannot be accessed by the current engine or any engine except to complete the session index (i.e., no other data except the session index data may be placed between the session trailer and the session index).

Session Trailer
For a back up session, the contents of the transfer buffer are written as the session trailer. Once an engine starts writing the session trailer, the media is not accessible to the current engine or any engine except to complete the session trailer or to write the session index. **NWSMSDCloseSession** must still be called to indicate the end of data transfers to this session (nothing is put onto the media).

If the engine writes the session trailer onto the media, the engine must do one of the following:

- Immediately write a session index and call **NWSMSDCloseSession**
- Call **NWSMSDCloseSession** without witting a session index

Witting a session index causes SDI to block *all* operations to the media until this engine calls **NWSMSDCloseSession**. This engine may continue to write trailers, indexes, etc., until **NWSMSDCloseSession**

is called. However, SDI enforces the order of the trailers, indexes, etc., as follows:

1. File mark
2. Session trailer
3. Session index
4. Media index
5. Set mark

No data is allowed between any section. If the engine writes anything in the wrong order, NWSMSD_NONSMS_COMPLIANT is returned.

transferBufferDataOffset is the offset from the beginning of the transfer buffer to the start of data.

bytesNotTransfered returns the number of bytes not transferred for the specified I/O operation.

beginningPosition returns the transfer buffer's beginning position. Transfer buffers are written on sector boundaries. All NWSMSD_MEDIA_POSITION fields contain session physical sector addresses.

endingPosition returns the transfer buffer's ending position. Since the transfer buffer's size is a multiple of the sector's size, the end lies on a sector boundary.

completionStatus returns the pending and completion status of nonwaiting requests.

NWSMSD_DEVICE_LIST

```
typedef struct
{
   UINT32               deviceTotalCount;
   UINT32               deviceMaxCount;
   UINT32               deviceResponseCount;
   UINT32               uniqueDeviceID;
   NWSMSD_DEVICE_ID     deviceID[];
} NWSMSD_DEVICE_LIST;
```

deviceTotalCount returns the total number of available devices. If the engine calls **NWSMSDListDevices** repetitively, the engine must be aware that SDI updates this field if a device is added or removed (NWSMSD_DEVICE_LIST_CHANGED is returned if the field is updated). We recommend that the engine monitor this value, while the list is being built, to ensure the list is up to date.

deviceMaxCount contains the maximum number of elements *deviceID* can have.

deviceResponseCount returns the number of device IDs that SDI put into *deviceID*. All device IDs are returned if the value is smaller than *deviceMaxCount*. However, if this value is equal to *deviceMaxCount*, the engine must call **NWSMSDListDevices** again.

uniqueDeviceID contains the next device ID to be returned.

deviceID is an array of *deviceMaxCount* elements. The engine must allocate these elements. That is, if *deviceMaxCount* is 3, *deviceID* must be set to *deviceID*[3], when memory is allocated for it. See NWSMSD_DEVICE_ID for more information.

NWSMSD_MEDIA_LIST

```
typedef struct
{
   UINT32               mediaTotalCount;
   UINT32               mediaMaxCount;
   UINT32               mediaResponseCount;
   UINT32               uniqueMediaID;
   NWSMSD_MEDIA_ID      mediaID[];
} NWSMSD_MEDIA_LIST;
```

mediaTotalCount returns the total number of available media. If the engine calls **NWSMSDListMedia** more than once to retrieve the list of media, the engine should be aware that SDI will update this field if media is added or removed. We recommend that the engine monitor this value while the list is being built to ensure the list is up to date.

mediaMaxCount passes the maximum number of elements *mediaID* can hold.

mediaResponseCount returns the number of IDs SDI put into *mediaID*. All media IDs are returned if this value is less than *mediaMaxCoun*t. However, if this value is equal to *mediaMaxCount*, the engine must call **NWSMSDListMedia** again.

uniqueMediaID contains the next media ID to be returned.

mediaID is an array of *mediaMaxCount* elements. The engine must allocate these elements. That is, if *mediaMaxCount* is 3, *mediaID* must be set to *mediaID*[3], when memory is allocated for it. See NWSMSD_MEDIA_ID for more information.

NWSMSD_MEDIA_STATUS

```
typedef struct
{
   UINT32                  mediaMounted;
   NWSMSD_OBJECT_STATUS    status;
} NWSMSD_MEDIA_STATUS;
```

mediaMounted returns the media's mounted status.

NWSMSD_MEDIA_IS_MOUNTED (0x00000001)
Media is mounted by an engine.

NWSMSD_MEDIA_IS_DISMOUNTED (0x00000002)
Media is not mounted by any engine.

NWSMSD_MEDIA_MOUNT_PENDING (0x00000003)
Media is not mounted by any engine but a mount request is pending.

status contains the media's status. See NWSMSD_OBJECT_STATUS for more information.

NWSMSD_TRANSFER_BUF_INFO

```
typedef struct
{
   UINT32   sectorSize;
   UINT32   maxTransferBufferSize;
   UINT16   applicationAreaSize;
   UINT16   applicationAreaOffset;
   UINT16   transferBufferDataOffset;
} NWSMSD_TRANSFER_BUF_INFO;
```

sectorSize returns the smallest writable unit the engine can put onto the media. The engine should allocate a transfer buffer that is a multiple of *sectorSize*.

applicationAreaSize passes the number of bytes required within the transfer buffer. This area contains the engine's session header information, if any.

applicationAreaOffset returns the offset to the engine area. This offset is from the beginning of the transfer buffer. This area contains data formatted according to the SIDF's specifications.

transferBufferDataOffset returns the offset from the beginning of the transfer buffer where the engine may begin writing the records (i.e., the data set information and data set data).

NWSMSD_TIMEOUTS

```
typedef struct
{
   UINT32   NWSMSDListMedia;
   UINT32   NWSMSDSubjugateDevice;
   UINT32   NWSMSDSubjugateMedia;
   UINT32   NWSMSDDismountMedia;
   UINT32   NWSMSDOpenSessionForWriting;
   UINT32   NWSMSDOpenSessionForReading;
   UINT32   NWSMSDCloseSession;
   UINT32   NWSMSDWriteSessionData;
   UINT32   NWSMSDReadSessionData;
   UINT32   NWSMSDLabelMedia;
   UINT32   NWSMSDDeleteMedia;
   UINT32   NWSMSDPositionMedia;
   UINT32   NWSMSDMoveMedia;
   UINT32   NWSDMSFormatMedia;
} NWSMSD_TIMEOUTS;
```

The time units for each field are in seconds.

NWSMSD_SDI_DEFAULTS

```
typedef struct
{
      NWSMSD_DEVICE_ID deviceDesc;
      NWSMSD_MEDIA_ID mediaDesc;
      NWSMSD_TIMEOUTS timeouts;
} NWSMSD_SDI_DEFAULTS;
```

deviceDesc passes or returns the default device's description.

mediaDesc passes or returns the default media's description.

timeouts passes or returns the timeout values for each nonwaiting call. SDI uses these values to determine how long it should wait before returning an NWSMSD_TIMEOUT error.