Novell
Developer Kit

SMS DEVELOPER COMPONENTS

Novell.

Novell Trademarks

For Novell trademarks, see [the Novell Trademark and Service Mark list (http://www.novell.com/company/legal/trademarks/tmlist.html)](http://www.novell.com/company/legal/trademarks/tmlist.html).

Third-Party Materials

All third-party trademarks are the property of their respective owners.

Contents

About This Guide

Novell® Backup infrastructure (Storage Management Services or SMS) provides backup applications with the framework to develop complete backup and restore solutions. SMS helps to back up file systems (such as NSS) or applications (such as GroupWise®) on NetWare® and OES Linux.

This guide contains information regarding SMS interfaces and their interrelationships that applications writing to SMS can use to protect data on NetWare or OES Linux.

This guide consists of the following sections:

Audience

The target audience for this guide are backup application developers and corporations that already have experience with SMS in building applications for NetWare and OES Linux. This guide helps to understand existing and newer features and functionality that SMS provides.

Feedback

We want to hear your comments and suggestions about this manual and the other documentation included with this product. Please use the User Comments feature at the bottom of each page of the online documentation, or go to www.novell.com/documentation/feedback.html and enter your comments there.

Additional Documentation

For documentation on SMS recommendations, see the SMS and Backup Application Development - Recommendation Guide (http://developer.novell.com/ndk/doc/smscomp/pdfdoc/sms_reco.pdf).

Documentation Conventions

In Novell documentation, a greater-than symbol (>) is used to separate actions within a step and items in a cross-reference path.

A trademark symbol (®, ™, etc.) denotes a Novell trademark. An asterisk (*) denotes a third-party trademark.

When a single pathname can be written with a backslash for some platforms or a forward slash for other platforms, the pathname is presented with a backslash. Users of platforms that require a forward slash, such as Linux or UNIX, should use forward slashes as required by your software.

Target Services Tasks

1

This documentation describes common tasks associated with Target Services.

1.1 Backing Up and Restoring

Backing up and restoring follow the same general steps as outlined below. The Defining Data Sets and Backing Up and Restoring Data sections contain slightly different steps, depending upon whether you want to backup or restore data.

1.1.1 Connecting to a TSA

1 Call NWSMListSMDRs to select an SMDR. Then call NWSMListTSAs using the returned SMDR name to find the available TSAs.

   The fastest method to find the TSAs is to list all active SMDRs and then list the TSAs that are local to that SMDR.

   You can also call NWSMListTSAs by itself to find all the active TSAs on the network. This method can take over an hour to complete because each site with an SMDR must be connected to and scanned to see if it has a TSA.

2 Call NWSMConnectToTSA to connect to a specific TSA.

   The actual connection to the TSA is made while connecting to Target Services.

1.1.2 Connecting to Target Services

After connecting to a TSA, you are now ready to list, select, and connect to the Target Service.

1 Call NWSMTSListTargetServices to find the Target Service.

   You can call NWSMTSScanTargetServiceName for this step. However, it will return only one Target Service name at a time.

2 Call NWSMTSConnectToTargetService or NWSMTSConnectToTargetServiceEx to connect to the Target Service and gain access to the Target Service data.

1.1.3 Defining Data Sets

After connecting to the Target Service, either a backup or restore operation can be performed. The following sections details how to define data sets for backup or restore,

Defining Backup Data Sets

The user or engine uses backup option lists to indicate how to proceed with the backup session and what data sets to back up.

1 Call NWSMTSGetTargetScanTypeString to build the Scan Type Options List.

2 Display the strings to the user to gather information about how to limit the scan.

When the user selects a scan type option, the engine checks the associated disallowed bit map to see if the selected scan type can be used. If it can be used, the engine uses the associated scan type bit mask and allowed bit mask to set the bit into a scan type variable.These values are then set in the NWSM_SCAN_CONTROL structure by calling NWSMTSScanDataSetBegin to define the scanning criteria.

3 Call NWSMTSOpenDataSetForBackup to define all backup open mode options and NWSMTSReadDataSet to define all reading mode options.

4 Call NWSMTSGetOpenModeOptionString to retrieve the TSA-specific open mode strings.

5 Display the string to the user for selection.

The associated values for these selections are placed into a 32-bit variable. You must ensure that only one numeric open mode is selected. For more information on open mode, refer Section 2.2.1, "Open Mode Options," on page 24.

6 Call NWSMTSGetTargetSelectionTypeStr to retrieve the selection option strings.

7 Either let the engine set the bits for the chosen selection type or call NWSMTSGetTargetSelectionTypeStr to use an SMS-defined mask to set the selection types.

8 Display the selection type string list to the user for selection.

Including Directories

If the user wants to include a certain directory in a backup, the engine allows the user to walk through the file system tree and builds the path as each resource (such as a primary resource, directory, and file) is selected as follows:

1 Call NWSMTSListTSResources to return a list of primary resources to the user for selection.

NWSMTSScanTargetServiceResource can be called to return one primary resource name at a time.

2 Call NWSMTSScanDataSetBegin, NWSMTSScanNextDataSet, and NWSMTSScanDataSetEnd to display a list of all immediate secondary resources under the selected primary resource to the user for selection.

3 If the secondary resource is a parent, scan for all immediate secondary resources under the selected secondary resource.

4 Append each resource name and the appropriate separator to a path string.

5 Repeat Steps 1-4 until the user find the desired data set.

Defining Restore Data Sets

After establishing connections to the TSA and Target Services, you need to build the restore option lists.

1 Call NWSMTSGetOpenModeOptionString to retrieve the TSA-specific open mode option strings.

When the engine is ready to restore the data set, it informs the TSA of the chosen open modes to use by sending the 32-bit variable to the TSA by calling NWSMTSOpenDataSetForRestore.

2 If the restore data sets contain paths, call the NWSMTSFixDataSetName to ensure the paths have the proper syntax and form.

1.1.4 Backing Up and Restoring Data

Backing Up Data

Before backing up data, make sure the backup options and the data sets names to be backed up are selected.The following flowchart summarizes the steps outlined in this section:

Figure 1-1 *Backing up data*

```
                        ┌─────────┐
                        │  Start  │
                        └────┬────┘
                             ▼
              ┌──────────────────────────────┐
              │   NWSMTSScanDataSetBegin      │
              └───────────────┬──────────────┘
                              ▼
                          ╱Return╲         No
                         ╱ Zero?  ╲────────────────────────────┐
                         ╲        ╱                            ▼
                          ╲      ╱                        ╱No more ╲
                            │Yes              No         ╱data sets? ╲  Yes
                            ▼            ┌──────────────╲           ╱────────┐
           ┌───────────────────────────┐│              ╲         ╱          ▼
           │ NWSMTSOpenDataSetForBackup │◄─(2)          ▼                ┌──────┐
           └─────────────┬──────────────┘             (1)               │ Done │
                         ▼                                              └───┬──┘
                     ╱Return╲       No                                     ▼
                    ╱ Zero?  ╲──────────►(1)                              (  )
                    ╲        ╱
                     ╲      ╱
                       │Yes
                       ▼
           ┌───────────────────────────┐
           │     NWSMTSReadDataSet      │
           └─────────────┬──────────────┘
                         ▼
                     ╱Return╲       No
                    ╱ Zero?  ╲──────────►(1)
                    ╲        ╱
                     ╲      ╱
                       │Yes
                       ▼
             No    ╱Read all╲
        ┌─────────╲  Data?  ╱
        │          ╲       ╱
        │            │Yes
        │            ▼
        │  ┌───────────────────────────┐
        │  │    NWSMTSCloseDataSet      │
        │  └─────────────┬──────────────┘
        │                ▼
        │  ┌───────────────────────────┐
        └─►│   NWMSTSScanNextDataSet    │
           └───────────────────────────┘
```

Figure 1-2 *Backing up data*
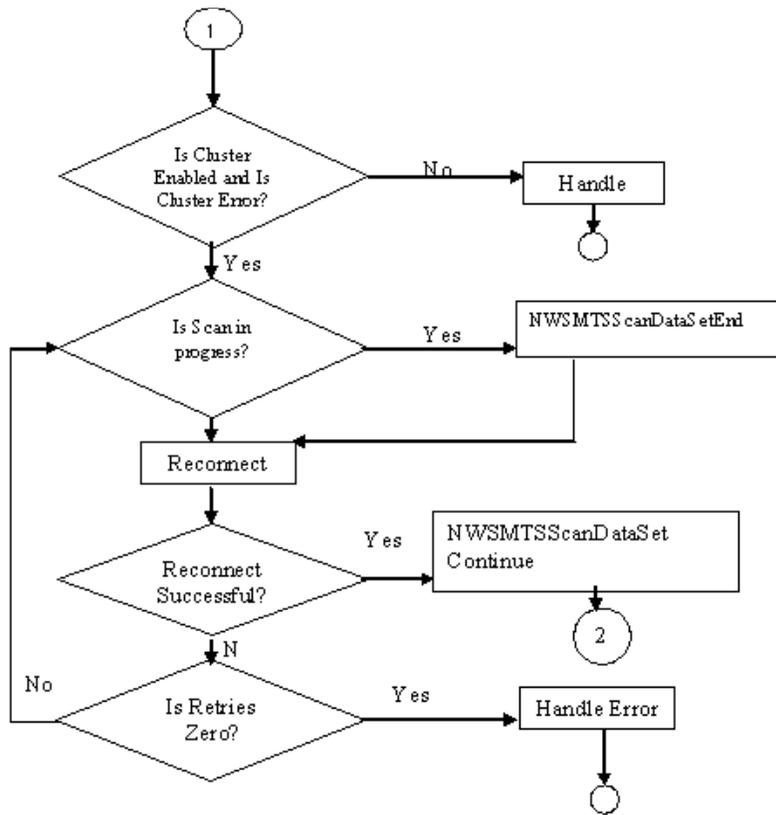
1 Call NWSMTSBuildResourceList to initialize the TSA.

The TSA will only scan the components of the internal primary resource list so the list must be updated in this step.

2 Call NWSMTSScanDataSetBegin to begin the scan.

If a failover or failback occurs, call NWSMTSScanDataSetEnd, reconnect to the TSA and call NWSMTSScanDataSetBegin.

3 Call NWSMTSOpenDataSetForBackup to open a data set.

If a failover or failback occurs, call NWSMTSScanDataSetEnd, reconnect to the TSA and call NWSMTSScanDataSetContinue only if the last successfully backed-up data set exists. If NWSMTSOpenDataSetForBackup fails in the first data set, call NWSMTSScanDataSetBegin.

4 Call NWSMTSReadDataSet to retrieve the first data set to backup.

If you need to retrieve a name from the data set name list, call the following functions:

NWSMGetDataSetName
NWSMGetFirstName
NWSMGetNextName
NWSMGetOneName

If a failover or failback occurs, reconnect to the TSA and call NWSMTSScanDataSetContinue only if the last successfully backed-up data set exists. If NWSMTSReadDataSet fails in the first data set, call NWSMTSScanDataSetBegin.

5 Call NWSMSDSessionWriteData to allocate memory for the transfer buffer.

6 Call NWSMSetNewRecordHeader and NWSMUpdateRecordHeader (see the example code for NWSMTSScanDataSetBegin) to prepare the data set (according to SIDF) for storage on the media.

These functions fill the transfer buffer with the scan information and data set information and automatically take care of data sets that span transfer buffers.

7 Call NWSMSDSessionWriteData to write the data set to the media.

8 Call NWSMTSSetArchiveStatus to reset the archive status of the data set.

9 Call NWSMTSCloseDataSet to close the data set.

NWSMTSCloseDataSet will restore the original access attributes of the data set.

10 Call NWSMTSScanNextDataSet to get the next data set to backup and continue with Step 3.

If a failover or failback occurs, reconnect to the TSA and call NWSMTSScanDataSetContinue only if the last successfully backed-up data set exists. If NWSMTSScanNextDataSet fails in the first data set, call NWSMTSScanDataSetBegin.

To prematurely terminate the scan, call NWSMTSScanDataSetEnd. If an error was returned from NWSMTSScanDataSetBegin or NWSMTSScanNextDataSet, NWSMTSScanDataSetEnd should not be called.

Restoring Data

The following flowchart summarizes the steps outlined in this section:

Figure 1-3 *Restoring Data*
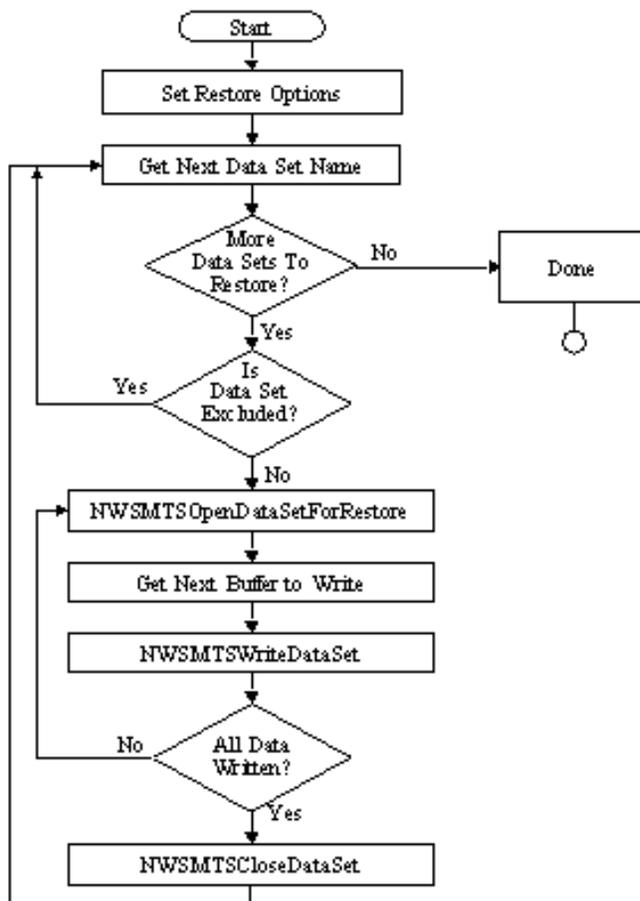
1 Call NWSMTSSetRestoreOptions to set the restore options.

2 Retrieve a data set name from the backup session that needs to be restored.

If a name was successfully retrieved, continue with Step 3. Otherwise, follow the steps outlined in Section 1.1.5, "Terminating a Process," on page 20.

3 Call NWSMTSIsDataSetExcluded to check if the data set has been excluded from the restore session.

If the data set is excluded, go to Step 2. Otherwise, continue with Step 4.

4 Call NWSMTSOpenDataSetForRestore to open the data set.

5 Call NWSMGetRecordHeaderOnly and NWSMGetDataSetInfo to retrieve the data set information from the SIDF medium transfer buffers.

You can also call the following functions to retrieve the data set's name from the NWSM_DATA_SET_NAME_LIST structure:

NWSMGetDataSetName (page 201)
NWSMGetFirstName (page 202)
NWSMGetNextName (page 204)
NWSMGetOneName (page 206)

6 Call NWSMTSWriteDataSet to write the data set to Target Services.

7 Call NWSMTSCloseDataSet to close the data set.

If no error occurred, loop to Step 2. Otherwise, handle the error.

This restore method consists of three loops. The first loop gets the next data set name and checks if it is excluded from the restore session. The second loop gets the buffers (containing the data set) from the medium and writes it to the Target Service. The outer loop goes back to the top and gets the next data set name.

1.1.5 Terminating a Process

When a backup or restore task completes, release the connections and free the allocated memory by following these steps:

1 Call NWSMTSReleaseTargetService to disconnect Target Service from the TSA.

2 Call NWSMReleaseTSA to disconnect the engine from the TSA.

3 Call the following functions (as needed) to free allocated memory.

NWSMDestroyList frees the memory allocated to the NWSM_LIST_PTR (page 321) structure.

NWSMFreeString frees the memory allocated by NWSMTSCatDataSetName, SMSTSFixDataSetName, NWSMTSGetNameSpaceTypeInfo, and NWSMTSSeparateDataSetName for the STRING_BUFFER (page 178) structure.

NWSMFreeList frees the memory allocated for the NWSM_NAME_LIST structure by calling the following functions:

NWSMListSMDRs
NWSMListTSAs
NWSMTSListTargetServices
NWSMTSListTSResources
NWSMTSListSupportedNameSpaces

free is a C-language function that frees the memory allocated by NWSMTSParseDataSet for the UINT16_BUFFER structure.

1.2 Building Log Files

Follow these steps to build a log file:

1 Call NWSMGetFirstName, NWSMGetNextName, NWSMGetDataSetName, and NWSMGetOneName to get a name from the skipped data sets list.

2 Set createSkippedDataSetsFile of NWSM_SCAN_CONTROL to TRUE to direct the TSA to build the skipped data sets log file.

1.2.1 Reading Log Files

To read built log files, follow these steps:

1 Call NWSMTSScanDataSetBegin and pass ERROR LOG or SKIPPED DATA SETS to resourceName to receive the name of the log file contained in dataSetNames.

2 Call NWSMTSOpenDataSetForBackup to open the log file and receive a data set handle.

3 Call NWSMTSReadDataSet to read the contents of the log file.

4 Call NWSMTSCloseDataSet to close the file.

5 Call NWSMTSScanDataSetEnd to end the read file session.

> **WARNING:** Do not send the data from the error or the skipped data sets log to NWSMTSWriteDataSet because *scanInformation* and *dataSetNames* will not contain valid SIDF information and the system might abend.

1.2.2 Engine Log File

1 Call NWSMTSScanDataSetBegin and NWSMTSScanNextDataSet.

2 Build a log file or database of the data sets that were backed up from the returned information.

1.3 Scanning

Follow these steps to find data sets on Target Services that match the scan criteria:

1 Call NWSMTSScanDataSetBegin to start scanning.

2 Call NWSMTSScanNextDataSet to continue the scan

3 Call NWSMTSScanNextDataSet to continue the backup job after cluster failover/failback, if the last successfully backed-up data set exists.

Call NWSMTSScanDataSetEnd to prematurely stop scanning.

4 Call NWSMTSScanDataSetEnd when all data sets have been scanned.

1.3.1 Modifying the Scan Order

1 Call NWSMTSReturnToParent to move the scanning point.

2 Call NWSMTSScanNextDataSet to continue the scan.

3 Call NWSMTSCloseDataSet to close the data set's handle once scanning is complete.

The data set's handle should not be closed until all the parent's subordinates are scanned.

The following diagram illustrates a scan order. A scan that has begun at parent A moves to parent F. After scanning child h, the engine decides it is done with parent F, and moves the scan to the next parent by calling NWSMTSReturnToParent. When the function is called, the scan is moved from parent F to parent E. When NWSMTSScanNextDataSet is called, there are no more data sets to scan in parent E so scanning continues in parent G.

Figure 1-4 *Scan Order*
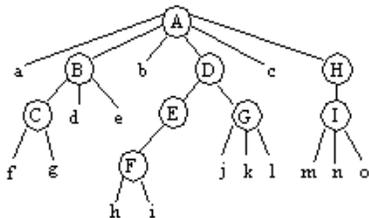
Target Services Concepts

2

This documentation describes Target Services, its functions, and features.

2.1 Options

The Target Service Agent (TSA) is a target-specific process that interacts with the Target Service's file system to read, write, and scan data. Each TSA reflects the features and limitations offered by the Target Service. If the Target Service has a hierarchical file system, the Target Service offers options that pertain to a hierarchical file system (for example, traversing the file system tree).

Targets services differ in the type of data sets they contain and the methods used to scan (search for) them. For example, one target has a flat file system while another has a hierarchical file system. When scanning for data sets on these targets, the option to exclude files under a directory does not exist under a flat file system.

Each TSA provides different options such as traversing the file system, excluding all databases, or backing up a file server. Since the engine should not know these differences, SMS provides a way for each TSA to express the options and resources available to the user. Each TSA defines a list of strings that describe its options and resources. The engine retrieves these strings, presents them to the user for selection, and notifies the TSA of the user's selections through bit maps or values.

2.1.1 Types of TSA Options

There are two categories for TSA options:

TSA-specific

TSA-specific options are unique to a specific TSA. For example, a NDS TSA might offer the option to restore only the objects of distinguished name, while a filesystem TSA might offer the option to restore all `*.exe` files.

TSA-specific options contain a subset called the predefined TSA options. Because the engine is aware of predefined options (as is every TSA), it can automatically assign these options to the data sets selected by the user for backup or restore.

For example, if the exclude directories and exclude files options are selected, the engine can display all children and parent data sets. The user can then select the desired data sets to backup or restore. The engine can then appropriately assign exlude directories or exclude files to the chosen data set since it knows the type of each data set.

The TSAs are not required to support the TSA-specific options, but each TSA must acknowledge to the engine if a predefined option is supported. The engine does not know about the TSA-specific options.

Generic TSA

Generic TSA options are the options common to most TSAs. All TSAs are not required to support these options, but each TSA must acknowledge to the engine whether each option is supported or not. The engine is aware of these options and is responsible for:

- Finding out if the options are supported by the TSA.
- Providing the strings that name the options.
- Communicating the options selected by the user to the TSA.

The TSA builds the strings for the TSA-specific options while the engine builds the strings for the generic TSA options.

2.2 Backup and Restore Options

The backup and restore options are used to specify the data sets to either backup or restore. Both processes use the Open Mode Options which are discussed in the next section. Other unique options are discussed within each relevant section entitled *Restore Options* and *Backup Options*.

2.2.1 Open Mode Options

Open mode options are defined by NWSMTSWriteDataSet and contain a mixture of restore modes and writing modes such as: update the data set, overwrite the data set, and do not restore the trustee information. These modes allow a finer control (than scan control options) in defining data sets since the open mode options are applied to individual data sets as each data set is opened, read, or written to. For example, the engine can back up all data sets or lock data sets before they are opened.

There are two basic groups of open modes:

- Numeric open modes are represented by values numbered 0-15. Because these modes are sequential, the engine can specify only one numeric open mode.
- Bitmap open modes are represented by bit positions. More than one of these modes can be ORed together.

Both modes are contained in a 32-bit integer where bits 0-3 contain the numeric open modes, and bits 4-31 contain the bitmap open modes.

Figure 2-1   *Two basic groups of Open Modes*
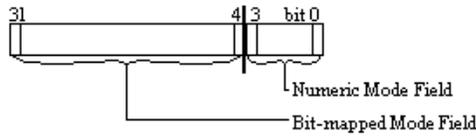
All numeric open modes and bits 4-7 of the bitmap open modes are classified as predefined TSA open modes. Each TSA must either support the options or indicate which options are not supported.

Bits 8-31 contain open modes specific to a particular TSA and are classified as TSA-specific open modes. These open modes are required by NWSMTSWriteDataSet.

If a numeric open mode is chosen and set to Yes, its corresponding value is set into the numeric mode field of the open mode bit map. The engine must ensure that only one numeric option is chosen. If a bitmap mode is chosen, its corresponding bit is set into the bit map.

Open Mode Option Lists

The open mode option lists display the TSA-specific and generic open options to the user for selection and are optional. The engine can internally set the values for the open modes without any user intervention (see Section 9.2, "Target Services Generic Open Mode Values," on page 337).

The open mode options list has four parts:

- The open mode generic strings
- The open mode TSA-specific strings
- The open mode string index number
- The open mode number

2.2.2 Restore Options

The restore options specify the data sets to restore. Options include excluding data streams, file trustees, directory space restrictions, etc.

Figure 2-2   *Restore Options*
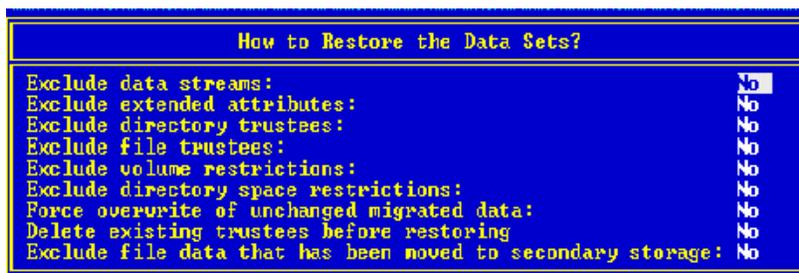
There are basically two restore option types:

Names Option Lists

The names option list is a list of data sets that were defined by the user to be restored (such as restore all *.exe and *.txt files) and their optional associated path. The data set names can contain wild cards (in the terminal path node-parent or child only), fully qualified paths, or terminal names.

After the data sets are specified, the engine puts the list into an NWSM_SELECTION_LIST structure. The engine must set the selection type for all entries in this structure to No Selection Type or zero.

Unsupported Options

Since TSAs might not support all TSA options, the engine can call NWSMTSGetUnsupportedOptions to find out which generic TSA option string it should not build and display to the user.

2.2.3 Backup Options

The backup options specify the data sets to backup.

There are three backup option lists:

The order of the strings returned by these functions corresponds to the bit position of a bit map.

The engine builds the strings for generic TSA options, and the TSA builds all strings for predefined and SMS-defined options. The engine uses the information from the NWSM_SCAN_CONTROL structure and the generic open mode options to build the strings for its part of the options list.

All of the options do not have to be used by the engine. The engine can pick and choose which options it wants to display to user.

Scan Control Options

Scan Control Options specify the attributes and characteristics of all data sets to scan for and are applied to every data set in the session. For example, you can specify to scan for all DOS data sets accessed within the last 10 days.

The scan control options are based upon the information required by the NWSM_SCAN_CONTROL structure and are returned by NWSMTSGetTargetScanTypeString.

Scan Control Options Lists

The scan control option lists contains at least two different lists. The number of lists is dependent upon how the engine presents the options to itself or the user. The engine can set the values in both of the following lists without any user intervention:

◆ Date and Time Option Lists
◆ Scan Type Option Lists

The date and time options list is optional. The engine builds the strings for the date and time fields and all other fields of the NWSM_SCAN_CONTROL structure except the scan type field. The fields that have no date values should be considered as having a zero value. The engine uses these strings as prompts to get the user's input.

The scan type options list displays the TSA-specific and predefined scan options to the user for selection and is also optional. However, unlike other lists, the engine must check for valid combinations of scan types. The TSA will also check for valid combinations, but this checking is not done until all options are specified and the backup has begun.

The scan type options list has three parts:

- The scan type strings are built by the TSA.
- The scan type string index number specifies which scan type string to retrieve.
- The scan type value which indicates the type of the scan to perform.

Selection Options

Selection options allow the selection of an option and name the data sets to backup. For example, a user can select the exclude directories option and specify the sub-directory to be excluded. These options are returned by NWSMTSGetTargetSelectionTypeStr.

Selection options are used with path information to specify the name of the data sets to apply the selection type to. Call NWSMPutFirstName and NWSMPutNextName to build the selection list.

There are basically two selection option types:

Selection Types

There are two kinds of selection types: the SMS-defined selection types and the selection types particular to a TSA.

SMS-defined TSA selection types are common to most TSAs, but each TSA does not have to support them. However, every TSA must acknowledge whether it supports a SMS-defined selection type.

Both selection types are represented in the 32-bit selection type bit map. Each selection type is represented by two bits within the bit map. Bit 0 represents a TSA-specific defined action, while bits 1-31 represent the data sets to apply the action against. For example, the TSA defines bit 0 as an include/exclude action. When this bit is not set, a data set is excluded from the session.

Figure 2-3 *Selection Type*
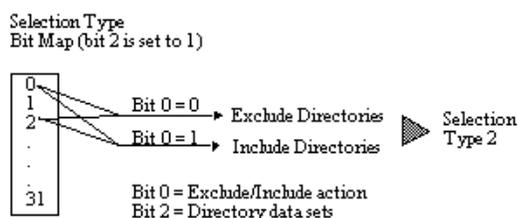
The following table lists the SMS-defined bits 1 through 4.

Table 2-1 *SMS-defined Bits*

Bit	Description
1	TSA defined resource: These resources are always TSA-specific resources.
2	Parents (e.g., Directories or folders)
3	Children (Files)
4	Children by full name (a fully qualified path)

A separate bit map must be used to represent each selection type.

Selection Options List

The selection options list displays the TSA-specific and SMS-defined selection options to the user for selection and is optional.

The selection options list has four parts:

- The selection option strings
- The selection option string index number
- The selection option bit mask
- The path

The TSA builds the selection option strings. The selection option index number is a used to specify which selection string to get from the TSA. The selection option bit mask is used to set the selection type bit map. The path is entered by the user or the engine.

There are two methods for setting the selection type:

- The engine sets the bits for the chosen selection type.

  For example, if a user selects Include Directories, the engine knows that it is the second string of the first selection type and sets bit 0 to equal 1 and bit 2 is also set.
- The engine uses the masks defined by SMS.

  NWSMTSGetTargetSelectionTypeStr is passed a sequence number (selection type number) which indexes a string table. The sequence number is converted to a bit map or mask before being used.

Selection options are defined by NWSMTSScanDataSetBegin, NWSMTSScanNextDataSet, and NWSMTSSetRestoreOptions which act as data set filters.

Wild cards are used only in the terminal path node. The terminal path node can be a child or a parent.

2.3 Name Type Option

Name Type refers to the format in which a name space is represented. Thus, information in each name space maybe represented in multiple formats as determined by the Name Type Options supported by the TSA.

These options can be obtained using NWSMTSGetsupportedNameTypes.

This option is supported, as a target service may require to support multiple name formats for every name space that it supports. For example, the TSA for the file system (TSAFS) represents resource names in MBCS as well as UTF-8 formats from Open Enterprise Server 1.0 release. Hence, the TSA uses the UTF-8 and MBCS name types.

2.4 Open Files During a Back Up

TSAs make every precaution to ensure that the engine receives no corrupt data. If the engine must backup files that are opened by other applications, a special TSA is needed to perform this task. File system TSAs generally do not back up files that are already opened.

The engine controls the mode of opening a data set by passing a mode parameter to NWSMTSOpenDataSetForBackup. The TSA provides the required open modes to ensure data integrity. Whenever a file is backed up, it should either be successfully locked and protected or be opened in a way that denies write access to other applications as the file is backed up.

SMS provides two modes that circumvent lock and write protection. These modes are NWSM_NO_LOCK_NO_PROTECTION and NWSM_OPEN_READ_ONLY. The engine should only use these modes when data integrity is not required.

2.5 Backup and Restore of Cluster Resources

Novell Cluster Services™ allows you to configure up to 32 NetWare servers into high-availability cluster, where resources can be dynamically switched or moved to any server in the cluster. Consolidation of applications and operations on a cluster has benefits such as lower costs, scalability, and increased availability.

For a cluster to work as a high-availability system, the file system, the applications, and services that run on the cluster should be cluster-enabled. SMS supports backup and restore of cluster resources. In addition, the backup session can be automatically recovered in case of a failover or failback condition.

The backup engines have to be modified to be able to support backup and restore of cluster resources. For more details, refer Section 1.1.4, "Backing Up and Restoring Data," on page 15.

2.5.1 Recovering Backup Session on Cluster Failover or Failback

SMS supports automatic recovery of backup sessions in failover and failback situations.

If the connection to the TSA is terminated after a failback or failover, the engine attempts to reconnect to the TSA through SMDR. Ensure SMDR and TSA modules are loaded in the failed-over/failed-back cluster nodes.

If a scan is in progress when the connection terminates, the engine should end the scan by calling NWSMTSScanDataSetEnd. After this, the engine should try to reconnect after waiting for a configurable time and then retry at regular intervals until the connection is re-established or the number of retries are expired. The engine should call NWSMConnectToTSA in a loop until the connection is established. After the engine establishes the connection, to continue from the last completely backed up dataset, call NWSMTSScanDataSetContinue keeping all the parameters the

same as NWSMTSScanDataSetBegin and pass an additional parameter called cursor. Cursor represents the full path of the last successfully backed up data set. The prerequisite for calling NWSMTSScanDataSetContinue is the successful back up of atleast one data set. The name of the next data set is returned and backup job continues. If there is no last backed-up successful data set, use NWSMTSScanDataSetBegin

---

NOTE: Recovering backup session on cluster failover or failback is not supported in NetWare versions earlier than NetWare 6. In NetWare 5.*x*, shared cluster resources are represented as normal resources.

Recovery of a restore session in case of failover or failback is currently not supported.

How the tape should be managed in a failover or failback situation is not specified. Define a scheme that best suits your need.

Also consider the following while modifying the backup engine:

- Separate backup jobs are required per cluster resource (Shared Cluster enabled NSS pool for File System resource). Only SLP based discovery mechanism is supported for clusters.

2.6 Option Precedence

TSAs have the following precedence (in descending order):

- Exlusions
- Inclusions
- Other options

The algorithms used by the scan and restore functions has the following precedence (in descending order):

- Open, read, or write option (such as NWSM_DO_NOT_OVERWRITE_DATA_SET)
- Exclude a data set
- Include a data set

Exclusion always takes precedence over inclusion in any case and so the scan and restore functions deduce which files are implicitly excluded. The exceptions are the Include/Exclude option, Include Path/Files. These options includes the specified files in a directory and excludes all other files in that same directory. However, all other directories and the files in the entire file system tree are included in the scan (unless other options are used).

2.7 Path Information

Two kinds of paths exist: the fully qualified path and the terminal name. The fully qualified path contains a full path including the primary resource and continuing to the data set itself. The terminal name is the name of the data set and contains no other path information.

SMS requires that all paths for parents are fully qualified paths. The paths for child data sets can be either fully qualified paths or terminal names.

A path's component can include the primary resource name, one or more secondary resource names, and the separator information. For example,

```
VOL1:BIN/DISPLAY.EXE
```

VOL1, BIN, and DISPLAY.EXE are the path nodes and the colon and slash are the separators. The first separator is used to separate the primary resource from the rest of the path, if first separators are supported for the name space type being used to construct the path. For example, VOL1 is the primary resource in the previous example. The second separator is used to separate secondary resources. The engine calls NWSMTSGetNameSpaceTypeInformation to retrieve the separators for a particular name space.

An example of path name that does not support a first separator,/home/user1/binEach path node, home, user1 and bin are the path nodes and the forward slash (/) are the separators. There is no first separator and hence the primary resource is not distinguished from the rest of the path. The engine calls NWSMTSGetNameSpaceTypeInfo to retrieve the separators for a particular name space.

2.7.1 Constructing a Path

The only entity under SMS that should know how to construct a path should be the TSA, since the path for each Target Service type can differ. However, the engine can build a path without knowing the target's path specifications for constructing a path. The following list shows what the engine needs to know about the construction of a path:

- First separators, if supported for the particular name space, always follow the primary resource. To get the pair of valid separators use NWSMTSGetNameSpaceTypeInfo.
- Second separators always precede each secondary resource.
- Second separators follow a parent path node if it is the terminal node in the path.
- If the terminal path node is a child, no second separator follows it.

For NDS paths, the following construction rules are used:

- The engine must reverse the path nodes so that the subordinates are to the left of their superiors. For example, Employee.Department.Company.Country is a reversed path. NWSMTSGetNameSpaceTypeInfo indicates if the name space requires a reversed path and the separators expected by a specific name space.
- If the terminal path node (the left-most path node) is a parent, a dot precedes the name.

To ensure that a path is properly formatted, the engine can call NWSMFixDirectoryPath which does not check a path with a child's data set name.

Resource names can be used with selection types to specify the data sets the user wants to apply an action to. The constructed path contains the resource names to use with the selection type.

2.8 Resources

Resources are data sets that exist on the Target Service and vary for different targets. Resources are parts such as volume, drives, mount points and directories. For example, NetWare resources include file servers and volumes, while DOS resources include drives and Linux file system resources include mount points. There are two types of resources: the primary and the secondary resources.

The following table lists the primary and secondary resources for various target services.

Table 2-2 *Primary and Secondary Resources Table*

Target	Primary Resource	Secondary Resources
Suse Linux Enterprise Server 9.0	Mount points	Directories and files
NetWare 6.x	File Server, volume, and bindery.	Directories and files
	Cluster pools and volumes	Cluster-enabled volumes
NetWare 5.x	File Server and Volumes	Directories and files
NetWare 4.x	File Server and volume	Directories and files
Novell Directory Services	The full Directory	The Directory schema
GroupWise	GroupWise server	GroupWise Post Office

The primary resources can be viewed as the top level of the Target Service. The secondary resources can be viewed as the children of the primary resources. Not all primary resources have children. For example, databases and NetWare's bindery do not have children. Although BTrieve's database may consist of many files, they are backed up as one data set and have no secondary resources.

When a TSA returns its list of primary resources, the following resources may be listed: File Server, Sys: (on NetWare) or mount points (on Linux).

To get the name spaces supported by a resource, an engine passes the primary resource name to the NWSMTSScanSupportedNameSpaces or NWSMTSListSupportedNameSpaces API.

To specify a path to the data set to backup, the engine uses the resource names to build a path to it. Resource names are used with selection type options to specify the name of the data sets to apply the selection type to.

2.8.1 Resource Names

The first primary resource name returned by a TSA is significant. This name always represents the entire Target Service. For example, the TSAs for file servers return NetWare Server as the first primary resource name on NetWare and Linux Server on Linux. The order of the names following the first name has no significance.

2.9 Function Access Scope

An engine's connection (or lack of one) limits the kinds of functions it can call within the TS API. When the engine is not connected to a TSA, the only function that the engine can call are the following:

NWSMConnectToTSA
NWSMListSMDRs
NWSMListTSAs

After connecting to the TSA, the engine can access the following functions:

NWSMTSConnectToTargetService

NWSMTSConnectToTargetServiceEx
NWSMTSGetTargetServiceType
NWSMTSListTargetServices
NWSMReleaseTSA
NWSMTSScanTargetServiceName

After connecting to the Target Service, the engine can access all TS API functions.

Normally, when an engine connects to a TSA and the Target Service, and then disconnects from the Target Service, the engine can still call these functions.

2.10 SIDF

SIDF isolates SMS from the media type and the target's data format. Since SMS treats all file systems generically, a generic data format is required that is provided by SIDF.

SIDF formats data in two parts: fields and sections. A section represents a unit of data and each section consists of a set of related fields. For example, each of the following data units are represented by a corresponding section containing the full path and any trustees:

```
Full path field (first field identifies the section)
Offset to end of section value
Name space type of path
Beginning of each path node in path array
Beginning of each separator in path array
Path
Full path field (last field identifies the end of the section)
```

Through SIDF, similar data from diverse Target Services are presented in the same way. For example, many Target Services have data sets that have a hidden attribute. This attribute may be represented as bit 1 in one Target Service and bit 5 in another Target Service. SIDF allows the representation of the hidden bit to be independent of the bit's position by representing the bit as a field. The field contains the value of the bit (ON or OFF) and a Field IDentifier (FID) that labels the field as a hidden attribute field. When any TSA supports hidden attributes and finds a hidden field, the TSA can set the information.

The data set is represented by a set of related sections such as file header, path information, characteristics, attributes, trustees, data streams, and file trailer for a file data set.

You can call NWSMTSReadDataSet and NWSMTSWriteDataSet to read and write SIDF data. NWSMTSReadDataSet formats all Target Service data into SIDF data while NWSMTSWriteDataSet deformats the formatted data sets and writes it to the Target Service.

2.10.1 Transfer Buffers

A transfer buffer is a multiple of a medium's sector size and specifies how the data is put into the transfer buffer.

There are five components to the transfer buffer.

- The Transfer Buffer Header signifies the beginning of a transfer buffer.
- A Data Set Header is inserted before data is placed into a transfer buffer.

The data follows this header immediately. If there is room for more data, another Data Set Header is inserted immediately after the just inserted data. The data then follows the second Data Set Header. This process continues until there is not enough space for the next Data Set Header or data.

- A Data Set Subheader is placed into a new transfer buffer if data overflows the current transfer buffer.

  The overflow data (data fragment) follows the Data Set Subheader.

- The Data or Data Fragment is composed of three pieces of data:
  - Scan Information
  - Data Set Names
  - Data Set

    The scan information and data set names were returned by NWSMTSScanDataSetBegin or NWSMTSScanNextDataSet when the data set was scanned. The data set was returned by NWSMTSReadDataSet.

2.11 TSA Address

Before the engine connects to TSA, the engine must know the TSA's address. The address format for TSAs is

SMDR_Name.TSA_Name

SMDR_Name is the name of the SMDR that is local to the TSA, and TSA_Name is the name of the TSA. SMDR_Name is required if a DOS TSA is being addressed or if the TSA is remote with respect to the engine.

NOTE: The SMDR assumes the name of its host, and the TSA's name always reflects the Target Service that it services.

Sample Address

```
server1.NetWare Cluster File System
```

and
```
server2.NetWare File System
```

2.12 Log Files

During the backup session, the TSA builds a log file that lists the errors that occurred during the backup session and an optional list of skipped data sets. Both files are automatically deleted when NWSMTSReleaseTargetService is called.

The log file contains:

- Names of the backed up data sets
- Scan information and the name list of each data set that were returned by NWSMTSScanDataSetBegin and NWSMTSScanNextDataSet
- Addresses of the transfer buffer on the media

The error log is an ASCII file that lists the errors and warnings that occurred during the backup session.

The skipped data sets log is a binary file that lists the data sets that met the scanning criteria but were skipped because they could not be backed up. Each entry in the skipped data sets file is an NWSM_DATA_SET_NAME_LIST structure where the *bufferSize* field does not contain a valid value. The reserved field of each entry contains an error code that describes the reason for skipping the data set. The engine should translate the error code to a string by calling NWSMConvertError (see *Storage Management Services Utility Library*).

2.13 Scan Order

All TSAs must follow a specified scanning order if parent handles are used. The scan travels down the left side of the tree before traversing the right side as shown by the following steps:

1. Upon entering a parent, all the children of the current parent are scanned.

2. The next parent to scan is found. If the current parent has parents or subdirectories, each one is scanned by repeating Steps 1 and 2.

3. If the current parent does not have any more parents to scan, the scan of the current parent is considered finished and the scan traverses up the file system tree until the next parent to scan is found. Steps 1-3 are repeated.

4. If there are no more parents to find, the scan ends.

To illustrate this, let's consider the root of the scan as parent A. When the scan begins, the name of parent A is returned. Next, the children's names of parent A are returned which are: a, b, and c. Since there are no more children in parent A to scan, the name of the first parent in A-which is B-is returned. The steps are then repeated so that the children's names of parent B are returned next. Since there are no more children in B to scan, the next parent in B-which is C is scanned. Once the scan is done with parent C, the scan continues with parent D. The process continues until the scan stops with parent I

Figure 2-4 *Scan Order*
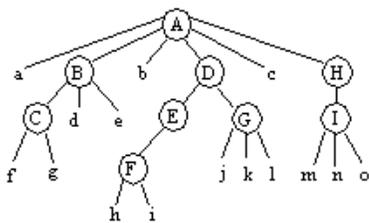
2.14 Other Documents

Standard ECMA-208 is available free of charge from:

- ECMA,114 Rue du Rhône, CH-1204 Geneva, Switzerland

- Fax:+41 22 849.60.01

- Internet: Standard ECMA-208 (http://www.ecma-international.org/publications/standards/Ecma-208.htm)

- As the E208-DOC.EXE or E208-PSC.EXE file from ECMANEWS

Target Services Functions 3

This documentation alphabetically lists the Target Services functions and describes their purpose, syntax, parameters, and return values.

Target Services functions perform the following types of tasks:

- Maintaining connections with the TSA and Target Services
- Providing a mechanism for specifying TSA-specific options
- Allowing the engine to read, write, modify, and scan the data on the Target Service
- Retrieving information about the Target Service
- Formatting data according to the SIDF specification at the data set level
- Manipulating path information

3.1 Backup Functions

The following functions backup and scan the data sets specified by the TSA options:

NWSMTSScanDataSetBegin

Begins a data set scan for the specified data set and returns information about the first data set found.

Syntax

```
#include <smstsapi.h>
#include <sms.h>

CCODE NWSMTSScanDataSetBegin (
   UINT32                  connection,
   NWSM_DATA_SET_NAME_LIST *resourceName,
   NWSM_SCAN_CONTROL       *scanControl,
   NWSM_SELECTION_LIST     *selectionList,
   UINT32                  *sequence,
   NWSM_SCAN_INFORMATION   **scanInformation,
   NWSM_DATA_SET_NAME_LIST  **dataSetNames);
```

Parameters

connection

   (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServicEx.

resourceName

   (IN) Points to the starting point or path of the resource to scan.

scanControl

   (IN) Points to the options of the data sets to scan or ignore (see Scan Control Options).

selectionList

   (IN) Points to the data sets to scan or ignore and contains the name as it appears under every name space supported by resourceName.

sequence

   (OUT) Points to the scanning sequence value (doesn't need to be initialized since it is set by the TSA).

scanInformation

   (OUT) Points to the general attributes and information of the first data set that meets the scanning criteria (optional).

dataSetNames

   (OUT) Points to the name of the first data set that met the scanning criteria (optional).

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFAF	NWSMTS_CLUSTER_TARGET_DOES_NOT_EXIST
0xFFFBFFFB	NWSMUT_OUT_OF_MEMORY
0xFFFBFFFC	NWSMUT_NO_MORE_NAMES
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER
0xFFFBFFFF	NWSMUT_INVALID_HANDLE
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFC9	NWSMTS_OUT_OF_MEMORY
0xFFFDFFDA	NWSMTS_INVALID_SEL_LIST_ENTRY
0xFFFDFFDC	NWSMTS_INVALID_PATH
0xFFFDFFDD	NWSMTS_INVALID_PARAMETER
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFDFFF2	NWSMTS_DATA_SET_NOT_FOUND
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Engine Developers

To build dataSetNames, the Data Set Name Functions described in *Storage Management Services Library* can be used. The scan information and data set names are known as the data set information by SIDF.

Remarks

Before NWSMTSScanDataSetBegin is called, the engine must be connected to a TSA and Target Service.

NWSMTSScanDataSetBegin can be called multiple times to initiate concurrent scans. Each time it is called, a different resource name is passed. If concurrent scans are used, the engine should ensure that the data area covered by a resource does not overlap the area covered by another resource.

NWSMTSScanDataSetBegin begins scanning where resourceName is pointing. The engine should copy the information returned by scanInformation and dataSetNames since the buffer is reused when NWSMTSScanDataSetBegin is called again. To free scanInformation and dataSetNames, call NWSMTSScanDataSetEnd.

resourceName can contain a primary resource, a path to a parent, or a fully qualified path to a child. It can be obtained from previous dataSetNames values, or by calling NWSMTSScanTargetServiceResource or NWSMTSListTSResources.

resourceName should be set to the first primary resource name instead of its parent.

There are two special values for resourceName as follows:

- ERROR LOG is an ASCII file, which list the errors that occurred during the back-up/restore session.
- SKIPPED DATA SETS is a log that indicates the data sets that met the scanning criteria, but were skipped (see Log Files).

Both files are automatically deleted when NWSMTSReleaseTargetService is called. If *resourceName* is ERROR LOG or SKIPPED DATA SETS, *scanInformation* and *dataSetNames* will not contain any valid information.

scanControl determines not only the kinds attribute information returned, but also how much path information is returned, and under which name spaces to search for data sets.

To scan for all data sets, set selectionList to NULL. Otherwise, set it to a list containing the scanning patterns, paths, or explicit data set names to filter (see Data Set Selection Options).

To build resourceName and selectionList, the Data Set Name Functions listed in *Storage Management Services Utility Library* can be used.

For any path built for resourceName or selectionList, the engine must ensure that the path is properly formatted for the intended name space.

For engines that use SMS DI or SIDF, scanInformation and dataSetNames can be formatted and placed into SIDF transfer buffers by calling NWSMSetNewRecordHeader and NWSMUpdateRecordHeader.

The following table lists the data sets that the file system TSA excludes from the scan operation even when the data sets are explicitly included

Target Resource	Files Excluded During the Scan	File Description
NetWare 4.x	Volume:NET$AUDIT	The volume audit files
NetWare versions excluding NetWare 3.x	SYS:SYSTEM/SERVDATA.NDS	NDS-specific data
	SYS:SYSTEM/DSMISC.LOG	NDS-specific log
All NetWare versions	SYS:SYSTEM/TSA/ \TSA$TEMP.*	TSA temporary files
	SYS:BACKOUT.TTS	Transaction Tracking files
All Linux versions	/proc	pseudo file system
	/dev	All peripheral devices
	/sys	Virtual file system
	devpts type mounts	All preipheral devices
	proc type mounts	pseudo file system
	tmpfs type mounts	temporary file system
	binfmt_misc type mounts	binary format registration
	sysfs type mounts	virtual file system

If scanControl is set to NULL, the TSA scans the Target Service as if the structure's values are 0.

The dataSetNames list can be used in many ways such as displaying or naming the data set in a database.

NWSMTSScanDataSetBegin can also perform the following tasks in addition to scanning for data sets to back up:

- Build a list of directories and files to display to the user
- Check if a data set exists before overwriting it during a restore session
- Scan for data sets to be deleted or renamed by passing sequence to NWSMTSDeleteDataSet and NWSMTSRenameDataSet respectively

See Also

NWSMTSBuildResourceList (page 97), NWSMTSDeleteDataSet (page 84), NWSMTSListTSResources (page 134), NWSMTSRenameDataSet (page 90), NWSMTSScanDataSetEnd (page 58), NWSMTSScanNextDataSet (page 53), NWSMTSScanTargetServiceResource (page 141)

Example

```
#include <smsutapi.h> #include <smstsapi.h>
#include <sms.h> /* include file for scan attributes. */
NWSM_DATA_SET_NAME_LIST *resourceName, *dataSetName = NULL;
NWSM_DATA_SET_NAME name;
NWSM_SCAN_CONTROL scanControl;
NWSM_SCAN_INFORMATION *scanInformation = NULL;
NWSM_SELECTION_LIST *selectionList;
UINT32 sequence, dataSetHandle, openModes, bytesRead,
transferBufferSpaceLeft, bytesWritten, transferBufferOffset,
maxTransferBufferSize;
NWSM_RECORD_HEADER_INFO recordHeaderInfo = {0};
BUFFERPTR transferBuffer;
/* Setup resourceName-see NWSMTSListTSResources and
NWSMTSScanTargetServiceResource. */
...
/* Gather the TSA's options and get the user's input*/
...
/* Setup scanControl-see the example code for
NWSMTSGetUnsupportedOptions and
NWSMTSGetTargetScanTypeString. */
...
/* Setup selectionList-see NWSMTSGetTargetSelectionTypeStr.*/
/* begin the back-up session. */
ccode = NWSMTSScanDataSetBegin(connection, resourceName, &scanControl,
selectionList,
&sequence, &scanInformation, &dataSetName);
if(ccode)
{
if(ClusterEnabled && IsClusterError(ccode)) // For ccode refer
"Recovery Error Codes" on page 56 if(!Reconnect)
```

```
// returns 0 if success after n retries
{// Cannot call NWSMTSScanDataSetContinue, as there is nothing to
continue from.
// as this being the beginning call
ccode = NWSMTSScanDataSetBegin(connection, resourceName, &scanControl,
selectionList,
&sequence, &scanInformation, &dataSetName);
}
else
// Handle reconnection error.
}
else
// Handle error
}
/
*************************************************************************
****
* Constructing the name space type info for the resource being
* backed up. Will need this for constructing the cursor that is to
* be passed to TSA after a fail over / fail back. This is done only
* once for a resource.
*************************************************************************
****/
if(!ccode && (sessionInfo->clusterparams)) // if ScanBegin is Success
{
NWSMGetDataSetName(dataSetNameList, 0L, &dataSetName);
cNameSpaceType=dataSetName.nameSpaceType;
NWSMTSGetNameSpaceTypeInfo(
sessionInfo->connection,
cNameSpaceType,
&cReverseOrder,
&cFirstSeparater,
&cSecondSeperater);
}
/* Set the open modes from the user's selection. The modes were
received when
the TSA options where gathered */
openModes = user selected modes;
/* Create transfer buffer. For more information about
maxTransferBufferSize, see
Storage Management Services Device Interface. */
transferBuffer = (BUFFERPTR)calloc(1, maxTransferBufferSize);
while(!ccode) // Successful ScanBegin
{
ccode = NWSMTSOpenDataSetForBackup(connection, sequence, openModes,
&dataSetHandle);
if(ccode)
{
if(ClusterEnabled && IsClusterError(ccode)) // ccode can be one of
NWSMDR_OPEN_FAILURE
{// or NWSMDR_READ_FAILURE or NWSMDR_WRITE_FAILURE or
NWSMDR_RECONNECT_FAILURE or NWSMDR_TRANSPORT_FAILURE
if(!Reconnect) // returns 0 if success after n retries
{
```

```
if(cParent || cChild) // if there is last successful data set
{
// construct parent or child full path for the last successfully backed
up dataset
// if only child, cat parent along with it.
NWSMPutOneName(
(void **)&lastDataSetNameList,// cursor to be used by ScanContinue
cNameSpaceType,// Name Space Type obtained after successful ScanBegin
call above
0L,//First param contains DataSetnameList so this is set to zero
cReverseOrder,
(char *)&cFirstSeparater,
(char *)&cSecondSeperater,
(char *)&(cFullPath->string)); // The full path of last successfully
backed up child or parent
ccode=NWSMTSScanDataSetContinue(sessionInfo->connection,
resourceName,
&sessionInfo->scanControl,
sessionInfo->selectionList,
lastDataSetNameList, // cursor
&sequence,
&sessionInfo->scanInfo,
&dataSetNameList);
}
else // if this is the first data set, then there is nothing to
continue from
{ // so call ScanBegin to start from first
ccode = NWSMTSScanDataSetBegin(connection, resourceName, &scanControl,
selectionList,
&sequence, &scanInformation, &dataSetName);
}
}
else
// Handle reconnection error.
}
else
// Handle error
}
/* Display the data set name to the user. See Storage Management
Services
Utility Library for the function below. */
NWSMGetOneName(dataSetName, &name);
if (scanInformation->parentFlag)
{
/* Display name.name as a parent (e.g., directory). */
...
}
else
{
/* Display name.name as a child (e.g., a file). */
...
}
/* Usually the transfer buffer size is negotiated between SMS DI and
the
```

```
engine-see NWSMSDSessionOpenForWriting in the SMS DI documentation.
However, it
is set to an assumed size to keep this example relatively simple. For
information
on transferBufferDataOffset see the Storage Device API document's
NWSMSDSessionOpenForWriting. */
transferBufferSpaceLeft = maxTransferBufferSize -
transferBufferDataOffset;
/* Put the data set information and data set data into an SIDF data
set, and
then into the transfer buffer. Here, we assume that the transfer buffer
is
defined somewhere else (see the Storage Device API document's
NWSMSDSessionOpenForWriting). For information on NWSMDOSTimeToECMA see
Storage Management Services Utility Library. */
recordHeaderInfo.isSubRecord = FALSE;
recordHeaderInfo.dataSetName = dataSetName;
recordHeaderInfo.scanInformation = scanInformation;
NWSMDOSTimeToECMA(NWSMGetCurrentTime(),
&recordHeaderInfo.archiveDateAndTime);
NWSMSetNewRecordHeader(&transferBuffer, &transferBufferSpaceLeft,
&bytesWritten, CRC_YES, &recordHeaderInfo);
/* Here we assume that the transfer buffer always has enough room to
receive
the data set data, that NWSMTSReadDataSet retrieves all of the data set
data on
the first call, and that the transfer buffer is completely filled. */
ccode = NWSMTSReadDataSet(connection, dataSetHandle,
transferBufferSpaceLeft,
&bytesRead, &transferBuffer[transferBufferDataOffset]);
if(ClusterEnabled)
{
if(!ccode)
{
if(scanInformation->parentFlag)
// Store parent as it is successfully backedup
else
// Store child as it is successfully backedup
}
if(IsClusterError(ccode)) // ccode can be one of NWSMDR_OPEN_FAILURE
{// or NWSMDR_READ_FAILURE or NWSMDR_WRITE_FAILURE or
NWSMDR_RECONNECT_FAILURE or NWSMDR_TRANSPORT_FAILURE
if(!Reconnect) // returns 0 if success after n retries
{
if(cParent || cChild) // if there is last successful data set
{
// construct parent or child full path for the last successfully backed
up dataset
NWSMPutOneName(
(void **)&lastDataSetNameList,// cursor to be used by ScanContinue
cNameSpaceType,// Name Space Type obtained after successful ScanBegin
call above
0L,//First param contains DataSetnameList so this is set to zero
cReverseOrder,
```

```
char *)&cFirstSeparater,
(char *)&cSecondSeperater,
(char *)&(cFullPath->string)); // The full path of last successfully
backed up child or parent
ccode=NWSMTSScanDataSetContinue(sessionInfo->connection,
resourceName,
&sessionInfo->scanControl,
sessionInfo->selectionList,
lastDataSetNameList, // cursor
&sequence,
&sessionInfo->scanInfo,
&dataSetNameList);
}
else // if this is the first data set, then there is nothing to
continue from
{ // so call ScanBegin to start from first
ccode = NWSMTSScanDataSetBegin(connection, resourceName, &scanControl,
selectionList,
&sequence, &scanInformation, &dataSetName);
}
}
else
// Handle reconnection error.
}
else
// Handle error
}
/* Keep track of how many bytes are written to the transfer buffer.
This
information is needed for recordHeaderInfo */
bytesWritten =+ bytesRead;
recordHeaderInfo.recordSize += bytesRead;
/* Since the transfer buffer is full, update the data set header size
and
CRC information, then send the transfer buffer to SMS DI. SMS DI will
write the transfer buffer to the media.
See NWSMSDSessionWriteData in Storage Device API
for more information. */
NWSMUpdateRecordHeader(&recordHeaderInfo);
...
/* close the data set and get the next one */
NWSMTSCloseDataSet(connection, &dataSetHandle);
ccode = NWSMTSScanNextDataSet(connection, &sequence, &scanInformation,
&dataSetName);
if(ccode)
{
if(ClusterEnabled && IsClusterError(ccode)) // ccode can be one of
NWSMDR_OPEN_FAILURE
{// or NWSMDR_READ_FAILURE or NWSMDR_WRITE_FAILURE or
NWSMDR_RECONNECT_FAILURE or NWSMDR_TRANSPORT_FAILURE
if(!Reconnect) // returns 0 if success after n retries
{
if(cParent || cChild) // if there is last successful data set
{
```

```
// construct parent or child full path for the last successfully backed
up dataset
// if only child, cat parent along with it.
NWSMPutOneName(
(void **)&lastDataSetNameList,// cursor to be used by ScanContinue
cNameSpaceType,// Name Space Type obtained after successful ScanBegin
call above
0L,//First param contains DataSetnameList so this is set to zero
cReverseOrder,
(char *)&cFirstSeparater,
(char *)&cSecondSeperater,
(char *)&(cFullPath->string)); // The full path of last successfully
backed up child or parent
ccode=NWSMTSScanDataSetContinue(sessionInfo->connection,
resourceName,
&sessionInfo->scanControl,
sessionInfo->selectionList,
lastDataSetNameList, // cursor
&sequence,
&sessionInfo->scanInfo,
&dataSetNameList);
}
else // if this is the first data set, then there is nothing to
continue from
{ // so call ScanBegin to start from first
ccode = NWSMTSScanDataSetBegin(connection, resourceName, &scanControl,
selectionList,
&sequence, &scanInformation, &dataSetName);
}
}
else
// Handle reconnection error.
}
else
// Handle error
}
} /* end while loop */
/* If the scanning functions did not return an error, end the scan */
NWSMTSScanDataSetEnd(connection, &sequence, &scanInformation,
&dataSetName);
```

NWSMTSGetTargetServiceAPIVersion

Returns the API version of the target service.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSGetTargetServiceAPIVersion (
   UINT32    connection,
   UINT32   *majorVersion,
   UINT32   *minorVersion);
```

Parameters

connection

> (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServiceEx.

majorversion

> (OUT) Specifies the major version of the TSA.

minorversion

> (OUT) Specifies the minor version of the TSA.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFDFFB1	NWSMTS_INTERNAL_ERROR
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

This API provides an internally negotiated minimum supported version for the connection. The version returned is specific to the connection and is based on the API version supported by SMDR at the backup server and the SMDR and TSA at the target server.

Before this API is called, the engine must be connected to the TSA.

The API is supported by the following versions:

- NetWare 6.5 Support Pack 1 and later

- NetWare 6.0 Support Pack 4
- NetWare 5.1 Support Pack 7 and later

NOTE: This symbol is not present in the SMS import file, as all versions of SMS do not export this symbol. To use this function, programmatically import the symbol and invoke it.

See Also

NWSMTSConnectToTargetService (page 65), NWSMTSConnectToTargetServiceEx (page 68)

NWSMTSOpenDataSetForBackup

Opens the data set referenced by the sequence value that returned the scanning functions.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSOpenDataSetForBackup (
   UINT32   connection,
   UINT32   sequence,
   UINT32   mode,
   UINT32 *dataSetHandle);
```

Parameters

connection

   (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServiceEx.

sequence

   (IN) Specifies the sequence number returned by NWSMTSScanDataSetBegin or NWSMTSScanNextDataSet.

mode

   (IN) Specifies the generic and TSA-specific open mode to apply to the data set (see Open Modes for possible values).

dataSetHandle

   (OUT) Points to a handle used to subsequently call NWSMTSReadDataSet or NWSMTSCloseDataSet.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFAF	NWSMTS_CLUSTER_TARGET_DOES_NOT_EXIST
0xFFFDFFB5	NWSMTS_WRITE_ERROR
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFC9	NWSMTS_OUT_OF_MEMORY
0xFFFDFFCC	NWSMTS_OPEN_ERROR
0xFFFDFFCD	NWSMTS_OPEN_DATA_STREAM_ERR
0xFFFDFFD8	NWSMTS_INVALID_SEQUENCE_NUMBER

0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFDFFF3	NWSMTS_DATA_SET_IS_OPEN
0xFFFDFFF5	NWSMTS_DATA_SET_IN_USE
0xFFFDFFF6	NWSMTS_DATA_SET_EXECUTE_ONLY
0xFFFDFFFB	NWSMTS_CLOSE_BINDERY_ERROR
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMTSOpenDataSetForBackup is called, NWSMTSScanDataSetBegin must be called to initiate the scan.

The TSA must ensure that no attributes are altered on the Target Service during the backup session.

NWSMTSOpenDataSetForBackup prepares the Target Service data set to be backed up. To close the data set, call NWSMTSCloseDataSet.

To determine the existing TSA-specific open modes for a given TSA, call NWSMTSGetOpenModeOptionString.

See Also

NWSMTSCloseDataSet (page 82), NWSMTSScanDataSetBegin (page 38), NWSMTSScanNextDataSet (page 53)

NWSMTSReadDataSet

Reads a data set on the Target Service, formats the data according to SIDF, and returns it in a buffer.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSReadDataSet (
   UINT32      connection,
   UINT32      dataSetHandle,
   UINT32      bytesToRead,
   UINT32     *bytesRead,
   BUFFERPTR   buffer);
```

Parameters

connection

(IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServiceEx.

dataSetHandle

(IN) Specifies the data set handle returned by NWSMTSOpenDataSetForBackup.

bytesToRead

(IN) Specifies the amount of free space in buffer.

bytesRead

(OUT) Points to the number of bytes read into buffer.

buffer

(OUT) Points to the buffer to contain the data (must be at least bytesToRead bytes large).

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFAF	NWSMTS_CLUSTER_TARGET_DOES_NOT_EXIST
0xFFFDFFB5	NWSMTS_WRITE_ERROR
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFC6	NWSMTS_READ_ERROR
0xFFFDFFC9	NWSMTS_OUT_OF_MEMORY
0xFFFDFFE5	NWSMTS_INVALID_DATA_SET_HANDLE

0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFDFFF9	NWSMTS_CREATE_ERROR
0xFFFDFFFF	NWSMTS_ACCESS_DENIED
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMTSReadDataSet is called, NWSMTSScanDataSetBegin must be called to initiate the scan and NWSMTSOpenDataSetForBackup must be called to open the data set.

The data returned in *buffer* is the data mentioned in *Standard ECMA-208*. The engine formats the scan information and data set names (both are jointly known as data set information under SIDF) and places the result and data into a transfer buffer. Call NWSMSetNewRecordHeader and NWSMUpdateRecordHeader to put the information into the transfer buffer.

If buffer cannot contain all of the data, NWSMTSReadDataSet must be called repeatedly to retrieve all the data. bytesRead and bytesToRead indicates if all the data has been read. If bytesRead is equal to bytesToRead, there is more data to read.

See Also

Example

NWSMTSScanNextDataSet

Continues the scan started by NWSMTSScanDataSetBegin and returns the next data set that meets the scanning criteria.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSScanNextDataSet (
    UINT32                        connection,
    UINT32                        *sequence,
    NWSM_SCAN_INFORMATION   **scanInformation,
    NWSM_DATA_SET_NAME_LIST **dataSetNames);
```

Parameters

connection

 (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServiceEx.

sequence

 (IN/OUT) Points to the sequence value from NWSMTSScanDataSetBegin or NWSMTSScanNextDataSet.

scanInformation

 (OUT) Points to the scan information for one data set (optional).

dataSetNames

 (OUT) Points to the list that contains a data set's name as it appears under every supported name space.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFAF	NWSMTS_CLUSTER_TARGET_DOES_NOT_EXIST
0xFFFBFFFB	NWSMUT_OUT_OF_MEMORY
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFC9	NWSMTS_OUT_OF_MEMORY
0xFFFDFFC4	NWSMTS_SCAN_ERROR
0xFFFDFFD1	NWSMTS_NO_MORE_DATA_SETS

0xFFFDFFD8	NWSMTS_INVALID_SEQUENCE_NUMBER
0xFFFDFFDC	NWSMTS_INVALID_PATH
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFDFFEB	NWSMTS_GET_NAME_SPACE_ENTRY_ERR
0xFFFDFFF3	NWSMTS_DATA_SET_IS_OPEN
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMTSScanNextDataSet is called, NWSMTSScanDataSetBegin must be called to initiate a scan.

Call NWSMTSScanNextDataSet repeatedly until all data sets are found or an error occurs.

The engine should copy the information returned by scanInformation and dataSetNames since the buffer is reused when NWSMTSScanNextDataSet is called again.

The engine must not call NWSMTSScanNextDataSet after it returns NWSMTS_NO_MORE_DATA_SETS, unless sequence from a different NWSMTSScanDataSetBegin or NWSMTSScanNextDataSet function is passed. Otherwise, NWSMTS_INVALID_SEQUENCE_NUMBER will be returned.

If there are no more data sets to scan, NWSMTSScanNextDataSet returns NWSMTS_NO_MORE_DATA_SETS, sets sequence to zero, and frees dataSetNames and scanInformation.

If NWSMTSScanNextDataSet does not return an error other than cluster error(valid only for cluster enabled backups), the engine must call NWSMTSScanDataSetEnd to end the scan.

See Also

Example

See the example for .

NWSMTSScanDataSetContinue

Continues the scan from the specified data set and returns information about the next data set.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSScanDataSetContinue (
   UINT32                   connection,
   NWSM_DATA_SET_NAME_LIST  *resourceName,
   NWSM_SCAN_CONTROL        *scanControl,
   NWSM_SELECTION_LIST      *selectionList,
   NWSM_DATA_SET_NAME_LIST  *cursorDataSetName,
   UINT32                   *sequence,
   NWSM_SCAN_INFORMATION    **scanInformation,
   NWSM_DATA_SET_NAME_LIST  **dataSetNames);
```

Parameters

connection

   (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServiceEx.

resourceName

   (IN) Points to the starting point or path of the resource to scan.

scanControl

   (IN) Points to the attributes and characteristics of all data sets to scan or ignore (see "Scan Control Options" on page 26).

selectionList

   (IN) Points to the data sets to scan or ignore and contains the name as it appears under every name space supported by resourceName.

cursorDataSetName

   (IN) Points to the point or the full path of last successfully backed up data set name, to continue the scan.

sequence

   (OUT) Points to the scanning sequence value (does not need to be initialized as it is set by the TSA).

scanInformation

   (OUT) Optional. Points to the general attributes and information of the first data set that meets the scanning criteria.

dataSetNames

   (OUT) Optional. Points to the name of the first data set that met the scanning criteria.

Return Values

See for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFAF	NWSMTS_CLUSTER_TARGET_DOES_NOT_EXIST
0xFFFBFFFB	NWSMUT_OUT_OF_MEMORY
0xFFFBFFFC	NWSMUT_NO_MORE_NAMES
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER
0xFFFBFFFF	NWSMUT_INVALID_HANDLE
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFC9	NWSMTS_OUT_OF_MEMORY
0xFFFDFFDA	NWSMTS_INVALID_SEL_LIST_ENTRY
0xFFFDFFDC	NWSMTS_INVALID_PATH
0xFFFDFFDD	NWSMTS_INVALID_PARAMETER
0xFFFDFFE4	NWSMTS_INVALID_DATA_SET_NAME
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFDFFF2	NWSMTS_DATA_SET_NOT_FOUND
0xFFFDFFB1	NWSMTS_INTERNAL_ERROR
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Recovery Error Codes

This section lists the return values returned by the following TSA and SMDR APIs, during cluster failover/failback.

- NWSMTSScanDataSetBegin
- NWSMTSOpenDataSetForBackup
- NWSMTSReadDataSet
- NWSMTSScanNextDataSet

0xFFFDFFAF	NWSMTS_CLUSTER_TARGET_DOES_NOT_ EXIST	Cluster target does not exist
0xFFFEFFCE	NWSMDR_OPEN_FAILURE	Unable to open the configuration file for reading
0xFFFEFFCD	NWSMDR_READ_FAILURE	Unable to read from source
0xFFFEFFCC	NWSMDR_WRITE_FAILURE	Unable to write to the destination

0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE	The transport mechanism has failed
0xFFFEFFB1	NWSMDR_DISCONNECTED	Resource disconnected

Remarks

- This API is cluster specific and is not to be used for normal backup failures
- This API doesn't support restore recovery during cluster failover/failback.
- If scanning is in progress, call NWSMTSScanDataSetEnd before reconnection.
- After cluster failover/failback, release the current connections of engine to TSA or SMDR by calling NWSMTSReleaseTargetService or NWSMReleaseTSA respectively.
- Re-establish the connection using NWSMConnectToTSA and NWSMTSConnectToTargetService or NWSMTSConnectToTargetServiceEx.
- The cursor should contain the last successfully backed-up data set information. If there are no such data sets during failover/failback, use NWSMTSScanDataSetBegin.

NOTE: This symbol is not present in the SMS import file, as all versions of SMS do not export this symbol. To use this function, programmatically import the symbol and invoke it.

See Also

NWSMConnectToTSA (page 60), NWSMReleaseTSA (page 76), NWSMTSConnectToTargetService (page 65), NWSMTSConnectToTargetServiceEx (page 68), NWSMTSScanDataSetBegin (page 38)

Example

See the example for NWSMTSScanDataSetBegin (page 38).

NWSMTSScanDataSetEnd

Stops the scan started by NWSMTSScanDataSetBegin.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSScanDataSetEnd
   UINT32                      connection,
   UINT32                     *sequence,
   NWSM_SCAN_INFORMATION   **scanInformation,
   NWSM_DATA_SET_NAME_LIST **dataSetNames);
```

Parameters

connection

   (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServiceEx.

sequence

   (IN/OUT) Inputs a pointer to the sequence value returned by NWSMTSScanDataSetBegin or NWSMTSScanNextDataSet. Outputs zero.

scanInformation

   (IN/OUT) Inputs a pointer to the memory to be freed. Outputs NULL.

dataSetNames

   (IN/OUT) Inputs a pointer to the memory to be freed. Outputs NULL.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFD8	NWSMTS_INVALID_SEQUENCE_NUMBER
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFDFFF3	NWSMTS_DATA_SET_IS_OPEN
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMTSScanDataSetEnd is called, NWSMTSScanDataSetBegin must be called to initiated a scan.

NWSMTSScanDataSetEnd stops a scan prematurely. Do not call NWSMTSScanDataSetEnd if NWSMTSScanDataSetBegin or NWSMTSScanNextDataSet returns an error.

See Also

NWSMTSReleaseTargetService (page 75), NWSMReleaseTSA (page 76)

Example

See the example for NWSMTSScanDataSetBegin (page 38).

3.2 Connection Functions

The following functions connect the engine to the TSA and the Target Service:

NWSMConnectToTSA

Connects the backup engine to a TSA.

Syntax

```
#include <smstsapi.h>

CCODE NWSMConnectToTSA (
   STRING   tsaName,
   UINT32  *connection);
```

Parameters

tsaName

   (IN) Specifies the name of the TSA to connect to returned by NWSMListTSAs.

connection

   (OUT) Points to the connection information used for connecting to and getting information about the Target Service.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFEFFF5	NWSMDR_SMDR_CONNECT_FAILURE
0xFFFEFFF6	NWSMDR_NO_SUCH_SMDR
0xFFFEFFF7	NWSMDR_TSA_NOT_LOADED
0xFFFEFFDF	NWSMDR_UNKNOWN_ADDRESS
0xFFFEFFFD	NWSMDR_OUT_OF_MEMORY
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFF1	NWSMDR_NO_MORE_CONNECTIONS
0xFFFEFFE9	NWSMDR_INVALID_CONTEXT

Remarks

After connecting to the TSA, the engine can access only information about the Target Service. To get information about the target, see NWSMTSGetTargetServiceType (page 71). To access the target data, see NWSMTSConnectToTargetService (page 65).

An engine can have one or more concurrent connections to one or more TSAs. For each connection, the engine must call NWSMConnectToTSA once.

NOTE: The internal primary resource list is created when NWSMConnectToTSA is called.

See Also

NWSMTSConnectToTargetService (page 65), NWSMTSConnectToTargetServiceEx (page 68), NWSMTSListTargetServices (page 73), NWSMReleaseTSA (page 76)

Example

```
UINT32 connection;
STRING TSAName;

/* Find a TSA to connect to-see NWSMListTSAs. */
NWSMConnectToTSA(TSAName, &connection);
/* Find a Target Service through the connected TSA-see
NWSMTScanTargetServiceName and NWSMTSListTargetServiceName. */
/* Connect to the target-see NWSMTSConnectToTargetService or
NWSMTSConnectToTargetServiceEx . */
```

NWSMListTSAs

Builds a list of TSAs as specified by a scan pattern.

Syntax

```
#include <smstsapi.h>
#include <smsdrapi.h>

CCODE NWSMListTSAs (
    STRING              scanPattern,
    NWSM_NAME_LIST (page 161) **serviceAgentNameList);
```

Parameters

scanPattern

   (IN) Specifies the pattern to search for.

serviceAgentNameList

   (OUT) Points to the names of all active TSAs.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFEFFF6	NWSMDR_NO_SUCH_SMDR
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFDF	NWSMDR_UNKNOWN_ADDRESS
0xFFFEFFB8	NWSMDR_PROTOCOL_NOT_FOUND
0xFFFEFFFD	NWSMDR_OUT_OF_MEMORY
0xFFFEFFFF	NWSMDR_INVALID_CONTEXT

Remarks

See TSA Scanning Pattern for a comprehensive list of legal scan patterns.

The TSA name in serviceAgentNameList-> name has a maximum buffer size of NWSM_MAX_TARGET_SRVC_NAME_LEN bytes and has the following format:
`smdrName.tsaName`

smdrName and tsaName contain either a complete name or partial name indicated by wildcard characters. smdrName is always the name of the Target Service where the TSA.

The following table lists the various wildcard options available for listing the TSAs.

Value	Description
smdrName.tsaName	Returns an exact match.
smdrName	Returns all TSAs on Target Services named smdrName.
*smdrName	Returns all TSAs on Target Services that have names that end with smdrName.
smdrName*	Return all TSAs on Target Services that have names that begin with smdrName.
smdrName.*	Return all TSAs on Target Service smdrName.
*.tsaName	Return all TSAs named tsaName.
*smdrName.tsaName	Return all TSAs named tsaName on Target Services that have names that end with smdrName.
smdrName.tsaName*	Return all TSAs that begin with tsaName on Target Service smdrName.
*	Return all available TSAs.
.	Return all available TSAs.

For example, "DJ.Linux File System" is a legal name. "DJ" is the smdrName and "Linux File System" is the tsaName. Note that smdrName cannot contain any periods, while tsaName can. There is no limit to the size of the name.

NWSMListTSAs may slow down if smdrName contains wild cards. To speed up the response time, the engine can call NWSMListSMDRs (see Finding the TSA, Method 1).

If SMDR and TSA are loaded on a cluster, NWSMListTSAs can return two different strings for file system TSA. If the name of the shared cluster resource (virtual NCP server name) is passed to this call, then the string, *NetWare Cluster File System* is returned. If the node name is passed, *NetWare File System or Linux File System* is returned as the case may be.

See Also

Example

```
/* This example shows method that may speed up the listings of TSAs */
#include <smstsapi.h>
#include <smsdrapi.h> /* include smsdrapi.h after smstsapi.h */
#include <string.h>

NWSM_NAME_LIST *smdrList = NULL
*serviceAgentNameList = NULL;
char pat[2] = "*";
STRING scanPattern = pat;
CHAR chosenSMDR[120], chosenTSA[120]; /* arbitrary length */

/* Get the list of all SMDRs See document Storage Management Services
Utility Library for a description of NWSMListSMDRs. */
```

```
/* Select a SMDR. The SMDR assumes the name of the Target Service where
the SMDR resides. */

if (smdrList->next)
{
   /* There is more than one SMDR. Display the SMDR list and have the
user select one.
The selected name is copied into chosenSMDR. */
}

else /* There is only one SMDR name. Automatically select it. */
   strcpy(chosenSMDR, smdrList->name);

/* Append ".*" to the smdr name to indicate that all TSAs on the
specified Target Service should be returned. */
strcat(chosenSMDR, ".*");
/* List all the TSAs that reside with the chosen SMDR. */
NWSMListTSAs(chosenSMDR,   &serviceAgentNameList);

/* Check if we have more than one TSA name. */
if (serviceAgentNameList->next)
{
   /* Have the user choose a TSA name, then copy it into chosenTSA. */
}
else
   strcpy(chosenTSA, serviceAgentNameList->name);
/* Connect to the selected TSA-see NWSMConnectToTSA. */
```

NWSMTSConnectToTargetService

Connects an engine to a specified Target Service.

Syntax

```
#include <smsutapi.h>
#include <smstsapi.h>

CCODE NWSMTSConnectToTargetService (
   UINT32  *connection,
   STRING   targetServiceName,
   STRING   targetUserName,
   void    *authentication);
```

Parameters

connection

(IN/OUT) Specifies a pointer to the connection information from NWSMConnectToTSA. Outputs a pointer to the connection to the Target Service in addition to the input information.

targetServiceName

(IN) Specifies the Target Service name returned by NWSMTSScanTargetServiceName or NWSMTSListTargetServices.

targetUserName

(IN) Specifies the user name on a file server (maximum size is NWSM_MAX_TARGET_USER_NAME_LEN).

authentication

(IN) Points to the authentication necessary to establish a connection with the Target Service (for example, a user password).

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFC9	NWSMTS_OUT_OF_MEMORY
0xFFFDFFD7	NWSMTS_LOGIN_DENIED
0xFFFDFFF9	NWSMTS_CREATE_ERROR
0xFFFEFFF2	NWSMDR_INVALID_PROTOCOL
0xFFFEFFFD	NWSMDR_OUT_OF_MEMORY

0xFFFEFFE0	NWSMDR_CORRUPTED_STATE
0xFFFEFFE9	NWSMDR_INVALID_CONTEXT
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFDFFAE	NWSMTS_CLUSTER_TARGET_HAS_NO_VOLUMES

Remarks

Before NWSMTSConnectToTargetService is called, the engine must be connected to a TSA that has access to the desired target.

targetServiceName should contain a Target Service name from the name list returned by NWSMTSListTargetServices.

NWSMTSConnectToTargetService connects an engine to the specified Target Service and establishes the user's rights on the Target Service. After successfully connecting, the engine can access the Target Service's data.

An engine can connect to only one Target Service per TSA connection. This means that a Target Service can have multiple connections, but each connection must be referenced by a different TSA connection.

targetUserName contains the user name that allows proper access to the Target Service's data. For NetWare 3.x and earlier names, these are the bindery names. For NetWare 4.0 and later, these are directory names. For Linux file system TSA it is the user name of a user having an identity on the Linux system.

authentication is an unencrypted, length preceded (UINT16) string. However, the SMDR encrypt/decrypts it if it is passed between SMDRs. If no authentication is passed, set the length to zero. Do not pass a NULL pointer.

See Also

Example

```
#include <smstsapi.h>
#include <smsdrapi.h>

UINT32 connection;
STRING TSAName ;
NWSM_NAME_LIST *serviceNameList = NULL;
char  targetServiceName[NWSM_MAX_TARGET_SRVC_NAME_LEN],
      targetUserName[NWSM_MAX_TARGET_USER_NAME_LEN],
       authentication[30];

/* Connect to the TSA. */
NWSMConnectToTSA(TSAName, &connection);
```

```
/* Find Target Service through the connected TSA. */
sprintf(targetServiceName, "%s%s", TSAName, ".*");
NWSMTSListTargetServices(connection, targetServiceName ,
&serviceNameList);

/* Set up the authentication. */
sprintf(&authentication[2], "%s", "Password");
*((UINT16 *)&authentication[0]) = strlen(&authentication[2]);

/* Connect to the Target Service. */
NWSMTSConnectToTargetService(&connection, (STRING)serviceNameList-
>name, (STRING)targetUserName, authentication);
```

NWSMTSConnectToTargetServiceEx

Connects an engine to a specified Target Service and also supports passwords with international and extended characters.

Syntax

```
#include <smsutapi.h>
#include <smstsapi.h>

CCODE NWSMTSConnectToTargetServiceEx (
   UINT32  *connection,
   STRING   targetServiceName,
   STRING   targetUserName,
   void    *authentication
   UINT32   optionFlag
   );
```

Parameters

connection

> (IN/OUT) Specifies a pointer to the connection information from NWSMConnectToTSA. Outputs a pointer to the connection to the Target Service in addition to the input information.

targetServiceName

> (IN) Specifies the Target Service name returned by NWSMTSScanTargetServiceName or NWSMTSListTargetServices.

targetUserName

> (IN) Specifies the user name on a file server (maximum size is NWSM_MAX_TARGET_USER_NAME_LEN).

authentication

> (IN) Points to the authentication data necessary to establish a connection with the Target Service (for example, a user password).

optionflag

> (IN) Specifies the type of the authentication data.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFD7	NWSMTS_LOGIN_DENIED
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL

0xFFFDFFC8	NWSMTS_OVERFLOW
0xFFFEFFF2	NWSMDR_INVALID_PROTOCOL
0xFFFEFFE0	NWSMDR_CORRUPTED_STATE
0xFFFEFFE9	NWSMDR_INVALID_CONTEXT
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFDFFAD	NWSMTS_UNSUPPORTED_OPTION

Remarks

Before NWSMTSConnectToTargetService is called, the engine must be connected to a TSA that has access to the desired target.

targetServiceName should contain a Target Service name from the name list returned by NWSMTSListTargetServices.

NWSMTSConnectToTargetService connects an engine to the specified Target Service and establishes the user's rights on the Target Service. After successfully connecting, the engine can access the Target Service's data.

An engine can connect to only one Target Service per TSA connection. This means that a Target Service can have multiple connections, but each connection must be referenced by a different TSA connection.

targetUserName contains the user name that allows proper access to the Target Service's data.

authentication is defined using the *optionflag* parameter. However, the SMDR encrypt/decrypts it if it is passed between SMDRs. If no authentication is passed, set the length to zero. Do not pass a NULL pointer.

The following option flags are supported.

Option Flags	Description
NWSM_AUTH_UTF8_DATA	The authentication data is a length preceded (UINT16) UTF-8 string
NWSM_AUTH_UNICODE_DATA	The authentication data is a length preceded (UINT16) Unicode string.
NWSM_AUTH_RAW_DATA	The authentication data is a length preceded (UINT16) string.
	NOTE: This flag provides the same functionality as NWSM_AUTH_LOCAL_DATA and has been renamed to accurately reflect the data type. Any existing usage of NWSM_AUTH_LOCAL_DATA will continue to be supported.
NWSM_AUTH_CASA_TOKEN	The authentication data is a length preceded (UINT 16) CASA token and targetUserName should be set to "CasaPrincipal".

NOTE: This symbol is not present in the SMS import file, as all versions of SMS do not export this symbol. To use this function, programmatically import the symbol and invoke it.

See Also

NWSMTSGetTargetServiceType

Returns the Target Service type and version information.

Syntax

```
#include <smsutapi.h>
#include <smstsapi.h>

CCODE NWSMTSGetTargetServiceType (
   UINT32   connection,
   STRING   serviceName,
   STRING   serviceType,
   STRING   serviceVersion);
```

Parameters

connection

   (IN) Specifies the connection information returned by NWSMConnectToTSA.

serviceName

   (IN) Specifies the Target Service name returned by NWSMTSScanTargetServiceName or NWSMTSListTargetServices.

serviceType

   (OUT) Returns the Target Service type name (must be NWSM_MAX_TARGET_SRVC_TYPE_LEN bytes).

serviceVersion

   (OUT) Returns the version string of the Target Service (must be NWSM_MAX_TARGET_SRVC_VER_LEN bytes).

Return Values

See for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFDD	NWSMTS_INVALID_PARAMETER
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMTSGetTargetServiceType is called, the engine must be connected to the TSA and Target Service.

The buffer sizes include a NULL terminator.

Example

```
#include <smsutapi.h>
#include <smstsapi.h>
NWSM_NAME_LIST *serviceNameList = NULL;
unsigned char serviceTypeBuf[NWSM_MAX_TARGET_SRVC_TYPE_LEN],
serviceVersionBuf[NWSM_MAX_TARGET_SRVC_VER_LEN], pat[2] = "*";
STRING serviceType = (STRING)serviceTypeBuf, serviceVersion
=(STRING)serviceVersionBuf, serviceName, scanPattern = pat;

/* Connect to a TSA-see NWSMConnectToTSA. */

/* Get a list of Target Services accessible through the connected TSA
(some TSAs can service multiple Target Services).
NWSMTSListTargetServices or NWSMTSScanTargetServiceName can be used to
create this list. */
NWSMTSListTargetServices(connection, scanPattern, &serviceNameList);

/* See if we have more than one Target Service */
if (serviceNameList->next)
{
/* Have the user choose a Target Service, set serviceName to point to
that name. */
}
else
serviceName = serviceNameList->name;
NWSMTSGetTargetServiceType(connection, serviceName, serviceType,
serviceVersion);
```

NWSMTSListTargetServices

Returns a list of all Target Services accessible through the connected TSA.

Syntax

```
#include <smstsapi.h>
#include <smsutapi.h>

CCODE NWSMTSListTargetServices (
   UINT32               connection,
   STRING               scanPattern,
   NWSM_NAME_LIST **serviceNameList);
```

Parameters

connection

   (IN) Specifies the connection information returned by NWSMConnectToTSA.

scanPattern

   (IN) Specifies the Target Service names pattern to search for.

serviceNameList

   (IN/OUT) Points to the list of Target Service names to be returned.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFD	NWSMDR_OUT_OF_MEMORY
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMTSListTargetServices is called, the engine must be connected to a TSA.

If there are no names to return, NWSMTSListTargetServices returns successfully and serviceNameList is set to NULL.

The values that can be set for scanPattern is listed in the following table.

Value	Description
*smdrName.tsaName	Return all TSAs named tsaName on Target Services that have names that end with smdrName.
smdrName.tsaName*	Return all TSAs that begin with tsaName on Target Service smdrName.
*	Return all available TSAs.
.	Return all available TSAs.

NOTE: NWSMTSListTargetServices is implemented by calling NWSMTSScanTargetServiceName.

When connection is a handle to a TSA that services only one Target Service, NWSMTSListTargetServices returns a list of one entry.

TIP: The returned list does not describe the type of Target Service (such as NetWare or Linux). This additional information is returned by NWSMTSGetTargetServiceType.

See Also

Example

```
#include <smstsapi.h>
#include <smsdrapi.h>

NWSM_NAME_LIST *serviceNameList = NULL;
/* Connect to a TSA-see NWSMConnectToTSA. */
/* Get a list of Target Services accessible through the connected TSA
(some TSAs  can service multiple Target Services).
NWSMTSListTargetServices or NWSMTSScanTargetServiceName can be used to
create this list. */
ccode = NWSMTSListTargetServices(connection, "*", &serviceNameList);
/* See if we have more than one Target Service */
if (serviceNameList->next)
{
/* Have the user choose a Target Service. */
}
/* Gather the optional information about the Target Service-see
NWSMTSGetTargetServiceType. */
/* Get the user's name and password, and connect to the selected Target
Services NWSMTSConnectToTargetService */
```

NWSMTSReleaseTargetService

Closes the connection between an engine and a Target Service.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSReleaseTargetService (
   UINT32  *connection);
```

Parameters

connection

> (IN) Points to the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServicEx.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFD6	NWSMTS_LOGOUT_ERROR
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMTSReleaseTargetService is called, the engine must be connected to the TSA.

NWSMTSReleaseTargetService cleans up the internal environment, releases a connection between the engine and the Target Service, and deletes the error log and skipped log files created by the TSA.

See Also

NWSMTSConnectToTargetService (page 65), NWSMTSConnectToTargetServiceEx (page 68)

NWSMReleaseTSA

Closes the connection between a TSA and an engine.

Syntax

```
#include <smstsapi.h>

CCODE NWSMReleaseTSA (
   UINT32  *connection);
```

Parameters

connection

   (IN) Points to the connection information returned by NWSMConnectToTSA.

Return Values

See for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFEFFE0	NWSMDR_CORRUPTED_STATE
0xFFFEFFE9	NWSMDR_INVALID_CONTEXT
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMReleaseTSA is called, the Target Service must be released.

There is only one engine/TSA pair for each session so it is not necessary to specify the TSA when releasing the connection. connection is set to an invalid value after the connection is released.

See Also

NWSMTSScanTargetServiceName

Retrieve a Target Service name from a connected TSA.

Syntax

```
#include <smsutapi.h>
#include <smstsapi.h>

CCODE NWSMTSScanTargetServiceName (
   UINT32   connection,
   UINT32  *tsnSequence,
   STRING   scanPattern,
   STRING   serviceName);
```

Parameters

connection

  (IN) Specifies the connection information returned by NWSMConnectToTSA.

tsnSequence

  (IN/OUT) Points to the Target Service name sequence number.

scanPattern

  (IN) Specifies the pattern to search for.

serviceName

  (OUT) Specifies the Target Service name.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFD2	NWSMTS_NO_MORE_DATA
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER

Remarks

Before NWSMTSScanTargetServiceName is called, the engine must be connected to a TSA.

A TSA can service only one Target Service name or multiple Target Services.
NWSMTSScanTargetServiceName must be called repeatedly to retrieve all specified Target Service

names. When provided with supporting code, NWSMTSScanTargetServiceName is the same as NWSMTSListTargetServices.

The engine sets tsnSequence to zero before starting each scanning session. If NWSMTSScanTargetServiceName returns an error, tsnSequence is set to 0xFFFFFFFF.

The values that can be set for scanPattern is listed in the following table.

Value	Description
"*"	Return all names
"*xxxx"	Return all names that end with "xxxx" [a]
"xxxx*"	Return all names that begin with "xxxx"[a]
"xxxx"	Find name "xxxx"[a]
a. Where "xxxx" is one or more characters.	

The engine passes a buffer to serviceName that is NWSM_MAX_TARGET_SRVC_NAME_LEN bytes long. A NULL string is returned if no Target Service is found.

See Also

NWSMTSConnectToTargetService (page 65), NWSMTSConnectToTargetServiceEx (page 68), NWSMTSGetTargetServiceType (page 71), NWSMTSListTargetServices (page 73)

Example

```
#include <smsutapi.h>
#include <smstsapi.h>

UINT32 tsnSequence = 0;
unsigned char pat[2] = "*", sName[NWSM_MAX_TARGET_SRVC_NAME_LEN];
NWSM_NAME_LIST *serviceNameList = NULL, *name, *last;
STRING pattern = pat, serviceName = sName;

/* Connect to a TSA-see NWSMConnectToTSA. */

/* Build the Target Service name list (some TSAs can service multiple
Target Services).
NWSMTSListTargetServices can be used to create this list. */
while (NWSMTSScanTargetServiceName(connection, &tsnSequence, pattern,
     serviceName) == 0)
{
   name = (NWSM_NAME_LIST *)calloc(1, sizeof(NWSM_NAME_LIST));
   name->name = (STRING)calloc(1, NWSM_MAX_TARGET_SRVC_NAME_LEN);
   strcpy(name->name, serviceName);
   if (serviceNameList)
   {
      last->next = name;
      last = last->next;
}
```

```
   else
        serviceNameList = last = name;
} /* End while */

/* See if we have more than one Target Service */
if (serviceNameList->next)
{
    /* Have the user choose a Target Service. */
}

/* Gather the optional information about the Target Service -
see NWSMTSGetTargetServiceType. */

/* Connect to the selected Target Service-see
NWSMTSConnectTargetService*/
```

3.3 Miscellaneous Functions

The following functions perform miscellaneous TSA tasks:

NWSMTSCatDataSetName

Joins two paths together.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSCatDataSetName (
   UINT32          connection,
   UINT32          nameSpaceType,
   STRING          dataSetName,
   STRING          terminalName,
   NWBOOLEAN       terminalNameIsParent,
   STRING_BUFFER **newDataSetName);
```

Parameters

connection

  (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServiceEx.

nameSpaceType

  (IN) Specifies the name space type of the names to concatenate (see "nameSpaceType Values" on page 347).

dataSetName

  (IN) Specifies the data set name to append to.

terminalName

  (IN) Specifies the data set's name (parent or child name only) to append to dataSetName.

terminalNameIsParent

  (IN) Specifies if terminalName is a parent or child:

  TRUE Parameter specifies a parent
  FALSE Parameter specifies a child

newDataSetName

  (OUT) Points to the concatenated path.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION

| 0xFFFDFFE7 | NWSMTS_INVALID_CONNECTION_HANDL |
| 0xFFFEFFFF | NWSMDR_INVALID_CONNECTION |

Remarks

NWSMTSCatDataSetName appends a terminal name onto an existing path. If terminalName is a parent, the appropriate delimiter for that namespace is appended to the end of the string. If terminalName is a parent, and if the Target Service requires it, a separator is placed at the end of newDataSetName.

newDataSetName must point to a valid structure or NULL. NWSMTSCatDataSetName allocates memory if NULL is passed, or allocates a larger memory block if the structure does not have enough space.

The engine should call NWSMFreeString to deallocate newDataSetName .

NWSMTSCloseDataSet

Closes a data set opened by NWSMTSOpenDataSetForBackup or
NWSMTSOpenDataSetForRestore.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSCloseDataSet (
   UINT32   connection,
   UINT32  *dataSetHandle);
```

Parameters

connection

> (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or
> NWSMTSConnectToTargetServicEx.

dataSetHandle

> (IN/OUT) Points to the address of the data set handle returned by
> NWSMTSOpenDataSetForBackup or NWSMTSOpenDataSetForRestore.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFAF	NWSMTS_CLUSTER_TARGET_DOES_NOT_EXIST
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFE5	NWSMTS_INVALID_DATA_SET_HANDLE
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Upon return, NWSMTSCloseDataSet sets dataSetHandle to zero. If parent handles are used and the
parent's subordinates are being restored, the engine should not call NWSMTSCloseDataSet until all
subordinates are restored.

NWSMTSCloseDataSet restores all of the data set's original attributes to what they were before the
data set was opened for backup (except its archive information). To set the data set's archive status,
the engine should call NWSMTSSetArchiveStatus before closing the data set.

Between the time NWSMTSOpenDataSetForBackup and NWSMTSCloseDataSet are called, the TSA ensures that the data set's attributes are not altered.

See Also

NWSMTSOpenDataSetForBackup (page 49), NWSMTSOpenDataSetForRestore (page 146), NWSMTSSetArchiveStatus (page 95)

Example

See NWSMTSOpenDataSetForRestore (page 146) for an example.

NWSMTSDeleteDataSet

Deletes a specified data set.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSDeleteDataSet (
   UINT32   connection,
   UINT32   sequence);
```

Parameters

connection

(IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServicEx.

sequence

(IN) Specifies the sequence number returned by NWSMTSScanDataSetBegin or NWSMTSScanNextDataSet.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFAF	NWSMTS_CLUSTER_TARGET_DOES_NOT_EXIST
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFD8	NWSMTS_INVALID_SEQUENCE_NUMBER
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFDFFF1	NWSMTS_DELETE_ERR
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMTSDeleteDataSet is called, the scanning function must return a valid sequence number.

All children must be deleted before NWSMTSDeleteDataSet can delete their parent.

See Also

NWSMTSScanDataSetBegin (page 38), NWSMTSScanNextDataSet (page 53)

NWSMTSFixDataSetName

Formats a fully qualified parent directory path according to the specified name space specifications.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSFixDataSetName (
    UINT32          connection,
    STRING          dataSetName,
    UINT32          nameSpaceType,
    NWBOOLEAN       isParent,
    NWBOOLEAN       wildAllowedOnTerminal,
    STRING_BUFFER **newDataSetName);
```

Parameters

connection

> (IN) Specifies the connection information returned by NWSMConnectToTSA.

dataSetName

> (IN/OUT) Specifies the path to be fixed.

nameSpaceType

> (IN) Specifies the name space type of *dataSetName*. (see "nameSpaceType Values" on page 347)

isParent

> (IN) Specifies the type of path contained in dataSetName:

> TRUE Specifies a fully qualified parent path
> FALSE Specifies a fully qualified child path or a terminal name

wildAllowedOnTerminal

> (IN) Specifies if the path contains wild cards only in the terminal path node:

> TRUE Wildcards allowed only in the terminal path
> FALSE Wildcards allowed elsewhere in the path

newDataSetName

> (OUT) Points to the buffer to receive the fixed up path.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful

0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFDFFE0	NWSMTS_INVALID_NAME_SPACE_TYPE
0xFFFDFFC9	NWSMTS_OUT_OF_MEMORY
0xFFFDFFDC	NWSMTS_INVALID_PATH
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER

Remarks

If dataSetName contains a fully qualified path, NWSMTSFixDataSetName ensures that the path has a separator and proper case. NWSMTSFixDataSetName also adds a separator to the terminal path node if isParent is set to TRUE. If the *nameSpaceType* defines a first separator and the path does not contain a valid separator, NWSMTSFixDataSetName returns NWSMTS_INVALID_PATH.

The fixed up path is returned in newDataSetName. NWSMTSFixDataSetName copies dataSetName to newDataSetName and performs the following fix ups and checks on the path:

- Converts the appropriate path nodes to upper case as follows:
  - DOS paths are converted to upper case
  - All primary resources before first separator (if supported by the *nameSpaceType* on the target service) in NFS, FTAM, and OS/2 are converted to upper case.
- For all non-Macintosh paths the following fixes and checks are performed:
  - All backslashes (\) are converted to forward slashes (/), except for NFSNamespace.
  - Ensures only the terminal path node contains wild cards.
  - If isParent is set to TRUE and dataSetName does not end with first separator (if supported by the *nameSpaceType* on the target service), backslash (\), or forwardslash (/), a forwardslash (/) is added to the end of the path.
  - If the path contains ":/" or :\," the slash is removed from the path, if the first separator for the *nameSpaceType* is a " : ".
- For all Macintosh paths the following checks and fixes are performed:
  - If isParent is TRUE and dataSetName does not end with a colon, a colon is added to the end of the path.
  - If isParent is TRUE, dataSetName is checked to ensure it contains a colon or double colon.

The values that can be set for *nameSpaceType* is listed in the following table.

Value	Description
0x000	DOSNameSpace
0x001	MACNameSpace
0x002	NFSNameSpace
0x003	FTAMNameSpace

Value	Description
0x004	OS2NameSpace
0x100	DOSNameSpaceUtf8Type
0x101	MACNameSpaceUtf8Type
0x102	NFSNameSpaceUtf8Type
0x104	LONGNameSpaceUtf8Type

The following table lists the wild cards options that can be used in the terminal path node.

Value	Option	Description
0x2A	ASTERISK	Regular asterisk
0x3F	QUESTION	Regular question mark
0xAE	SPERIOD	Special Period-the most significant bit set.
0xAA	SASTERISK	Special Asterisk-the most significant bit set.
0xBF	SQUESTION	Special Question-with the most significant bit set.

If NWSMTSFixDataSetName encounters an error as it tries to fix the path, a NULL string is returned in newDataSetName.

NWSMTS_INVALID_PATH is returned if dataSetname meets one or more of the following conditions:

- dataSetName is a parent and does not contain valid first separator, if the *nameSpaceType* supports a first separator on the target service..
- Non-Macintosh path nodes, other than the terminal path node, contain wild cards.
- wildAllowedOnTerminal is set to FALSE and non-Macintosh paths contain wild cards.
- The path is improperly formatted.

The engine is responsible for freeing the memory held by newDataSetName. To free this buffer, call NWSMFreeString. (See *Storage Management Services Utility Library*).

NOTE: NWSMTSFixDataSetName will eventually replace the utility function NWSMFixDirectoryPath (see *Storage Management Services Utility Library*).

NWSMTSParseDataSetName

Returns the number of path nodes and separators, and the beginning position of each path node and separator for a specified path.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSParseDataSetName (
    UINT32          connection,
    UINT32          nameSpaceType,
    STRING          dataSetName,
    UINT16          *count,
    UINT16_BUFFER **namePositions,
    UINT16_BUFFER **separatorPositions);
```

Parameters

connection

(IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServicEx.

nameSpaceType

(IN) Specifies the name space type of *dataSetName* (see "nameSpaceType Values" on page 347)

dataSetName

(IN) Specifies the data set name to be parsed.

count

(OUT) Points to the number of nodes and separators in dataSetName (the size of the namePositions and separatorPositions buffers).

namePositions

(OUT) Points to an array of indexes containing the beginning of each node in dataSetName.

separatorPositions

(OUT) Points to an array of indexes containing the beginning of each separator in dataSetName.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFB	NWSMUT_OUT_OF_MEMORY

0xFFFBFFFC	NWSMUT_NO_MORE_NAMES
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER
0xFFFBFFFF	NWSMUT_INVALID_HANDLE
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFE0	NWSMTS_INVALID_NAME_SPACE_TYPE
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

NWSMTSParseDataSetName parses a data set name and returns the number of nodes and separators and a list of indexes to each node and separator (see Path Information).

NWSMTSParseDataSetName can be called when the engine needs to pass a path to the NWSM_DATA_SET_NAME_LIST or NWSM_SELECTION_LIST structure.

NWSMTSRenameDataSet

Renames an existing child data set on the Target Service.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSRenameDataSet (
   UINT32   connection,
   UINT32   sequence,
   UINT32   nameSpaceType,
   STRING   newDataSetName);
```

Parameters

connection

> (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServicEx.

sequence

> (IN) Specifies the sequence value returned by NWSMTSScanDataSetBegin or NWSMTSScanNextDataSet.

nameSpaceType

> (IN) Specifies the name space type of *newDataSetName* (see "nameSpaceType Values" on page 347).

newDataSetName

> (IN) Specifies the data set's new name (cannot be NULL).

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFD8	NWSMTS_INVALID_SEQUENCE_NUMBER
0xFFFDFFDD	NWSMTS_INVALID_PARAMETER
0xFFFDFFEB	NWSMTS_GET_NAME_SPACE_ENTRY_ERR
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMTSRenameDataSet is called, NWSMTSScanDataSetBegin or NWSMTSScanNextDataSet must be called to return a valid sequence number.

NWSMTSRenameDataSet cannot relocate the data set to another directory or logical location. The engine can only move data sets during the restore session (see NWSMTSOpenDataSetForRestore (page 146)).

NWSMTSRenameDataSet may not apply to all Target Services because some services may not have a file system or the ability to rename a data set.

To determine if a TSA supports the ability to rename data sets, call NWSMTSGetUnsupportedOptions (page 127).

newDataSetName can contain any path information. Do not pass a NULL pointer.

nameSpaceType can be retrieved from the NWSM_DATA_SET_NAME_LIST or NWSM_SCAN_INFORMATION structure returned by NWSMTSScanDataSetBegin or NWSMTSScanNextDataSet.

See Also

NWSMTSScanDataSetBegin (page 38), NWSMTSScanNextDataSet (page 53)

NWSMTSReturnToParent

Stops the current scan and continues the scan on the next qualified parent.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSReturnToParent (
   UINT32   connection,
   UINT32  *sequence);
```

Parameters

connection

> (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServicEx.

sequence

> (IN) Points to the sequence number returned by NWSMTSScanDataSetBegin or NWSMTSScanNextDataSet.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFD8	NWSMTS_INVALID_SEQUENCE_NUMBER
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFDFFF3	NWSMTS_DATA_SET_IS_OPEN
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER

Remarks

Before NWSMTSReturnToParent is called, NWSMTSScanDataSetBegin must be called to initiate a scan.

When NWSMTSScanNextDataSet is called, the next parent is scanned.

See Also

NWSMTSScanNextDataSet (page 53)

NWSMTSSeparateDataSetName

Separates the terminal path node from the rest of the path.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSSeparateDataSetName (
   UINT32          connection,
   UINT32          nameSpaceType,
   STRING          dataSetName,
   STRING_BUFFER **parentDataSetName,
   STRING_BUFFER **terminalNode);
```

Parameters

connection

(IN) Specifies the connection information returned by NWSMConnectToTargetService.

nameSpaceType

(IN) Specifies the name space type of *dataSetName* (see "nameSpaceType Values" on page 347).

dataSetName

(IN) Specifies the data set name to be separated.

parentDataSetName

(OUT) Points to dataSetName less the terminal node (optional).

terminalNode

(OUT) Points to the terminal node of dataSetName (optional).

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFC9	NWSMTS_OUT_OF_MEMORY
0xFFFDFFDC	NWSMTS_INVALID_PATH
0xFFFDFFE0	NWSMTS_INVALID_NAME_SPACE_TYPE
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER

0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

NWSMTSSeparateDateSetName allocates memory if a NULL pointer is passed or if the structure does not have enough space. To free parentDataSetName and terminalNode, call NWSMFreeString (see *Storage Management Services Utilities Library* for more information).

For name spaces that have reversed path nodes such as NDS, the terminal node is the left-most path node. For name spaces that have unreversed paths such as DOS, the terminal node is the right-most path node.

NWSMTSSetArchiveStatus

Sets or restores the data set's archived status and other attributes.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSSetArchiveStatus (
   UINT32   connection,
   UINT32   dataSetHandle,
   UINT32   setFlag,
   UINT32   archivedDateAndTime);
```

Parameters

connection

   (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServicEx.

dataSetHandle

   (IN) Specifies the data set handle returned by NWSMTSOpenDataSetForBackup.

setFlag

   (IN) Specifies the archive information to set.

archivedDateAndTime

   (IN) Specifies the information to set as the data set's archived date and time.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFAF	NWSMTS_CLUSTER_TARGET_DOES_NOT_EXIST
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFBD	NWSMTS_SET_FILE_INFO_ERR
0xFFFDFFCE	NWSMTS_NO_SUCH_PROPERTY
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMTSSetArchiveStatus is called, the data set must be open.

NWSMTSSetArchiveStatus should be called before closing the data set.

Check if the resource supports archive attributes using NWSMTSGetUnsupportedOptions and NWSMTSGetTargetResourceInfoEx before invoking the API.

NOTE: Calling NWSMTSOpenDataSetForBackup or NWSMTSReadDataSet does not alter the access date and time.

---

setFlag can be set to zero or to one of the flags listed in the following table.

Value	Description
0x0000	NWSM_SET_MODIFY_FLAG: Sets the modify flag.
0x0001	NWSM_CLEAR_MODIFY_FLAG: Clears the modify flag. For TSAs, the data modified flag and characteristics modified flags are cleared.
0x0002	NWSM_SET_ARCHIVE_DATE_AND_TIME: Sets the data set's archive date and time.
0x0004	NWSM_SET_ARCHIVER_ID: Sets the archiver's ID into the data set.

archivedDateAndTime is a DOS packed date and time value (see "DOS Date and Time Functions" in *Storage Management Services Utilities Library*).

3.4 Option Functions

The following functions gather the TSA options:

NWSMTSBuildResourceList

Updates the TSA's internal list of primary resources.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSBuildResourceList (
   UINT32   connection);
```

Parameters

connection

> (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServicEx.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFC9	NWSMTS_OUT_OF_MEMORY
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFDFFE8	NWSMTS_GET_VOL_NAME_SPACE_ERR
0xFFFDFFE9	NWSMTS_GET_SERVER_INFO_ERR
0xFFFDFFED	NWSMTS_GET_DATA_STREAM_NAME_ERR
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMTSBuildResourceList is called, the engine must be connected to a TSA and Target Service.

The engine calls NWSMTSBuildResourceList to ensure the TSA's resource list is up-to-date. The TSA will only scan what is in its internal primary resource list. If a virtual NCP server name is selected, the list of primary resources in the pool associated with the virtual NCP server is built. If a node name is selected, the list of non-cluster resources on the server will be built.

Do not call NWSMTSBuildResourceList if there are any active scans or if NWSMTSOpenDataSetForRestore still has data sets open.

See Also

NWSMTSConfigureTargetService

Configures the target service for a variety of tasks.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSConfigureTargetService (
   UINT32  connection
   UINT32  actionFlag,
   void    *actionData)
```

Parameters

connection

(IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServiceEx.

actionFlag

(IN) Specifies the appropriate action to be taken.

actionData

(IN) Specifies the data relevant to the *actionFlag*.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFFF	NWSMDR_INVALID_CONNECTION
0xFFFDFFFAD	NWSMTS_UNSUPPORTED_OPTION

Remarks

The following table describes the action flags and the associated action data.

Action Flag	Description	Data Type	Action Data	Supported By
NWSM_FLUSH_LOGFILES	Deletes the contents of the skipped data sets and error log files	UINT32 *	Set the value of the unsigned integer to 1.	TSAFS

Action Flag	Description	Data Type	Action Data	Supported By
NWSM_CONFIGURE_SKIPPED _LOG	By default, skipped data sets log file is created in the directory, sys:\system\tsa for NetWare and /var/opt/ novell/sms for Linux . You can override it by supplying the name along with the full path. The file is automatically deleted when NWSMTSReleaseTarge tService is called.	STRING	Set the value of the string to the full path name of the file.	TSAFS
NWSM_CONFIGURE_ERROR_ LOG	By default, error log file is created in the directory, sys:\system\tsa for NetWare and /var/opt/ novell/sms for Linux . You can override this by supplying the full path of the file name. The file is automatically deleted when NWSMTSReleaseTarge tService is called.	STRING	Set the value of the string to the full path name of the file.	TSAFS
NWSM_DONOT_IGNORE_ BACKUP_BIT	Does not ignore the zATTR_DONT_BACKU P attribute of NSS file system and considers a primary resource for backup only if this attribute is not set. By default, the zATTR_DONT_BACKU P attribute is ignored and all resources are considered as eligible for backup.	UINT32 *	Set the value of the unsigned integer to 1.	TSAFS
NWSM_USE_CACHING_MODE	Sets the TSAFS caching mode. TSAFS will operate on caching mode if this is set to true and operate on serial mode if is false. By default, caching mode is enabled.	UINT32 *	Set the value of the unsigned integer to 1 or 0.	TSAFS

NOTE: This symbol is not present in the SMS import file, as all versions of SMS do not export this symbol. To use this function, programmatically import the symbol and invoke it.

See Also

Example

```
#include <smstsapi.h>
#include <smsdrapi.h>
UINT32 connection;
STRING tsaName;
/* connection related code goes here*/
/* After the connection:
 * if it is a file system TSA,
 * 1. set the NWSM_DONOT_IGNORE_BACKUP_BIT for this connection
 * 2. rename skipped log file to BIGVOL:SKIPDATA.LOG */

if(strstr(tsaName, "NetWare File System") || strstr(tsaName, "NetWare
Cluster File System"))
{
     UINT32 doNotBackupBit;
     doNotBackupBit = 1;
     cCode = NWSMTSConfigureTargetService (connection,
NWSM_DONOT_IGNORE_BACKUP_BIT, &doNotBackupBit );
     if (cCode)
          {
               //do error handling
          }
     cCode = NWSMTSConfigureTargetService (connection,
NWSM_CONFIGURE_SKIPPED_LOG,
     "BIGVOL:SKIPDATA.LOG");
     if (cCode)
          {
               //do error handling
          }
}
```

NWSMTSGetNameSpaceTypeInfo

Returns the path node order and separators for the specified name space.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSGetNameSpaceTypeInfo (
   UINT32          connection,
   UINT32           nameSpaceType,
   NWBOOLEAN       *reverseOrder,
   STRING_BUFFER **firstSeparator,
   STRING_BUFFER **secondSeparator);
```

Parameters

connection

   (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServicEx.

nameSpaceType

   (IN) Specifies the name space type of the *name* (see "nameSpaceType Values" on page 347)

reverseOrder

   (OUT) Points to a flag indicating if reverse order should be used:

   TRUE Subordinates are on the left
   FALSE Subordinates are on the right

firstSeparator

   (OUT) Points to the first separator string.

secondSeparator

   (OUT) Points to the second separator string.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFC9	NWSMTS_OUT_OF_MEMORY
0xFFFDFFE0	NWSMTS_INVALID_NAME_SPACE_TYPE
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL

| 0xFFFEFFFE | NWSMDR_INVALID_PARAMETER |
| 0xFFFEFFFF | NWSMDR_INVALID_CONNECTION |

Remarks

Before NWSMTSGetNameSpaceTypeInfo is called, the engine must be connected to a TSA and Target Service.

The information returned by NWSMTSGetNameSpaceTypeInfo can be used to build lists for NWSM_DATA_SET_NAME_LIST and NWSM_SELECTION_LIST.

If the path order is reversed, the subordinates are to the left of their superiors. For example, "Employee.Department.Company.Country" is a reversed path.

firstSeparator and secondSeparator must point to a valid non-autovariable structure or NULL. NWSMTSGetNameSpaceTypeInfo allocates memory when a NULL is passed, or reallocates memory if the structure does not have enough space. To free the memory for the separators, call NWSMFreeString (see *Storage Management Services Library*).

Example

See for an example.

NWSMTSGetOpenModeOptionString

Returns TSA-specific open mode option strings.

Syntax

```
#include <smsutapi.h>
#include <smstsapi.h>

CCODE NWSMTSGetOpenModeOptionString (
   UINT32   connection,
   UINT8    optionNumber,
   STRING   optionString);
```

Parameters

connection

   (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or
   NWSMTSConnectToTargetServicEx.

optionNumber

   (IN) Specifies the open mode option number (0- 23) to return.

optionString

   (OUT) Specifies the string that describes the option specified by optionNumber (must be
   NWSM_MAX_STRING_LEN bytes).

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFCB	NWSMTS_OPEN_MODE_TYPE_NOT_USED
0xFFFDFFDE	NWSMTS_INVALID_OPEN_MODE_TYPE
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMTSGetOpenModeOptionString is called, the engine must be connected to a TSA and
Target Service.

The options returned by NWSMTSGetOpenModeOptionString allow the engine to open different
parts of the data set on a per data set basis (see Backup Open Mode Options List). For example, the

data set's attributes or trustee information. The engine applies the open mode options on an individual basis when NWSMTSWriteDataSet is called.

optionNumber represents the bit position within the TSA-specific open mode option bit map. This value is used by NWSMTSOpenDataSetForBackup and NWSMTSOpenDataSetForRestore.

The following table lists the open mode options.

Option	Value	Description
0	0x00000100	NWSM_NO_DATA_STREAMS: Do not read/write the data set's data streams (e.g., file data).
1	0x00000200	NWSM_NO_EXTENDED_ATTRIBUTES: Do not read/write the data set's extended attributes.
2	0x00000400	NWSM_NO_PARENT_TRUSTEES: Do not read/write the parent's trustees.
3	0x00000800	NWSM_NO_CHILD_TRUSTEES: Do not read/write the child's trustees.
4	0x00001000	NWSM_NO_VOLUME_RESTRICTIONS: Do not read/write the resource's restrictions.
5	0x00002000	NWSM_NO_DISK_SPACE_RESTRICTIONS: Do not read/write the resource's space restrictions.
6	0x00004000	NWSM_INCLUDE_MIGRATED_DATA: Restore migrated data (the migration key is ignored).
7	0x00008000	NWSM_DELETE_EXISTING_TRUSTEES: Delete all trustees of a data set before restoring the data sets.
8	0x00010000	NWSM_EXPAND_COMPRESSED_DATA_SET: Expand data sets that are currently compressed on the host (backup only).
9	0x00020000	NWSM_EXCLUDE_MIGRATED_DATA: Do not backup migrated data, however do backup its migration key and directory entries.
10	0x00040000	NWSM_PRESERVE_ACCESS_TIME : Preserves the access time of the data set after a backup session.
11	0x00080000	NWSM_NO_HARDLINK_DATA : Do not backup the data of consecutive hard link nodes of a particular hardlink network, except for the first encountered node.

If a TSA does not support a TSA-specific open mode option, optionString returns NULL and NWSMTSGetOpenModeOptionString returns successfully (see Backup Open Mode Options List).

The order of the returned strings corresponds to the order of the following bit map starting from bit 8-string one represents bit zero while string two represents bit one, etc.:

Call NWSMTSGetOpenModeOptionString repeatedly until all TSA-specific open mode strings are returned. optionNumber is converted to a bit mask and is used to index a string table.

See Also

Example

```
#include <smsutapi.h>
#include <smstsapi.h>

typedef struct TSA_SPECIFIC_OPEN_MODE
{
   NWBOOLEAN selected;
   UINT32 openMode;
   char openModeString[NWSM_MAX_STRING_LEN];
   struct TSA_SPECIFIC_OPEN_MODE *next;
} TSA_SPECIFIC_OPEN_MODE;

UINT8 optionNumber = 0;
char optionString[NWSM_MAX_STRING_LEN];
TSA_SPECIFIC_OPEN_MODE *TSAOpenModes = NULL, *last, *tmp;
UINT32 openMode, chosenOpenModes;

for (optionNumber = 0, openMode = 0x100; optionNumber <= 23;
optionNumber++,
      openMode <<= 1)
{
   if (NWSMTSGetOpenModeOptionString(connection, optionNumber,
      (STRING)optionString) != 0)
   break;

   if (!*optionString)
      continue;

   TMp = (TSA_SPECIFIC_OPEN_MODE *)calloc(1,
sizeof(TSA_SPECIFIC_OPEN_MODE));
   TMp->openMode = openMode;
   strcpy(tmp->openModeString, optionString);

   if (TSAOpenModes)
   {
      last->next = TMp;
      last = TMp;
   }

   else
      TSAOpenModes = last = TMp;
 }

/* Create the strings for the generic open modes-see "Backup Open Mode
Options List" and the "Remarks" for more
information */

/* Display the open mode options to the user, get the user's selections
and set it into chosenOpenModes, then pass
chosenOpenModes to NWSMTSOpenDataSetForBackup or
NWSMTSOpenDataSetForRestore. */
```

The following is a simplified example of returning the TSA-specific open mode strings.

```
/* optionNumber is initialized to zero to indicate to start from the
first open mode option string.
It is then incremented from 0 through 23 to get all the strings.
openMode contains the open mode bit mask that represents the open mode
option.
It is initialized to 0x1000 because the first TSA-specific option
starts at bit 8 of the open mode bit map. */

for (optionNumber = 0, openMode = 0x100; optionNumber <= 23;
optionNumber++, openMode <<= 1)
{
   if (NWSMTSGetOpenModeOptionString(connection, optionNumber,
         (STRING)optionString) != 0)
      break;

   if (!*optionString)
      continue;
/* Continue takes care of any TSA-specific open mode options that may
not be supported. */
   /* Copy openMode and optionString to a link list used to build the
remaining portion of the open modes option list. */

   ...
}
```

NWSMTSGetSupportedNameTypes

Returns the name types supported by the target service

Syntax

```
#include <smsutapi.h>
#include <smstsapi.h>

CCODE NWSMTSGetSupportedNameTypes(
   UINT32   connection,
   UINT32  *nameTypes);
```

Parameters

connection

(IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServiceEx.

nameTypes

(OUT) Specifies the name types supported by the target service.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

The nameType parameter contains a bit map of all name types that are supported by the target service.

The following table shows the *nameTypes* values.

Return Values	Description	Supported By
0x01	NWSM_NAME_TYPE_MBCS:Characters are represented in Multi Byte Character Set (MBCS)format.	TSAFS

Return Values	Description	Supported By
0x02	NWSM_NAME_TYPE_UTF8: Characters are represented in UTF-8 format.	TSAFS

NOTE: This symbol is not present in the SMS import file, as all versions of SMS do not export this symbol. To use this function, programmatically import the symbol and invoke it.

Example

```
#include <smsutapi.h>
#include <smstsapi.h>
#define ADDITIONAL_BUFFER_SIZE 512
NWSM_SCAN_CONTROL *scanControl;
UINT32connection, nameTypes;
/* NWSMTSConnectToTargetService/NWSMTSConnectToTargetServiceEx is used
to initialize the connection parameter used
in the example below */
/* Backing up using UTF-8 data set names */
/* Setup the scan control structure. Note that some fields in
scanControl are set to their default values
when the memory is calloc'd. */
scanControl = (NWSM_SCAN_CONTROL *)calloc(1, sizeof(NWSM_SCAN_CONTROL)
+ ADDITIONAL_BUFFER_SIZE);
scanControl->bufferSize = sizeof(NWSM_SCAN_CONTROL) +
ADDITIONAL_BUFFER_SIZE;
scanControl->scanControlSize = sizeof(NWSM_SCAN_CONTROL);
scanControl->otherInformationSize = 0;
/* Set other scanControl parameters to defaults or chosen values */
/* Get the supported name types from the connected target service */
cCode = NWSMTSGetSupportedNameTypes(connection, &nameTypes);
if (!cCode && (nameTypes & NWSM_NAME_TYPE_UTF8))
{
/* Set the name space type that the TSA should return to UTF8 format,
this would return the names in
NWSM_DATA_SET_NAME_LIST in UTF8 format */
scanControl->returnNameSpaceType = NWSM_ALL_NAME_SPACES_UTF8;
/* NOTE: If engines want all names in both UTF-8 and MBCS set the above
to NWSM_ALL_NAME_SPACES_FORMATS instead of NWSM_ALL_NAME_SPACES, as
the latter will return only the MBCS name space types to maintain
backward compatability */
/* NOTE: Some engines may scan in only a particular name space like
OS2NameSpace, in such cases pass in the returnNameSpaceType as
LONGNameSpaceUtf8Type */
}
else
{
/* If there are any errors, or the API is unsupported, use the older
mechanisim of backing up names */
scanControl->returnNameSpaceType = NWSM_ALL_NAME_SPACES;
}
```

```
/* Start the backup of data sets by calling NWSMTSScanDataSetBegin with
the above scanControl.
Refer to NWSMTSScanNextDataSet (page 53),NWSMTSOpenDataSetForBackup
(page 49), and NWSMTSReadDataSet (page 51) for further details on how
to backup data sets */
```

NWSMTSGetTargetResourceInfo

Returns information about a primary resource.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSGetTargetResourceInfo (
   UINT32     connection,
   STRING     resourceName,
   UINT16    *blockSize,
   UINT32    *totalBlocks,
   UINT32    *freeBlocks,
   NWBOOLEAN *resourceIsRemovable,
   UINT32    *purgeableBlocks,
   UINT32    *notYetPurgeableBlocks,
   UINT32    *migratedSectors,
   UINT32    *precompressedSectors,
   UINT32    *compressedSectors);
```

Parameters

connection

   (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServicEx.

resourceName

   (IN) Specifies the resource name returned by NWSMTSScanTargetServiceResource or NWSMTSListTSResources.

blockSize

   (OUT) Points to the resource's disk block size.

totalBlocks

   (OUT) Points to the total number of blocks on the resource.

freeBlocks

   (OUT) Points to the number of free blocks on the resource.

resourceIsRemovable

   (OUT) Points to a flag indicating whether the resource is removable.

   TRUE Resource is removable
   FALSE Resource is not removable

purgeableBlocks

   (OUT) Points to the total number of blocks set aside as purgeable blocks.

notYetPurgeableBlocks

   (OUT) Points to the number of blocks not marked to be purged.

migratedSectors

(OUT) Points to the number of migrated sectors.

precompressedSectors

(OUT) Points to the number of sectors used by all data sets before they were compressed.

compressedSectors

(OUT) Points to the number of sectors used by all compressed data sets.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFD0	NWSMTS_NO_MORE_NAMES
0xFFFDFFDD	NWSMTS_INVALID_PARAMETER
0xFFFDFFE9	NWSMTS_GET_SERVER_INFO_ERR
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMTSGetTargetResource is called, the engine must be connected to a TSA and Target Service.

NWSMTSGetTargetResource does not return information for all resources listed by NWSMTSScanTargetServiceResource or NWSMTSListTSResources. Only resources that return information by calling NWSMTSGetTargetResourceInfo can return information. NWSMTS_INVALID_PARAMETER is returned for resources that do not have any information.

The maximum block size value returned by blockSize is 32 KB. If the resource has a block size of 64 KB, blockSize returns zero.

See Also

NWSMTSListTSResources (page 134), NWSMTSScanTargetServiceResource (page 141), NWSMTSConnectToTargetServiceEx (page 68)

Example

```
#include <smsutapi.h>
#include <smstsapi.h>

char resourceName[NWSM_MAX_RESOURCE_LEN];
```

```
UINT16 blockSize;
UINT32 totalBlocks, freeBlocks, purgeableBlocks,
notYetPurgeableBlocks, migratedSectors, precompressedSectors,
compressedSectors; NWBOOLEAN resourceIsRemovable;

/* select a resource name from the resource list and make the call */
NWSMTSGetTargetResourceInfo(connection, (STRING)resourceName,
&blockSize,
      &totalBlocks, &freeBlocks, &resourceIsRemovable,
&purgeableBlocks,
     &notYetPurgeableBlocks, &migratedSectors, &precompressedSectors,
      &compressedSectors);
```

NWSMTSGetTargetResourceInfoEx

Returns information about a primary resource.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSGetTargetResourceInfoEx (
   UINT32      connectionID,
   STRING      resourceName,
   UINT32      *bufferSize,
   void        *buffer);
```

Parameters

connectionID

   (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServicEx.

resourceName

   (IN) Specifies the resource name returned by NWSMTSScanTargetServiceResource or NWSMTSListTSResources.

bufferSize

   (IN/OUT) Specifies the size of buffer in bytes.

buffer

   (OUT) Points to the buffer containing resource information.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFC5	NWSMTS_RESOURCE_NAME_NOT_FOUND
0xFFFDFFDD	NWSMTS_INVALID_PARAMETER
0xFFFDFFE9	NWSMTS_GET_SERVER_INFO_ERR
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMTSGetTargetResourceEx is called, the engine must be connected to a TSA and Target Service.

NWSMTSGetTargetResourceEx does not return information for all resources listed by NWSMTSScanTargetServiceResource or NWSMTSListTSResources. Only resources that return information by calling NWSMTSGetTargetResourceInfoEx can return information. NWSMTS_INVALID_PARAMETER is returned for resources that do not have any information.

buffer contains TSA encoded extensions that hold resource information. This can be processed using the extension functions. See Section 6.3, "Extension Functions," on page 218. The extensions and their fields are detailed in Section 5.7, "Extensions," on page 182.

buffer should be allocated by the application and the corresponding size should be passed in bufferSize parameter. Applications can invoke the API with a bufferSize of 0 and a buffer of NULL to get the required bufferSize to encode all extensions.

If passed in buffer size is not adequate to store all extensions NWSMTS_BUFFER_UNDERFLOW error is returned. The engine can then try with a larger buffer size.

NOTE: This symbol is not present in the SMS import file, as all versions of SMS do not export this symbol. To use this function, programmatically import the symbol and invoke it.

See Also

NWSMTSGetTargetResourceInfo (page 111), NWSMTSGetUnsupportedOptions (page 127), NWSMGetFirstExtension (page 222)

Example

```
#include <smsutapi.h>
#include <smsdrapi.h>

CCODE cCode;
UINT32 connection;
UINT32 tsaSequence = 0;
STRING resourceName;
UINT32 bufferSize = 0;
void  *buffer = NULL;
UINT32handle = 0;
BOOL   doNotInvokeSetArchiveStatus = FALSE;
NWSM_EXTENSION_INFORMATION *extension = NULL;
NWSM_RESOURCE_INFO_EXTN_UNSUPPORTED_DATA_1 *unsupExtension;
/* connection related code goes here */
/* After the connection:
We will call NWSMTSScanTargetServiceResource and for each resource,
retrieve resource information using
NWSMTSGetTargetResourceInfoEx and check the unsupported options for
each.
Based on modify bit support we will set a global to call set archive
status or not */
```

```
/* Loop till no more resources or no errors */
while(!cCode)
{
      /* Scan a resource */
      cCode = NWSMTSScanTargetServiceResource(connection,
&tsaSequence, resourceName);
      if (cCode)
            continue;
      /* Get the required size to allocate the buffer */
      bufferSize = 0;
      if (buffer)
            free(buffer);
      cCode = NWMSTSGetTargetResourceInfoEx(connection, resourceName,
&bufferSize, NULL);
      /* Check if error is invalid parameter, as not all resources can
return resource specific information */
      if (cCode && cCode != NWSMTS_INVALID_PARAMETER)
            continue;
      else if (cCode == NWSMTS_INVALID_PARAMETER)
      {
      /* As the error is invalid parameter, could be that the resource
does not provide resource specific
information, continue to try and get information regarding other
resources */
            cCode = 0;
            continue;
      }
/* Allocate the required size and invoke the API again */
      buffer = malloc(bufferSize);
      if (!buffer)
      {
            cCode = -1;
            continue;
      }
      cCode = NWMSTSGetTargetResourceInfoEx(connection, resourceName,
&bufferSize, buffer);
      if (cCode)
      {
            free(buffer);
            buffer = NULL;
            continue;
      }
      /* Get the unsupported options extension from the resource
information buffer */
      cCode = NWSMGetExtension(buffer, bufferSize,
NWSM_RESOURCE_INFO_EXTN_UNSUPPORTED_TAG, &extension, &handle);
      if (cCode)
      {
            free(buffer);
            buffer = NULL;
            continue;
      }
```

```
      unsupExtension = (NWSM_RESOURCE_INFO_EXTN_UNSUPPORTED_DATA_1
*)(extension->info);
      /* Check if modify flag is supported on the resource */
      if (unsupExtension->usupportedBackupOptions &
NWSM_BACK_MODIFY_FLAG)
/* Although this needs to be determined for each resource, this example
assumes that this is globally true */
            doNotInvokeSetArchiveStatus = TRUE;
      NWSMCloseExtension(&handle);
}
```

NWSMTSGetTargetScanTypeString

Returns the strings that describe a supported scan type.

Syntax

```
#include <smsutapi.h>
#include <smstsapi.h>

CCODE NWSMTSGetTargetScanTypeString (
   UINT32   connection,
   UINT8    typeNumber,
   STRING   scanTypeString,
   UINT32  *required,
   UINT32  *disallowed);
```

Parameters

connection

> (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServicEx.

typeNumber

> (IN) Specifies the scan bit, or scan type number, (0- 31) to query the TSA about.

scanTypeString

> (OUT) Specifies the name of the scan type (maximum length is NWSM_MAX_STRING_LEN).

required

> (OUT) Points to the bit map of all scan type bits that must be set if typeNumber is used.

disallowed

> (OUT) Points to the bit map of all scan type bits that must be cleared if typeNumber is used.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFBF	NWSMTS_SCAN_TYPE_NOT_USED
0xFFFDFFDB	NWSMTS_INVALID_SCAN_TYPE
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER

0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMTSGetTargetScanTypeString is called, the engine must be connected to a TSA and Target Service.

NWSMTSGetTargetScanTypeString returns two bit maps that indicate the other scan type options that must be used if scan type typeNumber is used. If a TSA does not support a scan type, NWSMTSGetTargetScanTypeString returns a NULL string and a zero completion code. To get every scan type, the engine must call NWSMTSGetTargetScanTypeString repeatedly until NWSMTS_SCAN_TYPE_NOT_USED is returned.

NWSMTSGetTargetScanTypeString is given a sequence number, or typeNumber, which it uses as an index into a table of strings. This sequence number is not used to indicate the selected scan types. The sequence number must be converted to a bit map or mask before being used.

The following table lists the masks used by SMS for predefined scan types that are used to set the bits in the scan type bit map.

Option	Value	Description
0	0x0001	NWSM_DO_NOT_TRAVERSE: Do not traverse the file system tree
1	0x0002	NWSM_EXCLUDE_ARCHIVED_CHILDREN: Do not scan for children data sets whose archive flag is set.
2	0x0004	NWSM_EXCLUDE_HIDDEN_CHILDREN: Do not scan for children data sets whose hidden flag is set.
3	0x0008	NWSM_EXCLUDE_HIDDEN_PARENT: Do not scan for parent data sets whose hidden flag is set.
4	0x0010	NWSM_EXCLUDE_SYSTEM_CHILDREN: Do not scan for children data sets whose system flag is set.
5	0x0020	NWSM_EXCLUDE_SYSTEM_PARENT: Do not read for parent data sets whose system flag is set.

The following table lists the masks used for scan types that are returned by the file system TSAs:

Option	Value	Description
6	0x0040	NWSM_EXCLUDE_CHILD_TRUSTEES: Do not read for the trustee information of children data sets.
7	0x0080	NWSM_EXCLDE_PARENT_TRUSTEES: Do not read for the trustee information of parent data sets.
8	0x0100	NWSM_EXCLUDE_ACCESS_DATABASE: Do not read the database.
9	0x0200	NWSM_EXCLUDE_VOLUME_RESTS: Do not read for primary resource restriction information.

Option	Value	Description
10	0x0400	NWSM_EXCLUDE_DISK_SPACE_RESTS: Do not read for disk space restriction information.
11	0x0800	NWSM_EXCLUDE_EXTENDED_ATTRIBUTES: Do not read for extended attribute information.
12	0x1000	NWSM_EXCLUDE_DATA_STREAMS: Do not read a data set's data stream.
13	0x2000	NWSM_EXCLUDE_MIGRATED_CHILD: Do not read migrated data streams of children data sets. Read only the stub information for these migrated children.
14	0x4000	NWSM_EXPAND_COMPRESSED_DATA: Expand the data set before scanning it.
15	0x8000	NWSM_EXCLUDE_ARCH_CHILD_DATA: Do not scan the data of children that have been archived. This is used only for Directory Services.
16	0x10000	NWSM_EXCLUDE_ARCH_CHILD_CHAR: Do not scan the characteristics of children data sets. This is used only for Directory Services.
17	0x20000	NWSM_FLAG_PURGE_IMMED_ON_DELETE: Set the data set's purge immediately flag when it is deleted.
18	0x40000	NWSM_EXCLUDE_MIGRATED_FILES: Do not scan for children whose remote data access bit is set.
19	0x80000	NWSM_INCLUDE_PATH_COMPONENT: For each item in the include list, backup the individual parent components before processing the data sets.
20	0x100000	NWSM_EXCLUDE_HARDLINK_DATA : Do not backup the data of consecutive hard link nodes of a particular hardlink network, except for the first encountered node.

To indicate the selected scan types to a TSA, set and clear the appropriate bits of scanType in the NWSM_SCAN_CONTROL structure when calling NWSMTSScanDataSetBegin (see Connecting to the Target Service).

The order of scan type strings corresponds to the order of the scan type bit map in the *scanType* field of NWSM_SCAN_CONTROL-string one represents bit zero while string two represents bit one, etc.

See Also

Example

```
#include <smsutapi.h>
#include <smstsapi.h>

typedef struct TSA_SCAN_TYPE
{
   UINT32 scanType;
   UINT32 required;
   UINT32 disallowed;
```

```
   struct TSA_SCAN_TYPE *next;
   UINT8  scanTypeString[1];
} TSA_SCAN_TYPE;

char scanTypeString[NWSM_MAX_STRING_LEN];
UINT8 typeNumber = 0;
UINT32 required, disallowed, chosenScanTypes, scanType;
TSA_SCAN_TYPE *scanTypes = NULL, *last, *st;
NWSM_SCAN_CONTROL scanControl = {0};
CCODE ccode;

/* Build a list of the scan types that a TSA has. */
for(typeNumber = 0, scanType = 1; typeNumber < 32; typeNumber++,
scanType <<= 1)
{
   if ((ccode = NWSMTSGetTargetScanTypeString(connection, typeNumber,
         (STRING)scanTypeString, &required, &disallowed)) ==
         NWSMTS_SCAN_TYPE_NOT_USED)
      continue;

   if (ccode)
      break;
   st = (TSA_SCAN_TYPE *)calloc(1, sizeof(TSA_SCAN_TYPE) +
         strlen(scanTypeString));
   st->scanType = scanType;
   st->required = required;
   st->disallowed = disallowed;
   strcpy(st->scanTypeString, scanTypeString);

   if(scanTypes)
   {
      last->next = st;
      last = st;
   }

   else
      scanTypes = last = st;
}

/* Build a display from the information retrieved from
NWSMTSGetTargetScanTypeString, and get the user's selection.
The display will list the scanning type options available to the user.
The display routine will also check for invalid scan type combinations
by comparing the required and disallowed
bits of a just chosen scan type against already chosen scan types.
The valid chosen scan types will be set into chosenScanTypes. */

scanControl.scanType = chosenScanTypes;
```

This is a simplified example of retrieving all the scan type strings from the TSA.

```
/* typeNumber is initialized to zero to indicate to start from the
first scan type string.
It is then incremented from 0 to 31 to get all the strings.
```

```
scanType contains the scan type bit mask that represents the scan type
option.
scanType is also initialized to one since the first scan type option
starts at bit 0. */

for (typeNumber = 0, scanType = 1; typeNumber < 32; typeNumber++,
scanType <<= 1)
{
   if ((ccode = NWSMTSGetTargetScanTypeString(connection, typeNumber,
         (STRING)scanTypeString, &required, &disallowed)) ==
         NWSMTS_SCAN_TYPE_NOT_USED)
      continue;
/* The continue code takes care of cases where the TSA might not allow
specific options. scanType, scanTypeString,
required, and disallowed are copied into linked list that is used to
build the scan type option list. */
...
}
```

NWSMTSGetTargetSelectionTypeStr

Returns the SMS-defined selection type strings.

Syntax

```
#include <smsutapi.h>
#include <smstsapi.h>

CCODE NWSMTSGetTargetSelectionTypeStr (
   UINT32   connection,
   UINT8    typeNumber,
   STRING   selectionTypeString1,
   STRING   selectionTypeString2);
```

Parameters

connection

   (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or
   NWSMTSConnectToTargetServicEx.

typeNumber

   (IN) Specifies the selection type number (1-31) to query the TSA about.

selectionTypeString1

   (OUT) Receives the string that describes that describes the selection type.

selectionTypeString2

   (OUT) Receives the string that describes the selection type if bit zero and typeNumber of
   selectionType are set.

Return Values

See for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFBE	NWSMTS_SELECTION_TYPE_NOT_USED
0xFFFDFFD9	NWSMTS_INVALID_SELECTION_TYPE
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMTSGetTargetSelectionTypeStr is called, the engine must be connected to the TSA and Target Service.

One or both of the strings that NWSMTSGetTargetSelectionTypeStr returns can be NULL strings, depending on the options supported by the TSA. For a list of selection types, see NWSM_SELECTION_LIST (page 172). typeNumber is used as an index to a string table and must be converted to a bit value before being used in the selection list.

If only one selection option of the selection option pair is supported, NWSMTSGetTargetSelectionTypeStr returns successfully. If both selection types of the selection pair are not supported, NWSMTSGetTargetSelectionTypeStr returns NWSMTS_SELECTION_TYPE_NOT_USED.

If there is no option for bit zero and typeNumber of selectionType being set in selectionTypeString2, a NULL string is returned. The engine must allocate NWSM_MAX_STRING_LEN bytes for the buffer.

You must call NWSMTSGetTargetSelectionTypeStr repeatedly until all selection option strings are returned.

The first pair of returned strings represents the combination of bits 0 and 1. The second pair of returned strings represents the combination of bits 0 and 2, etc. This information is important so the proper bits can be set when the user chooses a selection type from the selection option list.

See Also

NWSMTSGetOpenModeOptionString (page 104)

Example

```
/* This example retrieves the selection types from the TSA, queries the
user for the selection types to use,
and sends the user selected selection types back to the TSA. */

#include <smsutapi.h>
#include <smstsapi.h>

typedef struct TSA_SELECTION_TYPE
{
   UINT32 selectionType1;
   UINT32 selectionType2;
   STRING selectionStringType1;
   STRING selectionStringType2;
   struct TSA_SELECTION_TYPE *next;
} TSA_SELECTION_TYPE;

typedef struct USER_SELECTION
{
   UINT32 selectionType;
   STRING resourceName;
   struct USER_SELECTION *next;
```

```
} USER_SELECTION;

UINT8 typeNumber;
unsigned char s1[NWSM_MAX_STRING_LEN], s2[NWSM_MAX_STRING_LEN];
STRING selectionTypeString1 = s1, selectionTypeString2 = s2;
TSA_SELECTION_TYPE *selectionTypeList = NULL, *last = NULL, *tmp;
USER_SELECTION *userSelections, *nextUserSelection;
NWSM_SELECTION_LIST *selectionList;
UINT32 HUGE nameHandle;
UINT32 selectionType;
CCODE ccode;

/* Get the target selection type strings. */
for(typeNumber = 1, selectionType = 2; typeNumber < 32; typeNumber++,
      selectionType <<= 1)
{
   if((ccode = NWSMTSGetTargetSelectionTypeStr(connection, typeNumber,
         selectionTypeString1, selectionTypeString2)) ==
         NWSMTS_SELECTION_TYPE_NOT_USED)
      continue;

   if(ccode)
      break;

   /* Append selection type to selection type list. */
   TMp = (TSA_SELECTION_TYPE *)calloc(1, sizeof(TSA_SELECTION_TYPE));
   TMp->selectionType1 = selectionType;
   TMp->selectionType2 = selectionType + 1;
   TMp->selectionStringType1 = selectionTypeString1;
   TMp->selectionStringType2 = selectionTypeString2;

   if (last)
   {
      last->next = TMp;
      last = TMp;
   }
   else
      selectionTypeList = last = TMp;
} /* /End for */

/* Build a display from the information retrieved from
NWSMTSGetTargetSelectionTypeStr, and NWSMTSListTSResources or
NWSMTSScanTargetServiceResource.
The last two functions build a list of resources that can be used with
each selection type.
See "Data Set Selection Options" and "Using Resources with Selection
Options."
Also see NWSMTSListTSResources or example code. */

/* Using the display, get the user's selection (put each selection type
and resource name into a list
using USER_SELECTION) userSelections points to this list. */

/* Build the selection list to pass to the TSA. For
```

```
defaultNameSpaceType,
reverseOrder, firstSeparator, and secondSeparator see example code of
NWSMTSListTSResources */

selectionList = NULL;
NWSMPutFirstName(&selectionList, defaultNameSpaceType,
      userSelections->selectionType, reverseOrder, firstSeparator,
      secondSeparator, userSelections->resourceName, &nameHandle);

for (nextUserSelection = userSelections; nextUserSelection;
      nextUserSelection = nextUserSelection->next)
{
   NWSMPutNextName(&selectionList, &nameHandle, defaultNameSpaceType,
       nextUserSelection->selectionType, reverseOrder, firstSeparator,
       secondSeparator, nextUserSelection->resourceName);
}

NWSMCloseName(&nameHandle);

/* Get open modes. */
```

This is a simplified example of retrieving strings from the TSA.

```
/* typeNumber is initialized to 1 to indicate to start from the first
pair of selection strings.
It is incremented from 0 through 31 to get all the strings.
selectionType contains the bit mask for the first option
of the selection pair.
One is added to this bit mask to get the bit mask for the second option
of the selection type pair.
selectionType is initialized to 0x2 because the first selection type
starts at bit 1 of the selection bit map. */

for (typeNumber = 1, selectionType = 0x2; typeNumber < 32;
typeNumber++,
        selectionType <<= 1)
{
   if ((ccode = NWSMTSGetTargetSelectionTypeStr(connection,
typeNumber,
        selectionTypeString1,
        selectionTypeString2)) == NWSMTS_SELECTION_TYPE_NOT_USED)
      continue;
/* Continue takes care of any cases where a TSA does not support a
selection type. */
    if(ccode)
        break;
   /* Copy each selection type string and bit map pair (i.e.,
selectionType/
     selectionTypeString1 and selectionType + 1/selectionTypeString2)
to a
     linked list that is used to build the selection option list. */
   ...
}
```

NWSMTSGetUnsupportedOptions

Returns a list of options not supported by the TSA.

Syntax

```
#include <smsutapi.h>
#include <smstsapi.h>

CCODE NWSMTSGetUnsupportedOptions(
   UINT32   connection,
   UINT32  *unsupportedBackupOptions,
   UINT32  *unsupportedRestoreOptions);
```

Parameters

connection

(IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServicEx.

unsupportedBackupOptions

(OUT) Points to a bit map that represents the TSA's unsupported backup options.

unsupportedRestoreOptions

(OUT) Points to a bit map that represents the TSA's unsupported restore options.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMTSGetUnsupportedOptions is called, the engine must be connected to a TSA and Target Service.

NWSMTSGetUnsupportedOptions returns two 32-bit maps that indicate which backup or restore options are not supported. It also indicates which options can be used in the NWSM_SCAN_CONTROL structure and which generic open mode options can be used by NWSMTSOpenDataSetForRestore and NWSMTSOpenDataSetForBackup.

Use NWSMTSGetTargetResourceInfoEx to get additional information on unsupported options by a specific primary resource.

unsupportedBackupOptions refers to the fields in the NWSM_SCAN_CONTROL and NWSM_SCAN_INFORMATION structures. If an engine passes information via one of these unsupported fields, the TSA ignores it.

The following tables list the unsupported backup and restore options that can be returned by the TSA.

Option	Value	Description
0	0x01	NWSM_BACK_ACCESS_DATE_TIME: The TSA does not support the following fields:
		NWSM_SCAN_CONTROL
		firstAccessDateAndTime lastAccessDateAndTime
		NWSM_SCAN_INFORMATION
		accessDateAndTime
1	0x02	NWSM_BACK_CREATE_DATE_TIME: The TSA does not support the following fields:
		NWSM_SCAN_CONTROL
		firstCreateDateAndTime lastCreateDateAndTime
		NWSM_SCAN_INFORMATION
		createDateAndTime
2	0x04	NWSM_BACK_MODIFIED_DATE_TIME: The TSA does not support the following fields:
		NWSM_SCAN_CONTROL
		firstModifiedDateAndTime lastModifiedDateAndTime
		NWSM_SCAN_INFORMATION
		modifiedDateAndTime
3	0x08	NWSM_BACK_ARCHIVE_DATE_TIME: The TSA does not support the following fields:
		NWSM_SCAN_CONTROL
		firstArchivedDateAndTime lastArchivedDateAndTime
		NWSM_SCAN_INFORMATION
		archivedDateAndTime
		NWSMTSSetArchiveStatus
		setFlag
		NWSM_SET_ARCHIVE_DATE_AND_TIME

Option	Value	Description
4	0x10	NWSM_BACK_SKIPPED_DATA_SETS: The TSA does not support the following fields: NWSM_SCAN_INFORMATION createSkippedDataSetsFil

5	0x08	NWSM_BACK_MODIFY_FLAG: The TSA does not support the following fields:
		
		NWSM_SCAN_CONTROL
		 scantype
		
		NWSM_EXCLUDE_ARCHIVED_CHILDREN
		NWSM_EXCLUDE_ARCH_CHILD_DATA
		
		NWSMTSSetArchiveStatus
		setFlag
		
		 NWSM_SET_MODIFY_FLAGNWSM_CLEAR_MODIFY_FLAG
		   NWSM_SET_ARCHIVER_ID
		
		NWSM_SCAN_INFORMATION
		attributes
		
		 NWFA_ARCHIVE
		   NWFA_OBJ_ARCHIVE_BIT
		
		modifiedFlag

Option	Value	Description
0	0x01	NWSM_RESTORE_NEW_DATA_SET_NAME: NWSMTSOpenDataSetForRestore does not rename data sets.
1	0x02	NWSM_RESTORE_CHILD_UPDATE_MODE: The TSA does not restore a child data set if it is newer than the data set on the Target Service.
2	0x04	NWSM_RESTORE_PARENT_UPDATE_MODE: The TSA does not restore a parent data set if it is newer than the one on the Target Service.
3	0x08	NWSM_RESTORE_PARENT_HANDLE: NWSMTSOpenDataSetForRestore and NWSMTSWriteDataSet do not accept parent handles.

See Also

NWSMTSGetNameSpaceTypeInfo (page 102),
NWSMTSGetOpenModeOptionString (page 104),
NWSMTSGetTargetResourceInfo (page 111),
NWSMTSGetTargetResourceInfoEx (page 114)
NWSMTSGetTargetScanTypeString (page 118),
NWSMTSGetTargetSelectionTypeStr (page 123),
NWSMTSListSupportedNameSpaces (page 132),
NWSMTSListTSResources (page 134),
NWSMTSScanDataSetBegin (page 38),

Example

```
#include <smsutapi.h>
#include <smstsapi.h>
#define ADDITIONAL_BUFFER_SIZE 512

UINT32 unsupportedBackupOptions = 0, unsupportedRestoreOptions = 0;
NWSM_SCAN_CONTROL *scanControl;

/* Find out which options the TSA does not support */
NWSMTSGetUnsupportedOptions(connection, &unsupportedBackupOptions,
    &unsupportedRestoreOptions);

/* Setup the scan control structure. Note that some fields in
scanControl are set to their default values
when the memory is calloc'd. */
scanControl = (NWSM_SCAN_CONTROL *)calloc(1, sizeof(NWSM_SCAN_CONTROL)
+
   ADDITIONAL_BUFFER_SIZE);
scanControl->bufferSize = sizeof(NWSM_SCAN_CONTROL) +
ADDITIONAL_BUFFER_SIZE;
scanControl->scanControlSize = sizeof(NWSM_SCAN_CONTROL);
scanControl->otherInformationSize = 0;

/* With the unsupported options, build an appropriate display. The
display is used to gather information from the
user about which date sets are to be selected or ignored. This
information is then put into an NWSM_SCAN_CONTROL
structure.
This process is the same for a back-up or restore session. */
if (!(unsupportedBackupOptions & NWSM_BACK_ACCESS_DATE_TIME))
{
   /* Build display field for first access time and last access time.
    scanControl->firstAccessDateAndTime and scanControl-
    >lastAccessDateAndTime are used to contain the input data. */
}
if (!(unsupportedBackupOptions & NWSM_BACK_CREATE_DATE_TIME))
{
   /* Build display field for first created time and last created time.
       scanControl->firstCreateDateAndTime and scanControl->
       lastCreateDateAndTime are used to contain the input data. */
}

if (unsupportedBackupOptions & NWSM_RESTORE_PARENT_HANDLE)
   scanControl->returnChildTerminalNodeNameOnly = FALSE;
else
   scanControl->returnChildTerminalNodeNameOnly = TRUE;

/* Display the options and gather the user's input for these options.
*/
```

```
/* Get and set the scan and selection options-see
NWSMTSGetTargetScanTypeString and NWSMTGetTargetSelectionTypeStr. */
/* Pass the options to the back-up or restore process. */
/*Make a log of the data sets that where skipped during the backup
process*/
/* Read the skipped data set log from the TSA, and put the data into a
user
   readable file. See NWSMTSScanDataSetBegin. */
If (!(unsupportedBackupOptions & NWSM_BACK_SKIPPED_DATA_SETS))
{
   /* Read the skipped data set log from the TSA, and put the data into
a user
      readable file. See NWSMTSScanDataSetBegin. */
}
/* Make a log of the errors that occurred during the backup process.
See
   NWSMTSScanDataSetBegin. */
```

NWSMTSListSupportedNameSpaces

Returns the name space supported by a primary resource.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSListSupportedNameSpaces (
   UINT32             connection,
   STRING             resourceName,
   NWSM_NAME_LIST **supportedNameSpaces);
```

Parameters

connection

(IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServicEx.

resourceName

(IN) Specifies a resource name returned by NWSMTSListTSResources or NWSMTSScanTargetServiceResource.

supportedNameSpaces

(OUT) Points to an allocated block of memory that contains a list of name spaces.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFC5	NWSMTS_RESOURCE_NAME_NOT_FOUND
0xFFFDFFC9	NWSMTS_OUT_OF_MEMORY
0xFFFDFFCE	NWSMTS_NO_SUCH_PROPERTY
0xFFFDFFD0	NWSMTS_NO_MORE_NAMES
0xFFFDFFE4	NWSMTS_INVALID_DATA_SET_NAME
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMTSListSupportedNameSpaces is called, the engine must be connected to a TSA and Target Service.

NWSMTSListSupportedNameSpaces is a data requestor function. The first list element is the default name space of resourceName.

NOTE: NWSMTSListSupportedNameSpaces is implemented with NWSMTSScanSupportedNameSpaces.

See Also

NWSMTSScanSupportedNameSpaces (page 138)

Example

```
/* This example queries the user for the resource that is to be
selected for the session. */

#include <smstsapi.h>

char *defaultNameSpaceName = NULL, *name;
NWSM_NAME_LIST *nameList = NULL;
UINT32 defaultNameSpaceType;

/*Build a complete list of primary resources that are on the file
server. */
NWSMTSListTSResources(connection, &nameList);

/* If the resource name list is not empty, get the name spaces
supported by the first resource.
Remember that the first name returned by NWSMTSListTSResources or
NWSMScanTargetServiceResrouce is always the
resource that contains all other resources.
Here, the first name is "FILE SERVER" */
if (nameList != NULL)
{
  /* Build the name space list. Remember that the first name space is
the default name space of the Target Service. */
   NWSMTSListSupportedNameSpaces(connection, nameList->name,
     &supportedNameSpaces);

   /* Get default name space */
   if (supportedNameSpaces)
   {
     defaultNameSpaceType = *((UINT32 *)supportedNameSpaces->name);
      name = (char *)((UINT32 *)supportedNameSpaces->name +
sizeof(UINT32));
     defaultNameSpaceName = strdup(name);
   }
}
```

NWSMTSListTSResources

Returns a list of primary resources.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSListTSResources (
   UINT32            connection,
   NWSM_NAME_LIST **serviceResourceList);
```

Parameters

connection

(IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServicEx.

serviceResourceList

(OUT) Points to the list containing the primary resources found on the Target Service (maximum length is NWSM_MAX_RESOURCE_LEN bytes).

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFEFFFD	NWSMDR_OUT_OF_MEMORY
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMTSListTSResources is called, the engine must be connected to the TSA and Target Service and you should call NWSMTSBuildResourceList.

The first entry in the list that NWSMTSListTSResources returns contains all the resources on a Target Service (see Resources).

The behavior of NWSMTSListTSResources differs from cluster backup and normal backup. If the engine is connected to a pool, NWSMTSListTSResources only lists the mounted resources under the pool. If the engine is connected to the server node, then all the non-clustered resources that are mounted on the server are listed.

Before passing a resource name to NWSMTSScanDataSetBegin, the engine must convert it to an NWSM_DATA_SET_NAME_LIST structure. Use the data set name functions described in *Storage Management Services Utilities Library* to help convert this name.

NWSMTSListTSResources is implemented with NWSMTSScanTargetServiceResource.

See Also

Example

```
/* This example queries the user for the resource that is to be
selected for the session. */

#include <smstsapi.h>
#include <smsutapi.h>

UINT32 sequence = 0, nameSpaceType, defaultNameSpaceType,
selectionType;
char nameSpaceName[61]; /* Name space names not longer than 60
characters */
NWSM_NAME_LIST *nameList = NULL;
NWSM_DATA_SET_NAME_LIST dataSetName = NULL, *parent;
NWBOOLEAN   reverseOrder;
NWSM_SCAN_CONTROL scanControl = {0};
NWSM_SELECTION_LIST   *selectionList;
STRING string = (STRING)nameSpaceName, firstResourceName
STRING_BUFFER *firstSeparator = NULL, *secondSeparator = NULL;

/*Build a complete list of primary resources that are on the file
server. */
NWSMTSListTSResources(connection, &nameList);

/*If the resource name list is not empty, get the name spaces supported
on the file server. Remember that the first name
returned by NWSMTSListTSResources or NWSMScanTargetServiceResrouce is
always the resource that contains
all other resources. Here, the first name is "FILE SERVER" */

if (nameList != NULL)
{
   /* Get all the name spaces supported by the file server. */
   sequence = 0;
 while(NWSMTSScanSupportedNameSpaces(connection, &sequence, nameList-
>name,
     &nameSpaceType, string) == 0)
   {
    /* Build the name space list here. Remember that the first returned
name
     space name is the default name space of the Target Service.
      defaultNameSpaceType is set to this name space. */
   }
```

```
}
/* Get the default name space information */
defaultNameSpaceType = default name space type;
NWSMTSGetNameSpaceTypeInfo(connection, defaultNameSpaceType,
&reverseOrder, &firstSeparator, &secondSeparator);


/* Present the primary resource name list to the user and find out
which resource is to be selected. */
/* If the user has chosen a resource, see if it has any secondary
resources.
If it has secondary resources, list them and show them to the user for
selection.
If the first resource name was chosen, there is no need to scan for
secondary resources.
Note: We are assuming that the selected resource was put into
firstResourceName.
The following code can form the path to what the user wants to select
or ignore. */
while(1)
{
    /* First put the resource name into an NWSM_DATA_SET_NAME_LIST
structure.
        selectionType is set to zero to indicate that dataSetName is
being used
      as a name list only. */
    selectionType = 0;
    firstResourceName = nameList->name;
    NWSMPutOneName(&dataSetName, nameSpaceType, selectionType,
reverseOrder, firstSeparator->string,
secondSeparator->string, firstResourceName);

    /* Now setup scanControl to specify that we are looking for names of
the parents just under the resource, not their
path information, the following are set. Here will assume that the TSA
supports all scan types. See
NWSMTSGetTargetScanTypeString for more information. */
    scanControl.scanType = 0;
    scanControl.returnChildTerminalNodeNameOnly = FALSE;
    scanControl.childrenOnly = FALSE;
    scanControl.parentsOnly = TRUE;
    scanControl.returnNameSpaceType = NWSM_ALL_NAME_SPACES;
    scanControl.bufferSize = scanControl.scanControlSize =
        sizeof(NWSM_SCAN_CONTROL);
    /* Since we want the names of all parents (subdirectories) just
under the
  resource name, set selectionList to NULL. */
    selectionList = NULL;
    parent = NULL;
  /* We do not know if the resource has any parents (subdirectories or
      children (files). To find this out, call data set begin. */
  if (NWSMTSScanDataSetBegin(connection, dataSetName, &scanControl,
        selectionList, &sequence, NULL, &parent) == 0)
  {
  /* Found one parent. Put the name into a name list. */
```

```
   /* Find the rest of subdirectory names, and put them into the name
list. */
      while(NWSMTSScanNextDataSet(connection, &sequence, NULL,
&dataSetname) == 0)
      {
         /* Put the name into the name list. */
      }


  /* Now show the finished list to the user for selection. If the user
chooses a name from this list,
it can be appended to what we already have.
Information from NWSMTSGetNameSpaceTypeInfo can be used to help build
the path. */
   } /* end if data set begin */
} /* end while(1) */
```

NWSMTSScanSupportedNameSpaces

Returns information about one name space that is supported by a primary resource.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSScanSupportedNameSpaces (
   UINT32   connection,
   UINT32  *sequence,
   STRING   resourceName,
   UINT32  *nameSpace,
   STRING   nameSpaceName);
```

Parameters

connection

   (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServicEx.

sequence

   (IN/OUT) Points to the sequence to use with the supported name spaces (set to zero initially).

resourceName

   (IN) Specifies a resource name returned by NWSMTSListTSResources or NWSMTSScanTargetServiceResource.

nameSpace

   (OUT) Points to a number that represents the name space.

nameSpaceName

   (OUT) Specifies the name of the name space (NWSM_MAX_STRING_LEN bytes).

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFC5	NWSMTS_RESOURCE_NAME_NOT_FOUND
0xFFFDFFD0	NWSMTS_NO_MORE_NAMES
0xFFFDFFE4	NWSMTS_INVALID_DATA_SET_NAME
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER

0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMTSScanSupportedNameSpaces is called, the engine must be connected to a TSA and Target Service

See Also

Example

```
/* This example queries the user for the resource that is to be
selected for the session. */

defaultNameSpaceType = *((UINT32 *)supportedNameSpaces->name);
name = (char *)((UINT32 *)supportedNameSpaces->name + sizeof(UINT32));
defaultNameSpaceName = (char *)malloc(strlen(name) + 1);
strcpy(defaultNameSpaceName, name);

#include <smstsapi.h>

char *defaultNameSpaceName, resourceNameBuf[NWSM_MAX_RESOURCE_LEN],
 nameSpaceNameBuf[61]; /* Name space names no longer than 60 characters
*/
NWSM_NAME_LIST *nameList = NULL;
UINT32 defaultNameSpaceType, sequence = 0, nameSpaceType,
firstTime = TRUE;
STRING resourceName = resourceNameBuf,
nameSpaceName = nameSpaceNameBuf;

/*Build a complete list of primary resources that are on the file
server. */
NWSMTSListTSResources(connection, &nameList);

/* If the resource name list is not empty, get the name spaces
supported by the first resource.
Remember that the first name returned by NWSMTSListTSResources or
NWSMScanTargetServiceResource is always the resource
that contains all other resources. Here, the first name is "FILE
SERVER" */

if (nameList != NULL)
{
  /* Build the name space list. Remember that the first name space is
the default name space of the Target Service. */
   while(NWSMTSScanSupportedNameSpaces(connection, &sequence,
resourceName, &nameSpaceType, nameSpaceName) == 0)
   {
      if (firstTime)
      {
```

```
                /* Get default name space */
                firstTime = FALSE;
                defaultNameSpaceType = nameSpaceType;
                defaultNameSpaceName = strdup(nameSpaceName);
            }
        }
    }
```

NWSMTSScanTargetServiceResource

Returns the name of one primary resource.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSScanTargetServiceResource (
   UINT32   connection,
   UINT32  *rsnSequence,
   STRING   resourceName);
```

Parameters

connection

> (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServicEx.

rsnSequence

> (IN/OUT) Points to the resource sequence number.

resourceName

> (OUT) Receives the name of a primary resource (maximum buffer size is NWSM_MAX_RESOURCE_LEN bytes).

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFC5	NWSMTS_RESOURCE_NAME_NOT_FOUND
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMTSScanTargetServiceResource is called, the engine must be connected to the TSA and Target Service and you must call NWSMTSBuildResourceList.

When provided with supporting code, NWSMTSScanTargetServiceResource provides the same functionality as NWSMTSListTSResources.

The behavior of NWSMTSScanTargetServiceResource differs from cluster backup and normal backup. If the engine is connected to a pool, NWSMTSScanTargetServiceResource only lists the mounted resources under the pool. If the engine is connected to the server node, then all the non-clustered resources that are mounted on the server are listed.

rsnSequence is used by the TSA to keep track of which primary resource was returned. The engine sets rsnSequence to zero before calling NWSMTSScanTargetServiceResource the first time.

If rsnSequence is set to zero (0), resourceName returns the resource that represents all the resources on a Target Service. For example, if the Target Service is a NetWare file server, the first resource name is "File Server." For NetWare, if this first resource name is passed to NWSMTSScanDataSetBegin, all binderies, volumes, directories, and files are scanned. For example, if the Target Service is a Linux server, the first resource name is "/". If a resource is passed to NWSMTSScanDataSetBegin, only the current mount point's directories and files are scanned, all other mount points under the current mount point are excluded. Additionaly, pseudo-file systems like proc, tmpfs, _admin, devpts, usbfs and device files under /dev are always excluded.

To get all the primary resources, the engine calls NWSMTSScanTargetServiceResource repeatedly until NWSMTS_RESOURCE_NAME_NOT_FOUND is returned.

Before passing a resource name to NWSMTSScanDataSetBegin, the engine must convert it to an NWSM_DATA_SET_NAME_LIST structure. Use the data set name functions described in *Storage Management Services Utilities Library* to help convert the name.

See Also

Example

```
/* See NWSMTSListTargetServiceResource's example. Replace the
following code for the call to NWSMTSListTSResources. */

UINT32 sequence;
char resourceName[NWSM_MAX_RESOURCE_LEN];
STRING string;
/*Build a complete list of primary resources that are on the file
server. */
sequence = 0;
string = resourceName;
while(NWSMTSScanTargetServiceResource(connection, &sequence, string)
== 0)
{
   /* Insert resource name into a list. nameList will contain the list.
*/
}
```

3.5 Restore Functions

The following functions restore the backed up data sets:

-
-
-
-

NWSMTSIsDataSetExcluded

Compares a data set name against its internal selection list (set up by NWSMTSSetRestoreOptions) and indicates if the data set is included or not.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSIsDataSetExcluded (
   UINT32                      connection,
   NWBOOLEAN                   isParent,
   NWSM_DATA_SET_NAME_LIST  *dataSetName);
```

Parameters

connection

    (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServicEx.

isParent

    (IN) Specifies a flag returned in NWSM_SCAN_INFORMATION indicating whether the data set is a parent:

    TRUE Is a parent
    FALSE Is a child

dataSetName

    (IN) Points to the data set's fully qualified path.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFDC	NWSMTS_INVALID_PATH
0xFFFDFFDD	NWSMTS_INVALID_PARAMETER
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFDFFF7	NWSMTS_DATA_SET_EXCLUDED
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

TSA Developer

To build dataSetName the "Data Set Name Functions" listed in *Storage Management Services Library* can be used.

Remarks

Before NWSMTSIsDataSetExcluded is called, NWSMTSSetRestoreOptions must be called with *dontCheckSelectionList* set to TRUE.

NWSMTSIsDataSetExcluded uses the selection list from NWSMTSSetRestoreOptions to decide if dataSetName is excluded. NWSMTSIsDataSetExcluded is used to help speed up the restore process and does not have to be used (see Set the Restore Options).

dataSetName can be taken from a database or from a transfer buffer received from SMS DI. If the data set name is not already in an NWSMTS_DATA_SET_NAME_LIST structure, the "Data Set Name Functions" shown in *Storage Management Services Utilities Library* can be used to insert it into the structure. The data for this list can only come from NWSMTSScanDataSetBegin or NWSMTSScanNextDataSet. From this list, the only element that uses the name space supported by the Target Service will be compared against the internal selection list.

NOTE: The bindery must be explicitly included in the restore session or it will not be restored.

If the data set name was not produced by the TSA, the engine must be aware of the two following precautions:

- ◆ If the engine builds the data set name list, the engine must ensure that the data set name format and case format follows the intended name space's specifications.
- ◆ Parent data set names may need an end separator if the name space requires it.

NOTE: NWSMTSIsDataSetExcluded will not correct improperly-formatted data set names.

See Also

NWSMTSFixDataSetName (page 85), NWSMTSSetRestoreOptions (page 152)

NWSMTSOpenDataSetForRestore

Creates a data set handle for the data set that is to be restored.

Syntax

```
#include <smsutapi.h>
#include <smstsapi.h>

CCODE NWSMTSOpenDataSetForRestore (
   UINT32                      connection,
   UINT32                      parentHandle,
   NWSM_DATA_SET_NAME_LIST  *newDataSetName,
   UINT32                       mode,
   UINT32                      *dataSetHandle);
```

Parameters

connection

(IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServicEx.

parentHandle

(IN) Specifies the data set handle returned by NWSMTSOpenDataSetForRestore.

newDataSetName

(IN) Points to the data set's new name and location.

mode

(IN) Specifies the open modes to apply.

dataSetHandle

(OUT) Points to a data set handle to use when calling NWSMTSWriteDataSet, NWSMTSCloseDataSet, and NWSMTSSetArchiveStatus.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFAF	NWSMTS_CLUSTER_TARGET_DOES_NOT_EXIST
0xFFFBFFFB	NWSMUT_OUT_OF_MEMORY
0xFFFBFFFC	NWSMUT_NO_MORE_NAMES
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION

0xFFFDFFC9	NWSMTS_OUT_OF_MEMORY
0xFFFDFFD0	NWSMTS_NO_MORE_NAMES
0xFFFDFFDC	NWSMTS_INVALID_PATH
0xFFFDFFDD	NWSMTS_INVALID_PARAMETER
0xFFFDFFE0	NWSMTS_INVALID_NAME_SPACE_TYPE
0xFFFDFFE5	NWSMTS_INVALID_DATA_SET_HANDLE
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFDFFF7	NWSMTS_DATA_SET_EXCLUDED
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Engine Developer

To build newDataSetName the "Data Set Name Functions" listed in *Storage Management Services Library* can be used.

Remarks

NWSMTSOpenDataSetForRestore validates the data set handle (if it has access to the data) and renames or moves the data set to a new location.

NWSMTSOpenDataSetForRestore requires that the combination of parentHandle, newDataSetName, and the path information contained in the data set's data form a fully qualified path. If parentHandle is used, the path information in newDataSetName and in the data set's data is ignored (except the terminal node). If newDataSetName is used, the path information in the data set's data is ignored.

If the original name of the data set is to be kept, newDataSetName must be set to NULL.

If a parent is not needed, the engine must set parentHandle to zero. If parent handles are used, the handles must remain open until all subordinate data sets are restored.

parentHandle is like a pointer to a location in the file system tree. It is used to restore immediate children and parents (subdirectories) of the parent referenced by the handle. Parent handles are used when Target Services has a hierarchical file system and supports the use of parent handles. For example, NDS is a Target Service with a hierarchical file system that does not support the use of parent handles.

NOTE: NWSMTSGetUnsupportedOptions must be called to see if the TSA supports renaming and moving of data set during a restore session.

newDataSetName must contain a fully qualified path if parentHandle is zero. If parentHandle is used, only the terminal node in newDataSetName is used.

NOTE: The data set's path information will automatically be placed into the data set's data by the TSA.

The following table shows when and how to use newDataSetName, parentHandle, and the path information contained in the data set data. The newDataSetName column indicates the contents of the data set name parameter and what portion of it is used. The parentHandle column indicates if a parent handle is passed to the function and if it points to a new location. The third column indicates the contents of the path information in the data set's data and how much of it is used. The first seven rows are used if the data set is renamed or moved. The last four rows are used if the data set's original name is used and newDataSetName is NULL. In rows five and six, parentHandle points to a new location. In rows nine and ten, it points to the data set's original location.

	newDataSetName	parentHandle	Path Information In Data Set	Comments
Rename/move Data Set	Full path	No	Ignored	Full path is provided by data set name. This path contains a new location and/or new name for the data set.
	Full path (only the terminal path node is used)	Yes	Ignored	Rename and/or move the data set. newDataSetName contains the data sets new name. Only the data set name is used from **newDataSetName**. **parentHandle** provides the rest of the path information. **parentHandle** points to either the that set's original or new location
	Terminal name only	No	Ignored	Invalid-cannot form a fully qualified path.
	Terminal name only	Yes	Ignored	**newDataSetName** provides the data set's new name and **parentHandle** provides the rest of the path information. **parentHandle** provides the rest of the path information. **parentHandle** points to either the that set's original or new location
	NULL	Yes	Terminal Name only	**parentHandle** points to a new location for the data set.
	NULL	Yes	Full path (only the terminal name is used)	**parentHandle** points to a new location for the data set.
Original location	NULL	No	Full path	Data set is restored to its original location and retains its original name
	NULL	No	Terminal name only	Invalid-cannot form a fully qualified path.
	NULL	Yes	Full path (only the terminal node is used)	Only the terminal name is used from data set's data. **parentHandle** points to the data set's original location.
	NULL	Yes	Terminal name only	The data set provides the terminal name only. **parentHandle** points to the data set's original location.

The open modes are used by NWSMTSWriteDataSet to determine how to write the data set (see).

See Also

Example

```
/*This example uses a log file built by the engine from information of
a previous backup session to restore the data sets.
The log file is used to get the names of the data set, find out if it
is a parent, and build the path to the data set
and contains the information from the NWSM_SCAN_INFORMATION and
NWSM_DATA_SET_NAME_LIST structures.
All parents were backed up with full paths and children were backed
with no path information.
All children of a parent were backed up before the next parent was
scanned.
Only one data set existed per transfer buffer.*/


#include <smsutapi.h>
#include <smstsapi.h>


UINT32 parentHandle = 0, mode = NWSM_OVERWRITE_DATA_SET,
dataSetHandle,
       maxTransferBufferSize, transferBufferSize;
/* For more informationabout maxTransferBufferSize and
transferBufferDataOffset see NWSMSDSessionOpenForWriting
in Storage Management Services Device Interface. */


NWBOOLEAN   checkCRC = TRUE, dontCheckSelectionList = TRUE;
BUFFERPTR transferBuffer;
NWSM_RECORD_HEADER_INFO recordHeaderInfo = {0};
/* Connect to a TSA and Target Service. */
/* Build the lists to display to the user for selection including
getting information about the Target Service type,
unsupported options, name space information, primary resource list,
open modes, scan types, and selection type.
See NWSMTSGetTargetServiceType, NWSMTSGetUnsupportedOptions,
NWSMTSListSupportedNameSpaces, NWSMTSListTSResources,
NWSMTSGetOpenModeOptionString, NWSMTSGetTargetScanTypeString, and
NWSMTSGetTargetSelectionTypeStr.*/
/* Connect to SMS DI. Subjugate a device and media, and mount the
media. See document Storage Device API. */
/* Have the user choose a session to restore. After the log is chosen
and opened, match the media information in the
log file against the one on the media to verify that we have the
correct session. */
/* Rewind the media, match the session information against the one on
the media, and open the session for reading by
using NWSMSDMediaPosition and NWSMSDSessionOpenForReading.
maxTransferBufferSize and transferBufferDataOffset is set by the last
function. */
/* Display the lists previously built to the user for selection and
```

```
specification. After the selections are made, package
the user's restore selections into selectionList and notify the TSA
what data sets to restore by calling
NWSMTSSetRestoreOptions. mode is set during this user interaction. */

NWSMTSSetRestoreOptions(connection, checkCRC, dontCheckSelectionList,
selectionList);
/* Retrieve the selected data sets and restore them. */
while(1)
{
/* Check if the first data set in the log file is to be restored.
First, from the information in the log construct a full
path for a data set by using NWSMPutFirstName and NWSMPutNextName.
The full path is put into newDataSetName. isParent is set from the
parentFlag field of the scan information kept
in the log file. */

   if(no more data sets)
      break;

   if (NWSMTSIsDataSetExcluded(connection, isParent, newDataSetName)
==
        NWSMTS_DATA_SET_EXCLUDED)
      cntinue;

/* We found a data set to restore. Move the media to the data set's
address, as indicated by the log file, and read the
transfer buffer from the media. NWSMSDMediaPosition and
NWSMSDSessionReadData can be used. */

   if (no transfer buffers)
      break;
/*Set up transfer buffer information. */
   transferBuffer = Transfer Buffer from the media;
   transferBuffer += transferBufferDataOffset;
/* Point to the Data Set Header or Subheader. */
   transferBufferSize = maxTransferBufferSize -
transferBufferDataOffset;
/* Extract the SIDF data from the transfer buffer. Only one SIDF data
set per transfer buffer exists and each data
set wholly occupies one transfer buffer. First, move the transfer
buffer pointer to the first record. */
while (transferBufferSize)
/* While there is data in the Transfer Buffer. */
   {
/*Get the next data set header or subheader from the Transfer Buffer.
*/
      NWSMGetRecordHeaderOnly(&transferBuffer, &transferBufferSize,
           &recordHeaderInfo);
/* If we have a record or subrecord, get the data from the Transfer
Buffer.
Notice that the second part of the if statement will be true if we have
gone through the loop more than once
(NWSMGetDataSetInfo sets recordHeaderInfo.dataSetInfoRetrieved to
```

```
DATA_SET_INFO_SPANNED). */
      if ((recordHeaderInfo.dataSetInfoRetrieved ==
DATA_SET_INFO_NOT_STARTED)
            || (recordHeaderInfo.dataSetInfoRetrieved ==
DATA_SET_INFO_SPANNED))
         NWSMGetDataSetInfo(&tranferBuffer, &transferBufferSize,
              &recordHeaderInfo);
      if (!transferBufferSize)
          break;
/* All data retrieved from current Transfer Buffer. Break and get the
next Transfer Buffer. */
      if (recordHeaderInfo.dataSetInfoRetrieved ==
DATA_SET_INFO_COMPLETE)
      {
/* Data in recordHeaderInof.dataSetName and
recordHeaderInfo.scanInformation can now be used. */
/* Restore the data set. Because of the way the data sets were backed
up, we need to use the parent handle to restore
children. No parent handles are needed for parents, because they
contain full path information. */
         if (recordHeaderInfo.scanInformation->parentFlag)
         {
            if (parentHandle) /* Close existing parent handle. */
               NWSMTSCloseDataSet(connection, &parentHandle);
            NWSMTSOpenDataSetForRestore(connection, 0, NULL, mode,
                 &parentHandle);
            dataSetHandle = 0;
         }

         else /* We have a child. */
         {
           NWSMTSOpenDataSetForRestore(connection, parentHandle, NULL,
mode,
                &dataSetHandle);
         }

         /* Now write the data to Target Service. */
        NWSMTSWriteDataSet(connection, dataSetHandle ? dataSetHandle :
              parentHandle, recordHeaderInfo.recordSize,
transferBuffer);

         /* Close the data set. */
         NWSMTSCloseDataSet(connection, &dataSetHandle);
      } /* end if DATA_SET_INFO_COMPLETE */

      /* Update the buffer's information. */
      transferBuffer += recordHeaderInfo.recordSize;
      transferBufferSize -= recordHeaderInfo.recordSize;
   } /* end while(transferBufferSize) */
} /* end while(1) */
free(recordHeaderInfo.scanInformation);
free(recordHeaderInfo.dataSetName);
```

NWSMTSSetRestoreOptions

Sets the restore options and defines the data sets to restore.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSSetRestoreOptions(
   UINT32                  connection,
   NWBOOLEAN               checkCRC,
   NWBOOLEAN               dontCheckSelectionList,
   NWSM_SELECTION_LIST  *selectionList);
```

Parameters

connection

   (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServicEx.

checkCRC

   (IN) Specifies the CRC verification flag:

   TRUE Check the data set's CRC
   FALSE Do not check the CRC

dontCheckSelectionList

   (IN) Specifies if the internal selection list should be checked.

selectionList

   (IN) Points to the list of data sets to restore.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFB	NWSMUT_OUT_OF_MEMORY
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER
0xFFFBFFFF	NWSMUT_INVALID_HANDLE
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFC9	NWSMTS_OUT_OF_MEMORY
0xFFFDFFDA	NWSMTS_INVALID_SEL_LIST_ENTRY
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL

0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

NWSMTSSetRestoreOptions clears the previous selection list, sets the TSA's internal restore selection list, and allows the engine to determine the mechanism used to find a data set to restore (see Set the Restore Options).

NWSMTSIsDataSetExcluded and NWSMTSWriteDataSet compare the names of the data sets to be restored against selectionList twice (once for each call). To prevent this double checking, set dontCheckSelectionList to TRUE. When the parameter is set to TRUE, only NWSMTSIsDataSetExcluded compares the data set name. This checking enhances data integrity, but decreases performance slightly. If no CRC was generated for the data set, no CRC check is performed.

If dontCheckSelectionList is set to FALSE, NWSMTSWriteDataSet will verify the data set's path information against the TSA's internal selection list. dontCheckSelectionList can be used with NWSMTSIsDataSetExcluded to speed up the restore process.

Each entry in selectionList contains the names or name patterns of the data sets to restore. If NULL is passed, all data sets are restored (see Using Resources with Selection Options and Documents). The "Data Set Name Functions" described in *Storage Management Services Library* can be used to help create this list.

Each time NWSMTSSetRestoreOptions is called, the previous selectionList will be replaced. NWSMTSSetRestoreOptions rebuilds the resource list which was built at the time of connection. The list is rebuilt because a resource may have been mounted or dismounted, or the resource's name space information may have changed. To rebuild the resource list, the engine must set selectionList to NULL and call NWSMTSSetRestoreOptions or NWSMTSBuildResourceList.

NOTE: The bindery must be explicitly included in the restore session or it will not be restored.

See Also

NWSMTSIsDataSetExcluded (page 144), NWSMTSOpenDataSetForRestore (page 146), NWSMTSWriteDataSet (page 154)

NWSMTSWriteDataSet

Writes a data set to the Target Service.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSWriteDataSet (
   UINT32      connection,
   UINT32      dataSetHandle,
   UINT32      bytesToWrite,
   BUFFERPTR   buffer);
```

Parameters

connection

   (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServicEx.

dataSetHandle

   (IN) Specifies the handle returned by NWSMTSOpenDataSetForRestore.

bytesToWrite

   (IN) Specifies the number of bytes to write.

buffer

   (IN) Points to the caller allocated buffer containing the data to be written back through the TSA. The contents of the buffer should have been obtained previously from the NWSMTSReadDataSet call during backup.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFAF	NWSMTS_CLUSTER_TARGET_DOES_NOT_EXIST
0xFFFDFFB5	NWSMTS_WRITE_ERROR
0xFFFDFFB6	NWSMTS_WRITE_ERROR_SHORT
0xFFFDFFB7	NWSMTS_WRITE_EA_ERROR
0xFFFDFFB8	NWSMTS_VALID_PARENT_HANDLE
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFC4	NWSMTS_SCAN_ERROR
0xFFFDFFC6	NWSMTS_READ_ERROR

0xFFFDFFC8	NWSMTS_OVERFLOW
0xFFFDFFC9	NWSMTS_OUT_OF_MEMORY
0xFFFDFFCA	NWSMTS_OUT_OF_DISK_SPACE
0xFFFDFFCC	NWSMTS_OPEN_ERROR
0xFFFDFFDC	NWSMTS_INVALID_PATH
0xFFFDFFE5	NWSMTS_INVALID_DATA_SET_HANDLE
0xFFFDFFE6	NWSMTS_INVALID_DATA
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFDFFEC	NWSMTS_GET_ENTRY_INDEX_ERR
0xFFFDFFEF	NWSMTS_EXPECTING_TRAILER
0xFFFDFFF0	NWSMTS_EXPECTING_HEADER
0xFFFDFFF4	NWSMTS_DATA_SET_IS_OLDER
0xFFFDFFF5	NWSMTS_DATA_SET_IN_USE
0xFFFDFFF7	NWSMTS_DATA_SET_EXCLUDED
0xFFFDFFF8	NWSMTS_DATA_SET_ALREADY_EXISTS
0xFFFDFFF9	NWSMTS_CREATE_ERROR
0xFFFDFFFA	NWSMTS_CREATE_DIR_ENTRY_ERR
0xFFFDFFFB	NWSMTS_CLOSE_BINDERY_ERROR
0xFFFDFFFC	NWSMTS_CANT_ALLOT_DIR_HANDLE
0xFFFDFFFD	NWSMTS_BUFFER_UNDERFLOW
0xFFFDFFFF	NWSMTS_ACCESS_DENIED
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Engine Developers

NWSMTSWriteDataSet deformats the data set data and writes it out to the Target Service. The FID functions described in *Storage Management Services Utilities Library* can be used to deformat the data set data.

Remarks

Before NWSMTSWriteDataSet is called, NWSMTSOpenDataSetForRestore must be called to open the data set.

NWSMTSWriteDataSet must be called repeatedly until all of the data is written to Target Services.

NWSMTSWriteDataSet writes the data set to the specified location. If the data spans many buffers, the engine will have to call NWSMTSWriteDataSet once for every buffer.

A special situation occurs when the restore mode is set to NWSM_DO_NOT_OVERWRITE_DATA_SET, and the data set spans multiple buffers. NWSMTSWriteDataSet must parse through the data to find the path information which may require sifting through multiple buffers. NWSMTSWriteDataSet can then determine if the data set exists on the Target Service.

NWSMTSWriteDataSet may not return NWSMTS_VALID_PARENT_HANDLE or NWSMTS_DATA_SET_ALREADY_EXISTS (indicating the data set exists) until it is called many times (see Invalid, Valid Handle).

NWSM_VALID_PARENT_HANDLE is returned when enough data has been received to create a valid parent handle and indicates that:

- ◆ The parent already exists on the Target Service
- ◆ The parent was not overwritten
- ◆ A parent handle was created for the existing parent
- ◆ The parent's children can be restored

If buffer contains fields that the TSA does not recognize, the fields are ignored.

NOTE: The bindery must be explicitly included in the restore session or it will not be restored.

See Also

NWSMTSOpenDataSetForRestore (page 146), NWSMTSSetRestoreOptions (page 152)

Example

See NWSMTSOpenDataSetForRestore (page 146) for an example.

Target Services Structures

4

This documentation alphabetically lists the Target Services structures and describes their purpose, syntax, and fields.

Data Set Name List

Lists the data sets to scan for or the data set just scanned in conjunction with NWSM_DATA_SET_NAME_LIST.

Syntax

```
UINT32   nameSpaceType;
UINT32   reserved;
UINT8    count;
UINT16   namePositions[count];
UINT16   separatorPositions[count];
UINT16   nameLength;
STRING   name[nameLength];
```

Fields

nameSpaceType

Specifies the name space type of the *name* (see "nameSpaceType Values" on page 347)

reserved

Is reserved for future use.

count

Specifies the size of namePositions and separatorPositions.

namePositions

Specifies the beginning position of each path node in name.

separatorPositions

Specifies the beginning position of each separator name.

nameLength

Specifies the length of name including the NULL terminator.

name

Specifies a NULL-terminated or fully qualified path string.

Remarks

The values that can set for nameSpaceType is listed in the following table.

Value	Description
0xFFFFFFFE	NWSM_TSA_DEFINED_RESOURCE_TYPE: name contains a primary resource.
0xFFFFFFFC	NWSM_DIRECTORY_NAME_SPACE: name contains a directory service path.
0x000	DOSNameSpace: name contains a DOS path in MBCS format.
0x001	MACNameSpace: name contains a Macintosh path in MBCS format.

Value	Description
0x002	NFSNameSpace: name contains an NFS path in MBCS format.
0x003	FTAMNameSpace: name contains an FTAM path in MBCS format.
0x004	OS2NameSpace: name contains an OS/2 path in MBCS format.
0x100	DOSNameSpaceUtf8Type: **name** contains a DOS path in UTF-8 format.
0x101	MacNameSpaceUtf8Type: **name** contains a Macintosh path in UTF-8 format.
0x102	NFSNameSpaceUtf8Type: **name** contains an NFS path in UTF-8 format.
0x104	LongNameSpaceUtf8Type: **name** contains a DOS path in UTF-8 format.

If a data set is a primary resource, nameSpaceType must be set to NWSM_TSA_DEFINED_RESOURCE_TYPE.

If count is zero, namePositions and separatorPositions are not used.

For more information on name positions, see Path Information and Using Resources with Selection Options. Although the name list example is a selection list, the same principles apply.

name will have a separator at the end for a parent terminal node.

NOTE: *name* must be properly formatted when passed to NWSMTSIsDataSetExcluded.

NWSM_DATA_SET_NAME_LIST

Contains the data set's path as it appears under one of more of the name spaces supported by the Target Service.

Syntax

```
typedef struct
{
   UINT16   bufferSize;
   UINT16   dataSetNameSize;
   UINT8    nameSpaceCount;
   UINT8    keyInformationSize;
   UINT8    keyInformation[keyInformationSize];
} NWSM_DATA_SET_NAME_LIST;
```

Fields

bufferSize

> Specifies the actual buffer size allocated to NWSM_DATA_SET_NAME_LIST.

dataSetNameSize

> Specifies the actual memory space used by the data set name list (excluding bufferSize).

nameSpaceCount

> Specifies the number of name spaces in the data set name list.

keyInformationSize

> Specifies the size of keyInformation in bytes.

keyInformation

> Is reserved for future use to specify key encryption information.

Remarks

NWSM_DATA_SET_NAME_LIST consists of two parts: a list information structure and the data set name list.The list contains the data set names and is appended to the end of the structure.

To access the data set name, see "Data Set Name Functions" in *Storage Management Services Utilities Library* and can be used to help transfer the data set's name into or out of the data set name list.

See NWSM_DATA_SET_NAME_LIST (page 160) for the members and definitions of the Data Set Name List.

NWSM_NAME_LIST

Builds a linked list of names that can be freed by calling NWSMFreeNameList.

Syntax

```
typedef struct _NWSM_NAME_LIST
{
    _NWSM_NAME_LIST  *next;
    STRING            name;
    void             *other_info;
} NWSM_NAME_LIST;
```

Fields

next

   Points to the next element in the list or to NULL (indicating the end of the list).

name

   Specifies a NULL-terminated string indicating the name space number and the associated name space.

other_info

   Points to an engine-defined value or to a structure.

Remarks

The values that can be set to name is listed in the following table, and the first four bytes of name contain the name space number.

Value	Description
0xFFFFFFFF	NWSM_ALL_NAME_SPACES: Specifies all name spaces
0xFFFFFFFE	NWSM_TSA_DEFINED_RESOURCE_TYPE: Specifies a TSA-defined resource
0xFFFFFFFD	NWSM_CREATOR_NAME_SPACE: Specifies the name of the name space that created the data set
0xFFFFFFFC	NWSM_DIRECTORY_NAME_SPACE: Specifies the directory services string
0x0	DOSNameSpace: Specifies the DOS name space
0x1	MACNameSpace: Specifies the Macintosh name space
0x2	NFSNameSpace: Specifies the NFS name space
0x3	FTAMNameSpace: Specifies the FTAM name space
0x4	OS2NameSpace: Specifies the OS/2 name space

NWSM_SCAN_CONTROL

Defines the characteristics and attributes of the data sets to scan for as well as the path information to return in conjunction with NWSM_SELECTION_LIST.

Syntax

```
typedef struct
{
   UINT16  bufferSize;
   UINT16  scanControlSize;
   UINT32  scanType;
   UINT32  firstAccessDateAndTime;
   UINT32  lastAccessDateAndTime;
   UINT32  firstCreateDateAndTime;
   UINT32  lastCreateDateAndTime;
   UINT32  firstModifiedDateAndTime;
   UINT32  lastModifiedDateAndTime;
   UINT32  firstArchivedDateAndTime;
   UINT32  lastArchivedDateAndTime;
   UINT8   returnChildTerminalNodeNameOnly;
   UINT8   parentsOnly;
   UINT8   childrenOnly;
   UINT8   createSkippedDataSetsFile;
   UINT8   generateCRC;
   UINT8   returnNFSHardLinksInDataSetName;
   UINT8   reserved[6];
   UINT32  scanChildNameSpaceType;
   UINT32  returnNameSpaceType;
   UINT8   callScanFilter;
   UINT16  otherInformationSize;
   UINT8   otherInformation[otherInformationSize];
} NWSM_SCAN_CONTROL;
```

Fields

bufferSize

　　Specifies the structure size (including memory allocated for otherInformation).

scanControlSize

　　Specifies the memory used by all the structure fields excluding bufferSize.

scanType

　　Specifies a bit map indicating the selected scan type options.

firstAccessDateAndTime

　　Specifies the DOS-packed date and time criteria to scan for.

lastAccessDateAndTime

　　Specifies the DOS-packed date and time criteria to scan for.

firstCreateDateAndTime

>   Specifies the DOS-packed date and time criteria to scan for.

lastCreateDateAndTime

>   Specifies the DOS-packed date and time criteria to scan for.

firstModifiedDateAndTime

>   Specifies the DOS-packed date and time criteria to scan for.

lastModifiedDateAndTime

>   Specifies the DOS-packed date and time criteria to scan for.

firstArchivedDateAndTime

>   Specifies the DOS-packed date and time criteria to scan for.

lastArchivedDateAndTime

>   Specifies the DOS-packed date and time criteria to scan for.

returnChildTerminalNodeNameOnly

>   Specifies if only the name of the child will be returned by the scan:

>   TRUE Return only the name for a child and full path for a parent
>   FALSE Always return the full path for a child or parent

parentsOnly

>   Specifies if only parents will be scanned:

>   TRUE Scan only parents
>   FALSE Scan both parents and children

childrenOnly

>   Specifies if only children will be scanned:

>   TRUE Scan only children
>   FALSE Scan both children and parents

createSkippedDataSetsFile

>   Specifies if the TSA will create a log of the data sets that qualified for the scan but were skipped:

>   TRUE Create a log file
>   FALSE Do not create a log file

generateCRC

>   Specifies if a CRC value will be generated for each data set:

>   TRUE Create a CRC value
>   FALSE Do not create a CRC value

returnNFSHardLinksINDataSetName

>   Specifies if the data set's NFS hard links will be returned in dataSetName:

TRUE dataSetName contains the returned NFS hard links
FALSE NFS hard links will not be returned

reserved

Is reserved for future use.

scanChildNameSpaceType

Specifies the name space of the child data sets to scan for-usually set to NWSM_ALL_NAME_SPACES (see Returned Name Space Type).

returnNameSpaceType

Specifies the name space types that the data set's name will be returned in.

callsScanFilter

Specifies if the TSA will use the specified scan filter:

TRUE Use the specified scan filter
FALSE Do not use the specified scan filter

otherInformationSize

Specifies the size of otherInformation.

otherInformation

Specifies developer-specific information.

Remarks

bufferSize may need to be larger than the structure because otherInformation may vary in size. To prevent the buffer size from being reallocated often, add extra space to bufferSize.

scanType is graphically represented by a 32-bitmap: bit 0-6 represents the predefined scan types and bit 7-31 represents the TSA specific scan types

Figure 4-1   *Scan Type Bitmap*
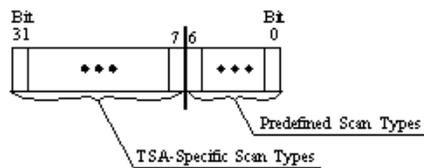
For Target Services that support purging deleted files, set scanType to NWSM_PURGE_IMMED_ON_DELETE before calling NWSMTSScanDataSetBegin.

The following table show the scan type that each TSA supports:

Value	Description
0x0001	NWSM_DO_NOT_TRAVERSE: Do not traverse the file system tree (see Scan Settings for more information).

Value	Description
0x0002	NWSM_EXCLUDE_ARCHIVED_CHILDREN: Do not scan for children data sets whose archive flag is set.
0x0004	NWSM_EXCLUDE_HIDDEN_CHILDREN: Do not scan for children data sets whose hidden flag is set.
0x0008	NWSM_EXCLUDE_HIDDEN_PARENTS: Do not scan for parent data sets whose hidden flag is set.
0x0010	NWSM_EXCLUDE_SYSTEM_CHILDREN: Do not scan for children data sets whose system flag is set.
0x0020	NWSM_EXCLUDE_SYSTEM_PARENTS: Do not scan for parent data sets whose system flag is set.
0x0040	NWSM_EXCLUDE_CHILD_TRUSTEES: Do not read the trustee information of children data sets.
0x0080	NWSM_EXCLUDE_PARENT_TRUSTEES: Do not read the trustee information of parent data sets.
0x0100	NWSM_EXCLUDE_ACCESS_DATABASE: Do not scan the database. For example, back up of server specific info from NetWare 4.x onwards.
0x0200	NWSM_EXCLUDE_VOLUME_RESTS: Do not read the resource restriction information.
0x0400	NWSM_EXCLUDE_DISK_SPACE_RESTS: Do not read the disk space restriction information.
0x0800	NWSM_EXCLUDE_EXTENDED_ATTRIBUTS: Do not read the extended attribute information.
0x1000	NWSM_EXCLUDE_DATA_STREAMS: Do not read a data set's data stream.
0x2000	NWSM_EXCLUDE_MIGRATED_CHILD: Do not read migrated data streams of children data sets. Read only the stub information for these migrated children.
0x4000	NWSM_EXPAND_COMPRESSED_DATA: Expand the data set before scanning it.
0x8000	NWSM_EXCLUDE_ARCH_CHILD_DATA: Do not scan the data of children that have been archived.
0x10000	NWSM_EXCLUDE_ARCH_CHILD_CHAR: Do not scan the child's characteristics if the child's characteristics archive bit is set.
0x20000	NWSM_FLAG_PURGE_IMMED_ON_DELETE: Set the data set's purge flag when it is deleted.
0x40000	NWSM_EXCLUDE_MIGRATED_FILES: Do not scan for children whose remote data access bit is set.
0x80000	NWSM_INCLUDE_PATH_COMPONENT: For each item in the include list, backup the individual parent components before processing the data sets.
0x100000	NWSM_EXCLUDE_HARDLINK_DATA : Do not backup the data for hardlinks except for the first encountered data node.
0X200000	NWSM_EXCLUDE_SECONDARY_DATA_STREAMS : Do not read a data set's secondary data stream.

Value	Description
0x400000	NNWSM_INCLUDE_SOFTLINK_CHILD : Reads the soft link along with real data of children data set's.
0X800000	NWSM_OR_DATE_TIME_FILTER: Backup the data, if the data sets date and time values are within the range specified in the date and time fields.

The TSA specifies which other scan types to enable and disable when a scan type is chosen. The TSA flags can be ORed to produce the desired action. However, the combination of various scan types can produce an illegal condition or unwanted results. The TSA will return an error indicating that an illegal condition might exist, but it cannot signal that an undesirable result might occur.

The date and time fields specify the range for the scan criteria. A data set meets the date and time criteria if all the data set's date and time values are within the range specified by all the date and time fields. Zero indicates an open-ended scan for that kind of date and time information (see Date and Time Fields for more information). These fields contain DOS-packed values and can be packed or unpacked by calling the "DOS Date and Time functions" shown in *Storage Management Services Utilities Library*. Some TSAs may not support all of the date and time fields (see NWSMTSGetUnsupportedOptions (page 127)).

The following table gives an example of the scan type settings and indicates the results of each scan

firstAccessDateAndTime	lastAccessDateAndTime	Description
October 12, 1985 12:00:00	April 1, 1989 12:00:00	Returns information on all files accessed between October 12, 1985 12:00:00 and April 1, 1989 12:00:00 inclusively.
0	April 1, 1989 12:00:00	Returns information on all files accessed until and including April 1, 1989 12:00:00.
October 12, 1985 12:00:00	0	Returns information on all files accessed since October 12, 1985 12:00:00.
0	0	Returns information about all files.

TSA support for returnChildTerminalNodeNameOnly is indicated by calling NWSMTSGetUnsupportedOptions.

If returnChildTerminalNodeNameOnly is set to TRUE, NWSMTSScanDataSetBegin returns the fully qualified path of the parent of the first child data set found. For the name of the first child data set, call NWSMTSScanNextDataSet (the path information is provided by the parent).

childrenOnly must be set to false if *returnChildTerminalNodeName* is set to TRUE (see Scan Settings for more information).

TSA support for createSkippedDataSetsFile can be determined by calling NWSMTSGetUnsupportedOptions (page 127) (see Log Files and NWSMTSScanDataSetBegin (page 38) for more information).

generateCRC enhances data integrity, but decreases the TSA's performance slightly if set to TRUE.

The values that can be set for returnNameSpaceType is listed in the following table.

Value	Description
0xFFFFFFFF	NWSM_ALL_NAME_SPACES: Return all the name spaces of the data set in Multi Byte Character Sets (MBCS) format.
0xFFFFFFFE	NWSM_TSA_DEFINED_RESOURCE_TYPE: Return the data set if it is a TSA defined resources.
0xFFFFFFFD	NWSM_CREATOR_NAME_SPACE: Return only the created name space of the data set in Multi Byte Character Sets (MBCS) format.
0xFFFFFFFC	NWSM_DIRECTORY_NAME_SPACE: Return the data set if it is a Directory Services data set.
0xFFFFFFFAL	NWSM_ALL_NAME_SPACES_UTF8: Return all the supported name spaces of the data set in UTF-8 format.
0xFFFFFFF9L	NWSM_CREATOR_NAME_SPACE_UTF8: Return only the created name space of the data set in UTF-8 format.
0xFFFFFFF8L	NWSM_ALL_NAME_SPACES_FORMATS: Return all the name spaces of the data set in all supported formats.
0x000	DOSNameSpace: Return only for DOS data set in MBCS format.
0x001	MACNameSpace: Return only for Mac data sets in MBCS format.
0x002	NFSNameSpace: Return only for NFS data sets in MBCS format.
0x003	FTAMNameSpace: Return only for FTAM data sets in MBCS format.
0x004	OS2NameSpace: Return only for OS/2 data sets in MBCS format.
0x100	DOSNameSpaceUtf8Type: Return only for DOS data sets in UTF-8 format
0x101	MACNameSpaceUtf8Type: Return only for Mac data sets in UTF-8 format
0x102	NFSNameSpaceUtf8Type: Return only for NFS data sets inUTF-8 format
0x104	LONGNameSpaceUtf8Type: Return only for LONG data sets in UTF-8 format

TSA defined resources are logical resource groupings. Logical groupings refers to the fact that a bindery or file server does not exist as a single entity but as a logical group of lesser resources. The bindery consists of 3 files while the file server consists of: the bindery, volume, directories, and files.

callScanFilter is not supported under NetWare 4.

NWSM_SCAN_INFORMATION

Contains information about one data set.

Syntax

```
typedef struct
{
   UINT16   bufferSize;
   UINT16   scanInformationSize;
   UINT32   attributes;
   UINT32   creatorID;
   UINT32   creatorNameSpaceNumber;
   UINT32   primaryDataStreamSize;
   UINT32   totalStreamsDataSize;
   UINT8    modifiedFlag;
   UINT8    deletedFlag;
   UINT8    parentFlag;
   UINT8    reserved[5];
   UINT32   accessDateAndTime;
   UINT32   createDateAndTime;
   UINT32   modifiedDateAndTime;
   UINT32   archivedDateAndTime;
   UINT16   otherInformationSize;
   UINT8    otherInformation[otherInformationSize];
} NWSM_SCAN_INFORMATION;
```

Fields

bufferSize

Specifies the actual buffer size allocated for NWSM_SCAN_INFORMATION.

scanInformationSize

Specifies the total memory used by NWSM_SCAN_INFORMATION (excluding bufferSize).

attributes

Specifies the data set's attributes.

creatorID

Specifies the ID of the entity who created the data set.

creatorNameSpaceNumber

Specifies the number of the name space that created the data set.

primaryDataStreamSize

Specifies the size of the primary data stream (not the size of the data set).

totalStreamsDataSize

Specifies the total number of 4 KB blocks used to contain all of the data set's data streams (not the data set's size).

modifiedFlag

Specifies if the data set was modified since the last backup:

TRUE Data set was modified
FALSE Data set was not modified

deletedFlag

Specifies if the data set was deleted:

TRUE Data set was deleted
FALSE Data set was not deleted

parentFlag

Specifies if the data set is a parent:

TRUE Data set is a parent
FALSE Data set is not a parent

reserved

Is reserved for future use.

accessDateAndTime

Specifies the data set's last accessed date and time in DOS-packed format.

createDateAndTime

Specifies the data set's created date and time in DOS-packed format.

modifiedDateAndTime

Specifies the data set's last modified date and time in DOS-packed format.

archivedDateAndTime

Specifies the data set's last archived date and time in DOS-packed format.

otherInformationSize

Specifies the size of otherInformation.

otherInformation

Is reserved for future use to specify TSA developer-specific information.

Remarks

The information returned by NWSM_SCAN_INFORMATION depends upon the settings of NWSM_SCAN_CONTROL, NWSM_SELECTION_LIST, and the options supported by the TSA (see Unsupported Backup Options). This information may be used for backup logs and determining if the data exists before restoring it.

Not all TSAs support all the fields of NWSM_SCAN_INFORMATION. All unsupported fields are zeroed out by the TSA. NWSM_SCAN_INFORMATION and NWSM_DATA_SET_NAME_LIST represents the only information available to the engine about the data set.

NOTE: For users of SIDF, the scan control information and the data set's name information are known as the data set information.

The values that can be set to attributes is listed in the following table.

Option	Value	Description
None	0x00000000	NWFA_NORMAL: The data set has its normal attributes set.
0	0x00000001	NWFA_READ_ONLY: The data set is marked as read only.
1	0x00000002	NWFA_HIDDEN: The data set is marked as hidden.
2	0x00000004	NWFA_SYSTEM: The data set is marked as a system file.
3	0x00000008	NWFA_EXECUTE_ONLY: The data set is marked as an executable.
4	0x00000010	NWFA_DIRECTORY: The data set is a parent.
5	0x00000020	NWFA_ARCHIVE: The data set's archive bit.
6		Reserved
7	0x00000080	NWFA_SHARABLE: The data set is marked as sharable.
8 - 11		Reserved
12	0x00001000	NWFA_TRANSACTION: The data set is part of a transaction in progress.
13		Reserved
14	0x00004000	NWFA_READ_AUDIT: The data set is marked for read auditing.
15	0x00008000	NWFA_WRITE_AUDIT: The data set is marked for write auditing.
16	0x00010000	NWFA_PURGE: The data set marked to be purged.
17	0x00020000	NWFA_RENAME_INHIBIT: The data set cannot be renamed.
18	0x00040000	NWFA_DELETE_INHIBIT: The data set cannot be deleted.
19	0x00080000	NWFA_COPY_INHIBIT: The data set cannot be copied.
20	0x00100000	NWFA_FILE_AUDITING: The data set's file auditing flag.
21		Reserved
22	0x00400000	NWFA_REMOTE_DATA_ACCESS: The data set can be remotely accessed.
23	0x00800000	NWFA_REMOTE_DATA_INHIBIT: The data set cannot be remotely accessed.
24	0x01000000	NWFA_REMOTE_DATA_SAVE_KEY_BIT: The data was migrated to tertiary storage. This bit applies only to child data sets.
25	0x02000000	NWFA_COMPRESS_FILE_IMMEDIATE: The data set is not yet compressed.
26	0x04000000	NWFA_DATA_STREAM_IS_COMPRESSED: The data set is compressed.
27	0x08000000	NWFA_DO_NOT_COMPRESS_FILE: The data cannot be compressed.

Option	Value	Description
28		Reserved
29	0x20000000	NWFA_CANT_COMPRESS_DATA: The data set's data cannot be compressed.
30	0x40000000	NWFA_OBJ_ARCHIVE_BIT: One or more of the following has been changed since the last backup: extended attributes, owner ID, or trustees.

creatorID is not generic and is defined by the TSA. For NetWare, it is the bindery ID.

creatorNameSpaceNumber has a corresponding name space name string that is returned by NWSMTSListSupportedNameSpaces. This number is always returned even if the creator name space type was not requested by field returnNameSpaceType of NWSM_SCAN_CONTROL.

Call NWSMTSGetUnsupportedOptions (page 127) to see if the TSA supports accessDateAndTime, createDateAndTime, modifiedDateAndTime, and archivedDateAndTime.

Support for Larger Files (file size more than 4 GB)

The NSS file system supports files of sizes more than 4 GB. The primaryDataStreamSize returns the first double word (low order four bytes) of the file size. The file system TSA returns the second double word as part of the otherInformation while setting the otherInformationSize to 4.

Support for Scan Information Extensions

Different TSAs may have meta data regarding data sets that do not fit into the existing definition of the scan information structure. Such information is returned as extensions in the otherInformation field. The otherInformationSize is set to the number of bytes of extension information that is returned in this case. This size and the otherInformation buffer can be used with the extension APIs to process this additional meta data. See Section 6.3, "Extension Functions," on page 218. The extensions and their fields are detailed in Section 5.7, "Extensions," on page 182.

NOTE: As the otherInformation field is already extended to contain the second double word to represent QUAD file sizes, extensions are appended after the first double word when present. Use otherInformationSize after subtracting the size of a double word to ensure that there are extensions encoded in otherInformation.

NWSM_SELECTION_LIST

Contains a list of data set names, encryption key information, and search patterns to scan for in conjunction with NWSM_SCAN_CONTROL.

Syntax

```
typedef struct
{
   UINT16         bufferSize;
   UINT16         selectionListSize;
   UINT8          selectionCount;
   UINT8          keyInformationSize;
   UINT8          keyInformation[keyInformationSize];
} NWSM_SELECTION_LIST;
```

Fields

bufferSize

Specifies the total buffer size allocated.

selectionListSize

Specifies the actual space used by the last four fields of NWSM_SELECTION_LIST as well as the selection list.

selectionCount

Specifies the number of entries in the selection list.

keyInformationSize

Specifies the size of keyInformation in bytes.

keyInformation

Is reserved for future use to specify the key encryption information used to encrypt the selection list.

Remarks

The selection list itself is not part of the structure declaration as shown in the following graphic. For information about the usage of the structure and the list, see Selection Options.

Figure 4-2   *Selection List*
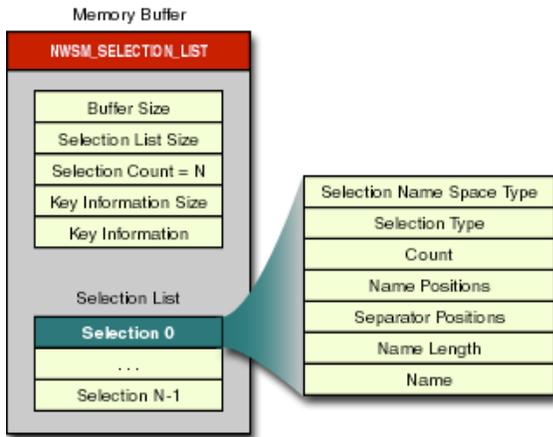
See Selection List for the members and definitions of the selection list.

Selection List

Contains user-specified data set names to perform a scan against in conjunction with NWSM_SELECTION_LIST.

Syntax

```
UINT32   selectionNameSpaceType;
UINT32   selectionType;
UINT8    count;
UINT16   namePositions[count];
UINT16   separatorPositions[count];
UINT16   nameLength;
STRING   name[nameLength];
```

Immediately following keyInformation of NWSM_SELECTION_LIST is an array of user-specified data set names (the selection list). The list is used to perform some sort of action against each element(selection) in the list has a selection type, which specifies how the data set is to be selected (e.g., included in or excluded from a scan).

Fields

selectionNameSpaceType

   Specifies the name space type used by name (usually the name space that created the data set).

selectionType

   Specifies how to select data.

count

   Specifies the size of namePositions and separatorPositions.

namePositions

   Specifies the beginning position of each path node contained in name.

separatorPositions

   Specifies the beginning position of each separator in name.

nameLength

   Specifies the length of name including NULL.

name

   Specifies a NULL-terminated path string.

Remarks

The following table lists the values that can be set for typeNumber used by NWSMTSGetTargetSelectionTypeStr and the mask information used in selectionType.

Option	Value	Description
0	0x0000	NONE: Cannot be used if a selection list is being built. If a data set name list is being built, selectionType must be "No selection type".
1	0x0002	NWSM_TSA_DEFINED_RESOURCE_EXC: Do not scan for the specified TSA-defined resources, including all of its' subordinates.
1	0x0003	NWSM_TSA_DEFINED_RESOURCE_INC: Scan for the specified TSA-defined resource, including all of its subordinates. All TSA-defined resources are included in the scan by default if the resource name passed to NWSMTSScanDataSetBegin is the first resource returned by NWSMTSListTSResources.
2	0x0004	NWSM_PARENT_TO_BE_EXCLUDED: Do not scan for the specified parents, including all of its subordinates from the scan.
2	0x0005	NWSM_PARENT_TO_BE_INCLUDED: Scan for the specified parents, including all of its subordinates from the scan.
3	0x0008	NWSM_CHILD_TO_BE_EXCLUDED: Do not scan for the specified children.
3	0x0009	NWSM_CHILD_TO_BE_INCLUDED: Scan for the specified children.
4	0x0010	NWSM_EXCLUDE_CHILD_BY_FULL_NAME: Do not scan for the specified child.
4	0x0011	NWSM_INCLUDE_CHILD_BY_FULL_NAME: Do not scan for the specified child.

All selection types (except the last two) can be applied globally to the target's data. NWSM_EXCLUDE_CHILD_BY_FULL_NAME and NWSM_INCLUDE_CHILD_BY_FULL_NAME can be applied to only one location on the target. However, the engine must manually set the boundaries of the area to select data sets from. For example, to include all .c files in directory /home/user/dir1, the engine must put into a selection list the following two list elements:

- ◆ For the first element, set selectionType to NWSM_INCLUDE_CHILD_BY_FULL_NAME and set name to /home/user/dir1/*.c

- ◆ For the second element, set selectionType to NWSM_PARENT_TO_BE_INCLUDED and name to /home/user/dir1

Call NWSMTSGetTargetSelectionTypeStr to find out if a selection type is supported.

NWSM_TSA_DEFINED_RESOURCE_EXC and NWSM_TSA_DEFINED_RESOURCE_INC are logical groupings of resources.

TSA-defined resources are logical groupings of resources. NetWare's bindery and file servers are examples of these resource since they do not exist as a single entity, but as a logical group of lesser resources. The bindery consists of 3 files. The file server consists of a bindery, a volume, directories, and files.

name must contain a fully qualified path for selection types, TSA-defined resources, and full-name parents and children.

name only contains a terminal name for NWSM_CHILD_TO_BE_EXCLUDED and NWSM_CHILD_TO_BE_INCLUDED.

Wild cards are only allowed in the terminal path node (even if the node is a parent). If the terminal path node is a parent, and if the name space requires it, an end separator is needed.

Call NWSMPutFirstName, NWSMPutNextName, NWSMPutOneName, and NWSMCloseName to insert the path information into a selection list. See WNSMTSGetTargetSelectionTypeStr for an example of these functions.

Example

Assume a user includes the following data sets in a backup session:

On Linux:

/nssvol/home/admin

where /nssvol/home/admin is a full path to a parent. These are the relevant fields of NWSM_SELECTION_LIST and the Selection List:

```
selection 1:
    selectionType = NWSM_PARENT_TO_BE_INCLUDED
    count = 3
    namePositions = 1, 8, 13
    separatorPositions = 0, 7, 12
    nameLength = 18
    name[nameLength] = "/nssvol/home/admin" + '\0'
```

On NetWare:
```
SYS:SYSTEM/PROTO/
SYS:PUBLIC/BIN/PCONSOLE.EXE
BINDERY
Employee.Department.Company.Country
```

where SYS:SYSTEM/PROTO/ is a full path to a parent, SYS:PUBLIC/PCONSOLE.EXE is a full path to a child, BINDERY is a TSA defined resource, and Employee.Department.Company.Country is a path to a Directory Services object.

These are the relevant fields of NWSM_SELECTION_LIST and the Selection List:
```
selectionCount = 3
selection 1:
    selectionType = NWSM_PARENT_TO_BE_INCLUDED
    count = 3
    namePositions = 0, 4, 11
    separatorPositions = 3, 10, 16
    nameLength = 18
    name[nameLength] =  "SYS:SYSTEM/PROTO/" + '\0'
selection 2:
    selectionType = NWSM_CHILD_TO_BE_INCLUDED
    count = 4
    namePositions = 0, 4, 11, 15
    separatorPositions = 3, 10, 14, 0
    nameLength = 28
    name[nameLength] =
        "SYS:PUBLIC/BIN/PCONSOLE.EXE" + '\0'
selection 3:
    selectionType = NWSM_TSA_DEFINED_RESOURCE
```

```
        count = 0
        namePositions = n/a
        separatorPositions = n/a
        nameLength = 8
        name[nameLength] = "BINDERY" + '\0'
selection 4:
        selectionType = NWSM_DIRECTORY_NAME_SPACE
        count = 4
        namePositions = 27, 20, 9, 0
        separatorPositions = 0, 26, 19, 8
        nameLength = 8
        name[nameLength] = Employee.Department.Company.Country +'\0'
```

The name position and separator position array size are always equal, regardless of the number of
path nodes and separators in the path. Selection two shows that the last value of the separator
position array is zero which indicates that there is no last separator and that the path contains a child.
Selection three shows that the name position and separator position arrays are not used because the
path contains only one node. Selection four shows a reversed path. The indices in both position
arrays are reversed because the first index value must indicate the beginning of the most significant
path node or separator and the second index value must indicate the next significant path node or
separator.

STRING_BUFFER

Syntax

```
typedef struct
{
   UINT16   size;
   char     string[1];
} STRING_BUFFER;
```

Fields

size

   Specifies the amount of memory (in bytes) allocated for string.

string

   Specifies an array of one character.

Remarks

string can be declared as a NULL-terminated string array as follows:

```
#define SIZE 32

STRING_BUFFER *s;

s = (STRING_BUFFER*)malloc(sizeof(STRING_BUFFER)+SIZE);
s->size = SIZE + 1;
strcpy(s->string, string);
```

UINT16_BUFFER

Is used by NWSMTSParseDataSetName.

Syntax

```
typedef struct
{
   UINT16   size;
   UINT16   buffer[1];
} UINT16_BUFFER;
```

Fields

size

   Specifies the size (in bytes) of buffer.

buffer

   Specifies user-specific data.

Remarks

buffer can be declared as a variable-length array of unsigned integers as follows:
```
#define ARRAY_SIZE 10

UINT16_BUFFER *i;

i=(UINT16_BUFFER
*)malloc(sizeof(UINT16_BUFFER)+(sizeof(UINT16)*ARRAY_SIZE));
i->size=sizeof(UINT16)*ARRAY_SIZE+1;
```

Utility Library Concepts

5

This documentation describes Utility Library, its functions, and features.

5.1 DOS Date and Time Format

The DOS date and time value is an unsigned 32-bit value divided into six fields as follows:
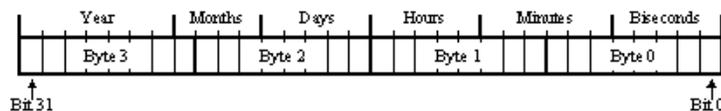
The legal values for each field is listed in the following table.

Field	Value
Year	Offset from 1980 (e.g., decimal 10 is 1990)
Months	1-12
Days	1-31
Hours	0-23
Minutes	0-59
Biseconds	0-29
	Each bisecond indicates two seconds.

5.2 Unix Time Format

The Unix date and time format is contained in an unsigned 32-bit field and contains the number of seconds since Greenwich Mean Time (GMT), January 1, 1970.

5.3 Path String Formats

NetWare Macintosh path strings have the following format:

```
"volume::dir:dir:file"
```

NetWare DOS, OS/2, FTAM, and NFS path strings have the following format:
`"volume:dir/dir/file"`

Linux NFS path strings have the following format:
`/dir/dir/file`

5.4 Records

A record consists of three SIDF objects: the data set header, the data set information, and the data set data. A subrecord consists of the data set subheader plus the remaining data from the data set information and/or data set data (data that could not fit into the record). The terms record and subrecord are used only when referring to NWSMSetNewRecordHeader, NWSMUpdateRecordHeader, and NWSMGetRecordHeaderOnly.

5.5 Data Types

The utility library uses the following data types:

```
BUFFER        unsigned char
BUFFERPTR     unsigned char *
CCODE         unsigned long, 4 bytes
LSTRING       length (1 byte) preceded char array
NWBOOLEAN     short, 2 bytes
STRING        NULL terminated unsigned char array
UINT8         unsigned char, 1 byte
INT16         signed short, 2 bytes
UINT16        unsigned short, 2 bytes
UINT32        unsigned long, 4 bytes
```

5.6 Other Documents

Standard ECMA-208 is available free of charge from:

- ECMA,114 Rue du Rhône, CH-1204 Geneva, Switzerland
- Fax:+41 22 849.60.01
- Internet:helpdesk@ecma.ch
- It's also available as the E208-DOC.EXE or E208-PSC.EXE file from ECMANEWS.

5.7 Extensions

An extension contains TSA specific encoded data. TSAs can return relevant information regarding data sets that cannot be represented by the existing structures and their field definitions using extensions.

An extension is returned as a size, buffer pair. The size holds the number of encoded bytes of information in the buffer. All extension functions require the buffer and size parameters to process the extensions.

Each extension is represented by a extension tag and a tag version. the extension tag is used to denote the type of extension that is returned and the version is used to denote the version of the

extension. Extension versions are provided to have the capability to extend existing tags without disrupting existing applications usage of older versions of the same tag.

As extension information for some tags can have dynamic fields, all extension functions allocate the extension pointer and require a close of the extension to free the allocated resources.

Extension structure provides the extension information as a void pointer and hence can be type casted to the appropriate structure for accessing the structure fields.

Utility Library Functions

6

This section lists the Utility Library functions and describes their purpose, syntax, parameters, and return values.

6.1 Date and Time Functions

Date and Time functions pack, unpack, and convert date and time data to DOS, UNIX, ECMA formats for DOS and UNIX file systems. They also provide the capability to check for the valid date and time and retrieve the current date and time.

NWSMCheckDateAndTimeRange

Compares a date and time value against a date and time range.

Syntax

```
#include <smsutapi.h>

CCODE NWSMCheckDateAndTimeRange (
   UINT32   firstDateAndTime,
   UINT32   lastDateAndTime,
   UINT32   compareDateAndTime);
```

Parameters

firstDateAndTime

   (IN) Specifies the starting date and time value in DOS date and time format.

lastDateAndTime

   (IN) Specifies the ending date and time value in DOS date and time format.

compareDateAndTime

   (IN) Specifies the date and time value to be compared against the starting and ending values in DOS date and time format.

Return Values

The following table lists the return values associated with the function.

TRUE	compareDateAndTime is out of bounds.
FALSE	compareDateAndTime is within bounds.

Remarks

NWSMCheckDateAndTimeRange assumes that the input values are correct. Zero specifies no beginning or ending boundary and FALSE will be returned.

See Also

NWSMGetCurrentDateAndTime, NWSMPackDateTime

NWSMDOSTimeToECMA

Converts a DOS format date and time value to ECMA's local date and time format.

Syntax

```
#include <smsutapi.h>

UINT32 NWSMDOSTimeToECMA (
   UINT32     dosDateTime,
   ECMATime  *ecmaTime);
```

Parameters

dosDateTime

   (IN) Specifies the DOS packed date and time value to convert.

ecmaTime

   (OUT) Points to the ECMA equivalent of dosDateTime.

Return Values

NWSMDOSTimeToECMA always returns zero.

NWSMECMATimeCompare

Compares two ECMA time values and indicates if they are the same, less than, or greater than one another.

Syntax

```
#include <smsutapi.h>

int NWSMECMATimeCompare (
   ECMATime  *ECMATime1,
   ECMATime  *ECMATime2);
```

Parameters

ECMATime1

 (IN) Points to the time to compare against ECMATime2.

ECMATime2

 (IN) Points to the time to compare against ECMATime1.

Return Values

The following table lists the return values associated with the function.

1	ECMATime1 is greater than ECMATime2.
0	ECMATime1 equals ECMATime2.
-1	ECMATime1 is less than ECMATime2.
0xF801	ECMA_TIME_ZONE_UNKNOWN (-2047)

Remarks

ECMA_TIME_ZONE_UNKNOWN is returned if either, or both, time zones of ECMATime1 and ECMATime2 are unknown.

NWSMECMAToDOSTime

Converts an ECMA time value to a DOS packed date and time value.

Syntax

```
#include <smsutapi.h>

CCODE NWSMECMAToDOSTime (
   ECMATime  *ecmaTime,
   UINT32    *dosTime);
```

Parameters

ECMATime

   (IN) Points to the ECMA time value to convert.

dosTime

   (OUT) Points to the DOS packed date and time equivalent of ECMATime.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
Nonzero	Cannot convert ECMATime to a valid DOS time value.

Remarks

If ECMATime contains a date value that is before 1980, dosTime returns zero and
NWSMECMAToDOSTime returns a nonzero value.

NWSMECMAToUnixTime

Converts an ECMA date and time value to a Unix time value.

Syntax

```
#include <smsutapi.h>

CCODE NWSMECMAToUnixTime (
   ECMATime  *ecmaTime,
   UINT32    *unixTime,
   INT32     *tzOffset);
```

Parameters

ecmaTime

    (IN) Points to the ECMA date and time value to convert.

unixTime

    (OUT) Points to the Unix time equivalent of ecmaTime.

tzOffset

    (OUT) Points to the offset in minutes from Coordinated Universal Time (CUT).

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
Nonzero	Cannot convert ecmaTime to valid Unix time.

Remarks

If ecmaTime.typeAndTimeZone contains an invalid type, it is set to ECMA_TIME_ZONE_UNKNOWN.

If unixTime contains local time, tzOffset contains the offset from Coordinated Universal Time. Otherwise tzOffset is set to zero.

If ECMATime contains a value before 1970, unixTime returns 0 and NWSMECMAToUnixTime returns a nonzero value.

tzOffset is always zero if unixTime contains a CUT value, and nonzero if unixTime contains a local time value.

See Also

NWSMGetCurrentDateAndTime

Returns the current date and time information into a DOS packed date and time value.

Syntax

```
#include <smsutapi.h>

UINT32 NWSMGetCurrentDateAndTime (
    void);
```

Return Values

If NWSMGetCurrentDateAndTime is successful, it returns a DOS packed date and time value.

NWSMPackDate

Packs separate date values into a DOS packed date value.

Syntax

```
#include <smsutapi.h>

UINT16 NWSMPackDate (
    UINT16   year,
    UINT16   month,
    UINT16   day);
```

Parameters

year

(IN) Specifies the year value, in DOS date and time format, to pack (minimum value is 1980).

month

(IN) Specifies the month value, in DOS date and time format, to pack (1-12).

day

(IN) Specifies the day value, in DOSdate and time format, to pack (1-31).

Return Values

If NWSMPackDate is successful, it returns the packed date.

Remarks

NWSMPackDate returns the year, month, and day as a 16-bit DOS packed value as follows:
`yyyyyyymmmmddddd`

NWSMPackDate assumes that the input values are correct.

See Also

NWSMPackDateTime

Packs the date and time information into a DOS packed date and time value.

Syntax

```
#include <smsutapi.h>

UINT32 NWSMPackDateTime (
   UINT16   year,
   UINT16   month,
   UINT16   day,
   UINT16   hours,
   UINT16   minutes,
   UINT16   seconds);
```

Parameters

year

   (IN) Specifies the year value to pack (minimum value is 1980).

month

   (IN) Specifies the month value to pack (1-12).

day

   (IN) Specifies the day value to pack (1-31).

hours

   (IN) Specifies the hour value to pack (0-23).

minutes

   (IN) Specifies the minute value to pack (0-59).

seconds

   (IN) Specifies the seconds value to pack (0-59).

Return Values

If NWSMPackDateTime is successful, it returns a DOS packed date and time.

Remarks

NWSMPackDateTime assumes that the input values are correct.

Do not divide seconds by two.

See Also

NWSMUnPackDateTime (page 197)

NWSMPackTime

Packs time information into a DOS packed time value.

Syntax

```
#include <smsutapi.h>

UINT16 NWSMPackTime (
   UINT16   hours,
   UINT16   minutes,
   UINT16   seconds);
```

Parameters

hours

(IN) Specifies the hour value to pack (0-23).

minutes

(IN) Specifies the minute value to pack (0-59).

seconds

(IN) Specifies the number of seconds to pack.

Return Values

If NWSMPackTime is successful, it returns the DOS packed time value.

Remarks

NWSMPackTime returns a DOS packed time value in a 16-bit variable as follows:
```
hhhhhmmmmmmbbbbb
```

where
```
h = hours
m = minutes
b= biseconds
```

NWSMPackTime assumes that all input values are correct.

Do not divide seconds by two.

See Also

NWSMUnixTimeToECMA

Converts a Unix local or Coordinated Universal Time (CUT) time value to ECMA date and time format.

Syntax

```
#include <smsutapi.h>

CCODE NWSMUnixTimeToECMA (
   UINT32        unixTime,
   ECMATime     *ECMATime,
   NWBOOLEAN32   local);
```

Parameters

unixTime

   (IN) Specifies the Unix local or CUT time value.

ECMATime

   (OUT) Points to the converted time value in ECMA date and time format.

local

   (IN) Specifies the time format flag:

   TRUE Local Time
   FALSE CUT Time

Return Values

NWSMUnixTimeToECMA always returns zero.

Remarks

NWSMUnixTimeToECMA does not convert local time to CUT, or vice-versa and accounts for day light savings.

If unixTime contains a CUT value, ECMATime.typeAndTimeZone is set to zero.

See Also

NWSMUnpackDate

Unpacks a DOS packed date value into its separate year, month, and day values.

Syntax

```
#include <smsutapi.h>

void NWSMUnPackDate (
    UINT16   date,
    UINT16  *year,
    UINT16  *month,
    UINT16  *day);
```

Parameters

date

> (IN) Specifies he date and time to unpack in DOS date and time format.

year

> (OUT) Points to the unpacked year value.

month

> (OUT) Points to the unpacked month value.

day

> (OUT) Points to the unpacked day value.

Remarks

NWSMUnPackDate assumes that the input value is correct.

See Also

NWSMUnPackDateTime

Unpacks a DOS packed date and time value into separate date and time values.

Syntax

```
#include <smsutapi.h>

void NWSMUnPackDateTime (
   UINT32   dateTime,
   UINT16  *year,
   UINT16  *month,
   UINT16  *day,
   UINT16  *hours,
   UINT16  *minutes,
   UINT16  *seconds);
```

Parameters

dateTime

   (IN) Specifies the date and time value to unpack in DOS date and time format.

year

   (OUT) Points to the unpacked year value.

month

   (OUT) Points to the unpacked month value.

day

   (OUT) Points to the unpacked day value.

hours

   (OUT) Points to the unpacked hours value.

minutes

   (OUT) Points to the unpacked minutes value.

seconds

   (OUT) Points to the unpacked seconds value.

Remarks

NWSMUnPackDateTime assumes that the input value is correct.

Do not divide seconds by two.

See Also

NWSMPackDateTime (page 193)

NWSMUnpackTime

Unpacks a DOS packed time value into its separate hour, minute, and second values.

Syntax

```
#include <smsutapi.h>

void NWSMUnPackTime (
   UINT16   time,
   UINT16  *hours,
   UINT16  *minutes,
   UINT16  *seconds);
```

Parameters

time

   (IN) Specifies the time value to unpack in DOS time format.

hours

   (OUT) Points to the unpacked hour value.

minutes

   (OUT) Points to the unpacked hour value.

seconds

   (OUT) Points to the unpacked seconds value.

Remarks

NWSMUnPackTime assumes that the input value is correct.

Do not divide seconds by two.

See Also

NWSMPackTime (page 194)

6.2 Data Set Name Functions

Data Set Name functions parse and build the NWSM_DATA_SET_NAME_LIST and NWSM_SELECTION_LIST structures. The Data Set Name functions support maximum of 255 names or paths.

Data Set Name functions use the following definitions:
```
#define VALID     0x2AAAAAAAL
#define INVALID   0x15555555L
```

Context information for each data set name list or selection list is kept in a list handle. valid contains two values, VALID or INVALID, which indicates if the structure is still allocated (VALID), or previously released (INVALID).

NWSMCloseName

Prematurely ends the parsing started by calling NWSMGetFirstName , or ends the name insertion process started by calling NWSMPutFirstName or NWSMPutFirstLName .

Syntax

```
#include <smsutapi.h>

CCODE NWSMCloseName (
   SMS_HANDLE SM_HUGE *handle);
```

Parameters

handle

(IN) Points to the name handle set by calling NWSMGetFirstName, NWSMPutFirstName, or NWSMPutFirstLName.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFF	NWSMUT_INVALID_HANDLE

NWSMGetDataSetName

Returns the data set name in the specified name space type.

Syntax

```
#include <smsutapi.h>

CCODE NWSMGetDataSetName (
   void HUGE                    *buffer,
   UINT32                        nameSpaceType,
   NWSM_DATA_SET_NAME HUGE  *name);
```

Parameters

buffer

   (IN) Points to the NWSM_DATA_SET_NAME_LIST or NWSM_SELECTION_LIST structure and accompanying name list.

nameSpaceType

   (IN) Specifies the name space type of name (see "nameSpaceType Values" on page 347).

name

   (OUT) Points to the data set name as it appears in nameSpaceType.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER
0xFFFBFFFB	NWSMUT_OUT_OF_MEMORY
0xFFFBFFFC	NWSMUT_NO_MORE_NAMES

See Also

NWSMGetNextName (page 204)

NWSMGetFirstName

Returns the first data set name contained in NWSM_DATA_SET_NAME_LIST or NWSM_SELECTION_LIST.

Syntax

```
#include <smsutapi.h>

CCODE NWSMGetFirstName (
   void SM_HUGE              *buffer,
   NWSM_DATA_SET_NAME SM_HUGE  *name,
   SMS_HANDLE SM_HUGE          *handle);
```

Parameters

buffer

  (IN) Points to the NWSM_DATA_SET_NAME_LIST or NWSM_SELECTION_LIST structure plus its accompanying name list.

name

  (OUT) Points to the data set name.

handle

  (OUT) Points to the name handle used for all subsequent get name functions.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFC	NWSMUT_NO_MORE_NAMES
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER
0xFFFBFFFB	NWSMUT_OUT_OF_MEMORY

Remarks

NWSMGetFirstName returns the first name contained in buffer. To get the next name in the list, call NWSMGetNextName. To end the retrieval of data set names, call NWSMCloseName.

buffer could contain a list of terminal path names, full path names, or a single TSA defined object.

If NWSMGetFirstName returns NWSMUT_NO_MORE_NAMES, do not call NWSMCloseName.

If the creator name space was requested when buffer was built, name contains the data set name as it appears under that name space.

See Also

NWSMCloseName (page 200), NWSMGetNextName (page 204)

NWSMGetNextName

Continues the data set name parsing process started by calling NWSMGetFirstName .

Syntax

```
#include <smsutapi.h>

CCODE NWSMGetNextName(
    SMS_HANDLE SM_HUGE          *handle,
    NWSM_DATA_SET_NAME SM_HUGE  *name);
```

Parameters

handle

(IN) Points to the name handle returned by NWSMGetFirstName.

name

(OUT) Points to the data set name information.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER
0xFFFBFFFF	NWSMUT_INVALID_HANDLE
0xFFFBFFFC	NWSMUT_NO_MORE_NAMES

Remarks

NWSMGetNextName returns the next name contained in buffer.

To close handle, call NWSMCloseName. handle is automatically closed when NWSMGetNextName returns NWSMUT_NO_MORE_NAMES.

See Also

NWSMGetNextName Example

```
#include <smsutapi.h>

CCODE ccode;
UINT32 HUGE handle
NWSM_DATA_SET_NAME HUGE name;
```

```
NWSM_DATA_SET_NAME_LIST *dataSetList;
void HUGE *buffer;

buffer = dataSetList;
if ((ccode = NWSMGetFirstName(buffer, &name, &handle)) == 0)
{
   while ((ccode = NWSMGetNextName(&handle, &name)) == 0)
   {
      ...
   }
}
if (!ccode)
   NWSMCloseName(&handle);
```

NWSMGetOneName

Returns the first data set name from a buffer.

Syntax

```
#include <smsutapi.h>

CCODE NWSMGetOneName (
   void HUGE                     *buffer,
   NWSM_DATA_SET_NAME HUGE  *name);
```

Parameters

buffer

   (IN) Points to the NWSM_DATA_SET_NAME_LIST or NWSM_SELECTION_LIST
   structure plus its accompanying name list.

name

   (OUT) Points to the data set name.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER
0xFFFBFFFB	NWSMUT_OUT_OF_MEMORY
0xFFFBFFFC	NWSMUT_NO_MORE_NAMES

See Also

NWSMGetFirstName (page 202), NWSMGetNextName (page 204)

NWSMPutFirstLName

Clears the name list and places one data set name into a date set name list or a selection list.

Syntax

```
#include <smsutapi.h>

CCODE NWSMPutFirstLName (
   void SM_HUGE  **buffer,
   UINT32          nameSpaceType,
   UINT32          selectionType,
   NWBOOLEAN       reverseOrder,
   void          *sep1,
   void          *sep2,
   UINT32          nameLength,
   void          *name,
   SMS_HANDLE SM_HUGE  *handle);
```

Parameters

buffer

   (IN/OUT) Points to the NWSM_DATA_SET_NAME_LIST or NWSM_SELECTION_LIST structure to receive the data set name.

nameSpaceType

   (IN) Specifies the name space type of name (see "nameSpaceType Values" on page 347).

selectionType

   (IN) Specifies the selection type for a selection list (see "selectionType Values" on page 348).

reverseOrder

   (IN) Specifies the order of the characters in name, sep1, and sep2, when placed into buffer as returned by NWSMTSGetNameSpaceTypeInfo:

   TRUE Reverse the names
   FALSE Do not reverse the names

sep1

   (IN) Points to the first separator used for paths in the specified name space as returned by NWSMTSGetNameSpaceTypeInfo.

sep2

   (IN) Points to the second separator used for paths in the specified name space as returned by NWSMTSGetNameSpaceTypeInfo.

nameLength

   (IN) Specifies the number of bytes within name to put into the list.

name

> (IN) Points to the name of the data set as it exists in the specified name space.

handle

> (OUT) Points to the name handle to use for subsequent name insertion functions.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER: name is a NULL pointer. Also returned if Turbo C or Microsoft C is used and the increase in buffer size is larger than 32,767 bytes.
0xFFFBFFFB	NWSMUT_OUT_OF_MEMORY
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW: The list of names or paths is greater than 255.

Remarks

NWSMPutFirstLName is similar to NWSMPutFirstName. However, NWSMPutFirstLName requires the length of the data set name.

NWSMPutFirstLName also supports data set names in Unicode format.

If there is not enough space for the name, the buffer is resized. If buffer is set to NULL, memory is allocated to it. You are responsible to free the memory allocated by buffer.

If buffer contains a data set name list, selectionType must be set to zero. If buffer contains a selection list, selectionType must be set to a selection type.

If reverseOrder is TRUE, C:\\SYSTEM\\TEMP.EXE would be EXE.TEMP\\SYSTEM\\:C.

See Also

NWSMPutFirstName

Places the first data set name into a NWSM_DATA_NAME_LIST or NWSM_SELECTION_LIST structure.

Syntax

```
#include <smsutapi.h>

CCODE NWSMPutFirstName (
   void SM_HUGE   **buffer,
   UINT32          nameSpaceType,
   UINT32          selectionType,
   NWBOOLEAN       reverseOrder,
   void           *sep1,
   void           *sep2,
   void           *name,
   SMS_HANDLE SM_HUGE  *handle);
```

Parameters

buffer

(IN/OUT) Points to the NWSM_DATA_SET_NAME_LIST or NWSM_SELECTION_LIST structure to receive the data set name.

nameSpaceType

(IN) Specifies the name space type of name (see "nameSpaceType Values" on page 347).

selectionType

(IN) Specifies the selection type for a selection list (see "selectionType Values" on page 348).

reverseOrder

(IN) Specifies the order of the characters in name, sep1, and sep2, when placed into buffer as returned by NWSMTSGetNameSpaceTypeInfo:

TRUE Reverse the names
FALSE Do not reverse the names

sep1

(IN) Points to the first separator used for paths in the specified name space as returned by NWSMTSGetNameSpaceTypeInfo.

sep2

(IN) Points to the second separator used for paths in the specified name space as returned by NWSMTSGetNameSpaceTypeInfo.

name

(IN) Points to the name of the data set as it exists in the specified name space.

handle

(OUT) Points to the name handle to use for subsequent name insertion functions.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER: name is NULL.
0xFFFBFFFB	NWSMUT_OUT_OF_MEMORY
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW: The list of names or paths is greater than 255.

Remarks

NWSMPutFirstName allocates memory for buffer, clears the existing name list, and inserts one data set name into the name list. All appropriate fields in the buffer are updated to indicate the number of data set names in the name list.

NWSMPutFirstName can build a selection list or data set name list for NWSMTSScanDataSetBegin and NWSMTSScanDataSetNext.

NWSMPutFirstName is similar to NWSMPutFirstLName. However, NWSMPutFirstLName requires the length of the data set name.

NWSMPutFirstName also supports data set names in Unicode format.

If there is not enough space for the name, the buffer is resized. If buffer is set to NULL, memory is allocated to it. You are responsible to free the memory allocated by buffer.

If buffer contains a data set name list, selectionType must be set to zero. If buffer contains a selection list, selectionType must be set to a selection type.

If reverseOrder is TRUE, C:\\SYSTEM\\TEMP.EXE would be EXE.TEMP\\SYSTEM\\:C.

See Also

NWSMCloseName (page 200), NWSMPutFirstLName (page 207), NWSMPutNextLName (page 211), NWSMPutNextName (page 213)

NWSMPutNextLName

Continues the data set name insertion process started by calling NWSMPutFirstLName and places the next data set name into a date set name list or a selection list.

Syntax

```
#include <smsutapi.h>

CCODE NWSMPutNextLName (
   void SM_HUGE         **buffer,
   SMS_HANDLE SM_HUGE   *handle,
   UINT32               nameSpaceType,
   UINT32               selectionType,
   NWBOOLEAN            reverseOrder,
   void                 *sep1,
   void                 *sep2,
   UINT32               nameLength,
   void                 *name);
```

Parameters

buffer

> (IN/OUT) Points to the NWSM_DATA_SET_NAME_LIST or NWSM_SELECTION_LIST structure to receive the data set name.

handle

> (IN) Points to the name handle returned by NWSMPutFirstName or NWSMPutFirstLName.

nameSpaceType

> (IN) Specifies the name space type of name (see "nameSpaceType Values" on page 347).

selectionType

> (IN) Specifies the selection type for a selection list (see "selectionType Values" on page 348).

reverseOrder

> (IN) Specifies the order of the characters in name, sep1, and sep2, when placed into buffer as returned by NWSMTSGetNameSpaceTypeInfo:

> TRUE Reverse the names
> FALSE Do not reverse the names

sep1

> (IN) Points to the first separator used for paths in the specified name space as returned by NWSMTSGetNameSpaceTypeInfo.

sep2

> (IN) Points to the second separator used for paths in the specified name space as returned by NWSMTSGetNameSpaceTypeInfo.

nameLength

(IN) Specifies the number of bytes within name to put into the list.

name

(IN) Points to the name of the data set as it exists in the specified name space.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER
0xFFFBFFFB	NWSMUT_OUT_OF_MEMORY
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW: The list of names or paths is greater than 255.

Remarks

NWSMPutNextLName is similar to NWSMPutNextName. However, NWSMPutNextLName requires the length of the data set name.

NWSMPutNextLName also supports data set names in Unicode format.

If there is not enough space for the name, the buffer is resized. If buffer is set to NULL, memory is allocated to it. You are responsible to free the memory allocated by buffer.

If buffer contains a data set name list, selectionType must be set to zero. If buffer contains a selection list, selectionType must be set to a selection type.

If reverseOrder is TRUE, C:\\SYSTEM\\TEMP.EXE would be EXE.TEMP\\SYSTEM\\:C.

See Also

NWSMCloseName (page 200), NWSMPutFirstName (page 209), NWSMPutNextName (page 213)

NWSMPutNextName

Continues the data set name insertion process started by calling NWSMPutFirstName and places the next data set name into a date set name list or a selection list.

Syntax

```
#include <smsutapi.h>

CCODE NWSMPutNextName (
    void SM_HUGE        **buffer,
    SMS_HANDLE SM_HUGE  *handle,
    UINT32              nameSpaceType,
    UINT32              selectionType,
    NWBOOLEAN           reverseOrder,
    void                *sep1,
    void                *sep2,
    void                *name);
```

Parameters

buffer

（IN/OUT) Points to the NWSM_DATA_SET_NAME_LIST or NWSM_SELECTION_LIST structure to receive the data set name.

handle

(IN) Points to the name handle returned by NWSMPutFirstName or NWSMPutFirstLName.

nameSpaceType

(IN) Specifies the name space type of name (see "nameSpaceType Values" on page 347).

selectionType

(IN) Specifies the selection type for a selection list (see "selectionType Values" on page 348).

reverseOrder

(IN) Specifies the order of the characters in name, sep1, and sep2, when placed into buffer as returned by NWSMTSGetNameSpaceTypeInfo:

TRUE Reverse the names
FALSE Do not reverse the names

sep1

(IN) Points to the first separator used for paths in the specified name space as returned by NWSMTSGetNameSpaceTypeInfo.

sep2

(IN) Points to the second separator used for paths in the specified name space as returned by NWSMTSGetNameSpaceTypeInfo.

name

> (IN) Points to the name of the data set as it exists in the specified name space.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER: handle is invalid or name is NULL.
0xFFFBFFFB	NWSMUT_OUT_OF_MEMORY
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW: The list of names or paths is greater than 255.

Remarks

NWSMPutNextName increments the number of list entries by one.

To end the data set name insertion, call NWSMCloseName.

NWSMPutNextName is similar to NWSMPutNextLName. However, NWSMPutNextLName requires the length of the data set name.

NWSMPutNextName also supports data set names in Unicode format.

If there is not enough space for the name, the buffer is resized. If buffer is set to NULL, memory is allocated to it. You are responsible to free the memory allocated by buffer.

If buffer contains a data set name list, selectionType must be set to zero. If buffer contains a selection list, selectionType must be set to a selection type.

If reverseOrder is TRUE, C:\\SYSTEM\\TEMP.EXE would be EXE.TEMP\\SYSTEM\\:C.

See Also

NWSMPutOneLName

Places one data set name into a data set name list or a selection list.

Syntax

```
#include <smsutapi.h>

CCODE NWSMPutLOneName (
   void HUGE   **buffer,
   UINT32       nameSpaceType,
   UINT32       selectionType,
   NWBOOLEAN    reverseOrder,
   void        *sep1,
   void        *sep2,
   UINT32       nameLength,
   void        *name);
```

Parameters

buffer

(OUT) Points to the NWSM_DATA_SET_NAME_LIST or NWSM_SELECTION_LIST structure to receive the data set name.

nameSpaceType

(IN) Specifies the name space type of name (see "nameSpaceType Values" on page 347).

selectionType

(IN) Specifies the selection type for a selection list (see "selectionType Values" on page 348).

reverseOrder

(IN) Specifies the order of the characters in name, sep1, and sep2, when placed into buffer as returned by NWSMTSGetNameSpaceTypeInfo:

TRUE Reverse the names
FALSE Do not reverse the names

sep1

(IN) Points to the first separator used for paths in the specified name space as returned by NWSMTSGetNameSpaceTypeInfo.

sep2

(IN) Points to the second separator used for paths in the specified name space as returned by NWSMTSGetNameSpaceTypeInfo.

nameLength

(IN) Specifies the number of bytes within name to put into the list.

name

(IN) Points to the name of the data set as it exists in the specified name space.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFB	NWSMUT_OUT_OF_MEMORY
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW: The list of names or paths is greater than 255.

Remarks

NWSMPutOneLName is similar to NWSMPutOneName. However, NWSMPutOneLName requires the length of the data set name.

NWSMPutOneLName also supports data set names in Unicode format.

If there is not enough space for the name, the buffer is resized. If buffer is set to NULL, memory is allocated to it. You are responsible to free the memory allocated by buffer.

If buffer contains a data set name list, selectionType must be set to zero. If buffer contains a selection list, selectionType must be set to a selection type.

If reverseOrder is TRUE, C:\\SYSTEM\\TEMP.EXE would be EXE.TEMP\\SYSTEM\\:C.

See Also

NWSMPutOneName (page 217)

NWSMPutOneLName Example

```
#include <smsutapi.h>

CCODE      ccode;
void HUGE *buffer;
UINT32    nameSpaceType, selectionType, len;
STRING    sep1, sep2, name;
NWBOOLEAN reverseOrder;

len = strlen(name);
ccode = NWSMPutOneLName(&buffer, nameSpaceType, selectionType,
reverseOrder, sep1, sep2, len, name);
```

NWSMPutOneName

Places one data set name into a data set name list or a selection list.

Syntax

```
#include <smsutapi.h>

CCODE NWSMPutOneName (
   void HUGE   **buffer,
   UINT32       nameSpaceType,
   UINT32       selectionType,
   NWBOOLEAN    reverseOrder,
   void        *sep1,
   void        *sep2,
   void        *name);
```

Parameters

buffer

(OUT) Points to the NWSM_DATA_SET_NAME_LIST or NWSM_SELECTION_LIST structure to receive the data set name.

nameSpaceType

(IN) Specifies the name space type of name (see "nameSpaceType Values" on page 347).

selectionType

(IN) Specifies the selection type for a selection list (see "selectionType Values" on page 348).

reverseOrder

(IN) Specifies the order of the characters in name, sep1, and sep2, when placed into buffer as returned by NWSMTSGetNameSpaceTypeInfo:

TRUE Reverse the names
FALSE Do not reverse the names

sep1

(IN) Points to the first separator used for paths in the specified name space as returned by NWSMTSGetNameSpaceTypeInfo.

sep2

(IN) Points to the second separator used for paths in the specified name space as returned by NWSMTSGetNameSpaceTypeInfo.

name

(IN) Points to the name of the data set as it exists in the specified name space.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFB	NWSMUT_OUT_OF_MEMORY
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER
0xFFFBFFFF	NWSMUT_INVALID_HANDLE
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW: The list of names or paths is greater than 255.

Remarks

NWSMPutOneName destroys the existing name list in buffer, puts one data set name into buffer, and sets the name list count to one.

NWSMPutOneName is similar to NWSMPutOneLName. However, NWSMPutOneLName requires the length of the data set name.

NWSMPutOneName also supports data set names in Unicode format.

NWSMPutOneName can build the resourceName parameter for NWSMTSScanDataSetBegin.

If there is not enough space for the name, the buffer is resized. If buffer is set to NULL, memory is allocated to it. You are responsible to free the memory allocated by buffer.

If buffer contains a data set name list, selectionType must be set to zero. If buffer contains a selection list, selectionType must be set to a selection type.

If reverseOrder is TRUE, C:\\SYSTEM\\TEMP.EXE would be EXE.TEMP\\SYSTEM\\:C.

Set nameSpaceType to NWSM_TSA_DEFINED_RESOURCE if name was returned by NWSMTSListTSResources.

6.3 Extension Functions

Extension information buffers are encoded in a byte array, the following extension functions parse the encoding and return the information as an extension structure.

These functions build the NWSM_EXTENSION_INFORMATION structure. The context information regarding the extensions are kept in the list handle.

NWSMCloseExtension

Prematurely ends the parsing started by NWSMGetFirstExtension. Also, used to free resources allocated when processing extension information using NWSMGetExtension.

Syntax

```
#include <smsutapi.h>

CCODE NWSMCloseExtension(
  SMS_HANDLE SM_HUGE *handle);
```

Parameters

handle

(IN) Points to the extension handle set by calling NWSMGetExtension or NWSMGetFirstExtension.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFF	NWSMUT_INVALID_HANDLE

NWSMGetExtension

Returns the extension as specified by the extension tag.

Syntax

```
#include <smsutapi.h>

CCODE NWSMGetExtension (
   void SM_HUGE                      *buffer,
   UINT32                            size,
   UINT32                            extensionTag,
   NWSM_EXTENSION_INFORMATION SM_HUGE  **extension,
   SMS_HANDLE                        *handle);
```

Parameters

buffer

   (IN) Points to the start of the extension buffer.

size

   (IN) Size of the extension buffer.

extensionTag

   (IN) Specifies the required extension information tag.

extension

   (OUT) Points to the required extension information.

handle

   (OUT) Points to the extension handle, used to maintain context information.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER
0xFFFBFFFB	NWSMUT_OUT_OF_MEMORY
0xFFFBFFF8	NWSMUT_NO_MORE_EXTENSIONS

Remarks

Extension information field info can contain pointer type fields and hence internal allocation are made to return such elements. NWSMCloseExtension should be called to free the resources that are allocated for such purpose.

On errors NWSMCloseExtension need not be called.

buffer should point to the start of the extension buffer. The size parameter should be set to the byte count of information present in the buffer parameter.

Application should copy the extension information as the pointer is reused by the call to get the next extension. To free the extension buffer call NWSMCloseExtension.

See Also

NWSMGetFirstExtension, NWSMCloseExtension

NWSMGetFirstExtension

Returns the first extension contained in buffer.

Syntax

```
#include <smsutapi.h>
 CCODE NWSMGetFirstExtension (
  void SM_HUGE                         *buffer,
  UINT32                               *size,
  NWSM_EXTENSION_INFORMATION SM_HUGE **extension,
  SMS_HANDLE SM_HUGE                   *handle);
```

Parameters

buffer

   (IN) Points to the start of the extension buffer.

size

   (IN) Size of the extension buffer.

extension

   (OUT) Points to the required extension information.

handle

   (OUT) Points to the extension handle, used to maintain context information.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER
0xFFFBFFFB	NWSMUT_OUT_OF_MEMORY
0xFFFBFFF8	NWSMUT_NO_MORE_EXTENSIONS

Remarks

NWSMGetFirstExtension returns the first extension encountered in buffer. To get the next extension in buffer, call NWSMGetNextExtension. To end the retrieval of extensions call NWSMCloseExtension.

buffer should point to the start of the extension buffer. The size parameter should be set to the byte count of information present in the buffer parameter.

On errors NWSMCloseExtension need not be called.

See Also

NWSMGetNextExtension, NWSMCloseExtension

NWSMGetNextExtension

Continues the extension parsing process started by NWSMGetFirstExtension.

Syntax

```
#include <smsutapi.h>

CCODE NWSMGetNextExtension(
  SMS_HANDLE SM_HUGE                  *handle,
  NWSM_EXTENSION_INFORMATION SM_HUGE **extension);
```

Parameters

handle

(IN) Points to the extension handle as returned by NWSMGetFirstExtension.

extension

(OUT) Points to the extension information.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER
0xFFFBFFFB	NWSMUT_OUT_OF_MEMORY
0xFFFBFFF8	NWSMUT_NO_MORE_EXTENSIONS

Remarks

NWSMGetNextExtension returns the next extension contained in buffer passed in to NWSMGetFirstExtension.

On errors NWSMCloseExtension need not be called.

Using extension handles as obtained from NWSMGetExtension can lead to unpredictable results.

Application should copy the extension information as the pointer is reused by the call to get the next extension. To free the extension buffer call NWSMCloseExtension.

See Also

NWSMGetFirstExtension

6.4 List Functions

List functions manage a linked list(s). Each element of the list has a text descriptor field and another information field in which any data can be stored.

-
-
-
-

NWSMAppendToList

Appends an element to NWSM_LIST list.

Syntax

```
#include <smsutapi.h>

NWSM_LIST *NWSMAppendToList (
   NWSM_LIST_PTR  *listPtr,
   BUFFERPTR        text,
   void            *otherInfo);
```

Parameters

listPtr

   (IN) Points to the list to append (cannot be a NULL pointer).

text

   (IN) Points to the text information to be added to a new element.

otherInfo

   (IN) Points to the buffer containing information associated with text (optional).

Return Values

The following table lists the return values associated with the function.

Nonzero	Successful, returns a pointer to the appended element.
NULL	Out of memory

Remarks

NWSMAppendToList creates an element, copies text to the element, attaches otherInfo to the element, and appends the element to the end of the list.

NWSMAppendToList assumes that list is valid.

listPtr was initialized by NWSMInitList.

See Also

NWSMDestroyList (page 227), NWSMInitList (page 229)

NWSMDestroyList

Releases the memory associated with NWSM_LIST.

Syntax

```
#include <smsutapi.h>

void NWSMDestroyList (
   NWSM_LIST_PTR  *list);
```

Parameters

list

   (IN) Points to a list initialized by NWSMInitList (cannot be NULL).

Remarks

Everything in the list, except the list head, is freed when NWSMDestroyList is called.

If the engine did not specify a free memory routine by calling NWSMInitList, NWSMDestroyList will free the elements, but not the engine defined structures.

NWSMGetListHead

Returns the first element in a list.

Syntax

```
#include <smsutapi.h>

NWSM_LIST *NWSMGetListHead (
    NWSM_LIST_PTR  *listPtr);
```

Parameters

listPtr

(IN) Points to a list initialized by NWSMInitList (cannot be a NULL pointer).

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
Nonzero	Points to the first element in the list.
NULL	The list is empty.

NWSMInitList

Initializes a list head for an NWSM_LIST list, and sets the functions to use in the list.

Syntax

```
#include <smsutapi.h>

void NWSMInitList (
   NWSM_LIST_PTR  *listPtr,
   void           (*freeRoutine)
    (void *memoryPointer));
```

Parameters

listPtr

  (IN/OUT) Points to an NWSM_LIST_PTR structure.

freeRoutine

  (IN) Points to the routine to free the memory associated with each element's otherInfo field.

Remarks

NWSMDestroyList uses NWSMInitList to help destroy the list.

NWSMInitList initializes a list head structure as follows:

```
head = tail = NULL
sortProc = stricmp
freeProcedure = freeRoutine
```

You can set sortProc to another compare function.

To get the current list head, call NWSMGetListHead. To build the list, call NWSMAppendToList.

If the list element's otherInfo field contains a non-allocated value, freeRoutine must be set to NULL. However, this routine will not free the memory allocated to otherInfo.

See Also

NWSMAppendToList (page 226), NWSMDestroyList (page 227)

6.5 Path Functions

Path functions remove a child node, return a pointer to the child node, qualify a path, verify a NetWare DOS name, concatenate strings, etc.

NWSMAllocGenericString

Allocates or reallocates memory for a STRING_BUFFER structure.

Syntax

```
#include <smsutapi.h>

STRING_BUFFER *NWSMAllocGenericString(
UINT32       nameSpaceType,
STRING_BUFFER **string,
INT32    size);
```

Parameters

nameSpaceType

Specifies the name space type of the source and destination strings (see "nameSpaceType Values" on page 347).

string

(IN/OUT) Points to the new or reallocated buffer.

size

(IN) Specifies the size of the new or reallocated structure.

Return Values

The following table lists the return values associated with the function.

NULL	Cannot reallocate memory.
Nonzero	Reallocation is successful and the pointer points to string.

Remarks

NWSMAllocGenericString is an enhanced version of NWSMAllocString and it supports strings in UTF-8 format.

You must free the memory allocated to buffer by calling NWSMFreeString.

If string is NULL, a new structure is allocated. Otherwise NWSMAllocGenericString assumes that it is a valid pointer and reallocates more memory for string.

If string is not NULL, and size is less than or equal to zero, a standard increment size is added to the structure's existing size or string is reallocated to the number of bytes specified by size.

The standard size is currently 256 bytes.

See Also

NWSMFreeString (page 246)

NWSMAllocString

Allocates or reallocates memory for a STRING_BUFFER structure.

Syntax

```
#include <smsutapi.h>
STRING_BUFFER *NWSMAllocString (
STRING_BUFFER **string,    INT16    size);
```

Parameters

string

(IN/OUT) Points to the new or reallocated buffer.

size

(IN) Points to the size of the new or reallocated structure.

Return Values

The following table lists the return values associated with the function.

NULL	Cannot reallocate memory.
Nonzero	Reallocation is successful and the pointer points to string.

Remarks

NOTE: This function is supported only for backward compatibility, the NWSMAllocGenericString function can be used instead of this, as it provides the same functionality.

You must free the memory allocated to buffer by calling NWSMFreeString.

If string is NULL, a new structure is allocated. Otherwise NWSMAllocString assumes that it is a valid pointer and reallocates more memory for string.

If string is not NULL, and size is less than or equal to zero, a standard increment size is added to the structure's existing size or string is reallocated to the number of bytes specified by size.

The standard size is currently 256 bytes.

See Also

NWSMFreeString (page 246)

NWSMCatGenericString

Appends a specified number of bytes from the source string to the destination string.

Syntax

```
#include <smsutapi.h>

void* NWSMCatGenericString (
   UINT32              nameSpacetype,
   STRING_BUFFER **dest,
   void              *source);
```

Parameters

name SpaceType

(IN) Specifies the name space type of the source and destination strings (see "nameSpaceType Values" on page 347).

dest

(IN/OUT) Points to the string to append.

source

(IN) Points to the string to append to the end of dest.

Return Values

The following table lists the return values associated with the function.

NULL	Cannot concatenate the strings.
Nonzero	The strings are concatenated and the pointer points to dest.string.

Remarks

NWSMCatGenericString supports strings in UTF-8 format and mult-byte formatting.

To free the string, call NWSMFreeString.

If dest is NULL, NWSMCatGenericString allocates memory for the string. If dest.string is too small to contain the concatenated strings, NWSMCatGenericString frees the memory and allocates a new structure with more memory.

See Also

NWSMCatStrings (page 239), NWSMFreeString (page 246), NWSMStr (page 255)

NWSMCatGenericString Example

```
#include <smsutapi.h>

STRING resultString;
STRING_BUFFER *dest = NULL;
char *source = "*.exe", *destString = "/home/user/dirl";
INT16 srcLen = -1;
UINT16 destLen;

destLen = strlen(destString);
dest = (STRING_BUFFER *)malloc(sizeof(STRING_BUFFER) + destLen);
dest->size = destLen + 1;
strcpy(dest->string, destString);

resultString = NWSMCatString(&dest, source, srcLen);
```

NWSMCatGenericStrings

Concatenates a specified number of strings into a STRING_BUFFER structure.

Syntax

```
#include <smsutapi.h>

void* NWSMCatGenericStrings (
   UINT32          nameSpaceType,
   UINT8           numStrings,
   STRING_BUFFER **dest,
   void           *src1,
   void           *src2,...);
```

Parameters

nameSpaceType

(IN) Specifies the name space type of the source and destination strings (see "nameSpaceType Values" on page 347).

numStrings

(IN) Specifies the number of source strings to concatenate.

dest

(OUT) Points to the buffer to receive the concatenated strings.

src1

(IN) Points to the first string to concatenate.

src2

(IN) Points to the second string to concatenate.

...

(IN) Points to other strings to concatenate.

Return Values

The following table lists the return values associated with the function.

NULL	Cannot concatenate the strings.
Nonzero	The strings are concatenated and the pointer points to dest.string.

Remarks

NWSMCatGenericStrings supports strings in UTF-8 format and mult-byte formatting.

To free the string, call NWSMFreeString.

If dest is NULL, NWSMCatGenericStrings allocates memory for the string. If dest.string is too small to contain the concatenated strings, NWSMCatGenericStrings frees the memory and allocates a new structure with more memory.

See Also

NWSMCatGenericStrings Example

```
#include <smsutapi.h>

STRING      resultString;
UINT8       numStrings = 2;
STRING_BUFFER *dest = NULL;
char *destString = "/home/user/dir1", *src1 = "dir1", *src2 = "*.c";
UINT16 destLen;

destLen = strlen(destString);
dest = (STRING_BUFFER *)malloc(sizeof(STRING_BUFFER) + destLen);
dest->size = destLen + 1;
strcpy(dest->string, destString);

resultString = NWSMCatStrings(numStrings, &dest, src1, src2);
```

NWSMCatString

Appends a specified number of characters to a string.

Syntax

```
#include <smsutapi.h>

STRING NWSMCatString (
   STRING_BUFFER **dest,
   void           *source,
   INT16           srcLen);
```

Parameters

dest

(IN/OUT) Points to the string to append.

source

(IN) Points to the string to append to the end of dest.

srcLen

(IN) Specifies the length of source.

Return Values

The following table lists the return values associated with the function.

NULL	Cannot concatenate the strings.
Nonzero	The strings are concatenated and the pointer points to dest.string.

Remarks

NOTE: This function is supported only for backward compatibility, the NWSMCatGenericString function can be used instead of this, as it provides the same functionality.

To free the string, call NWSMFreeString.

If dest is NULL, NWSMCatString allocates memory for the string. If dest.string is too small to contain the concatenated strings, NWSMCatString frees the memory and allocates a new structure with more memory.

If srcLen is set to -1, NWSMCatString calculates the length of source.

See Also

NWSMCatStrings (page 239), NWSMFreeString (page 246), NWSMStr (page 255)

NWSMCatString Example

```
#include <smsutapi.h>

STRING     resultString;
STRING_BUFFER *dest = NULL;
char *source = "*.exe", *destString = "/home/user/dir1";
INT16 srcLen = -1;
UINT16 destLen;

destLen = strlen(destString);
dest = (STRING_BUFFER *)malloc(sizeof(STRING_BUFFER) + destLen);
dest->size = destLen + 1;
strcpy(dest->string, destString);

resultString = NWSMCatString(&dest, source, srcLen);
```

NWSMCatStrings

Concatenates a specified number of strings into a STRING_BUFFER structure.

Syntax

```
#include <smsutapi.h>

STRING NWSMCatStrings (
   UINT8            numStrings,
   STRING_BUFFER **dest,
   void            *src1,
   void            *src2, ...);
```

Parameters

numStrings

   (IN) Specifies the number of source strings to concatenate.

dest

   (OUT) Points to the buffer to receive the concatenated strings.

src1

   (IN) Points to the first string to concatenate.

src2

   (IN) Points to the second string to concatenate.

...

   (IN) Points to other strings to concatenate.

Return Values

The following table lists the return values associated with the function.

NULL	Cannot concatenate the strings.
Nonzero	The strings are concatenated and the pointer points to dest.string.

Remarks

NOTE: This function is supported only for backward compatibility, the NWSMCatGenericStrings function can be used instead of this, as it provides the same functionality.

To free the string, call NWSMFreeString.

If dest is NULL, NWSMCatStrings allocates memory for the string. If dest.string is too small to contain the concatenated strings, NWSMCatStrings frees the memory and allocates a new structure with more memory.

See Also

NWSMCatStrings Example

```
#include <smsutapi.h>

STRING resultString;
UINT8  numStrings = 2;
STRING_BUFFER *dest = NULL;
char  *destString = "/home/user/dir1", *src1 = "dir1", *src2 = "*.c";
UINT16 destLen;

destLen = strlen(destString);
dest = (STRING_BUFFER *)malloc(sizeof(STRING_BUFFER) + destLen);
dest->size = destLen + 1;
strcpy(dest->string, destString);

resultString = NWSMCatStrings(numStrings, &dest, src1, src2);
```

NWSMCopyGenericString

Copies the source string to the destination string.

Syntax

```
#include <smsutapi.h>

void* NWSMCopyGenericString (
   UINT32              nameSpaceType,
   STRING_BUFFER **dest,
   void              *src);
```

Parameters

nameSpaceType

(IN) Specifies the name space type of the source and destination strings (see "nameSpaceType Values" on page 347).

dest

(OUT) Points to the buffer to receive the copied string.

src

(IN) Points to the string to copy.

Return Values

The following table lists the return values associated with the function.

NULL	Cannot copy the strings.
Nonzero	The strings are copied and the pointer points to dest.string.

Remarks

NWSMCopyGenericString supports strings in UTF-8 format and mult-byte formatting.

To free the string, call NWSMFreeString.

If dest is NULL, NWSMCopyGenericString allocates memory for the string. If dest.string is too small to contain the concatenated strings, NWSMCopyGenericString frees the memory and allocates a new structure with more memory.

See Also

NWSMFreeString (page 246)

NWSMCopyGenericString Example

```
#include <smsutapi.h>

STRING resultString;
STRING_BUFFER *dest = NULL;
char *src = "string";
INT16 srcLen = sizeof(src);

resultString = NWSMCopyString(&dest, src, srcLen);
```

NWSMCopyString

Copies a specified number of characters to a buffer.

Syntax

```
#include <smsutapi.h>

STRING NWSMCopyString (
   STRING_BUFFER **dest,
   void             *src,
   INT16             srcLen);
```

Parameters

dest

   (OUT) Points to the buffer to receive the copied string.

src

   (IN) Points to the string to copy.

srcLen

   (IN) Specifies the length of src.

Return Values

The following table lists the return values associated with the function.

NULL	Cannot copy the strings.
Nonzero	The strings are copied and the pointer points to dest.string.

Remarks

NOTE: This function is supported only for backward compatibility, the NWSMCopyGenericString function can be used instead of this, as it provides the same functionality.

To free the string, call NWSMFreeString.

If dest is NULL, NWSMCopyString allocates memory for the string. If dest.string is too small to contain the concatenated strings, NWSMCopyString frees the memory and allocates a new structure with more memory.

If srcLen is set to -1, NWSMCopyString calculates the length of src.

See Also

NWSMFreeString (page 246)

NWSMCopyString Example

```
#include <smsutapi.h>

STRING resultString;
STRING_BUFFER *dest = NULL;
char *src = "string";
INT16 srcLen = sizeof(src);

resultString = NWSMCopyString(&dest, src, srcLen);
```

NWSMFreeGenericString

Releases the memory held by a STRING_BUFFER structure.

Syntax

```
#include <smsutapi.h>

void NWSMFreeGenericString (
   STRING_BUFFER **string);
```

Parameters

string

   (IN) Points to the STRING_BUFFER structure to free.

Remarks

NWSMFreeGenericString frees the memory allocated to string and sets it to NULL.

If string is NULL, NWSMFreeGenericString does nothing.

NWSMFreeString

Releases the memory held by a STRING_BUFFER structure.

Syntax

```
#include <smsutapi.h>

void NWSMFreeString (
   STRING_BUFFER **string);
```

Parameters

string

> (IN) Points to the STRING_BUFFER structure to free.

Remarks

NOTE: This function is supported only for backward compatibility, the NWSMFreeGenericString function can be used instead of this, as it provides the same functionality.

NWSMFreeString frees the memory allocated to string and sets it to NULL.

If string is NULL, NWSMFreeString does nothing.

NWSMGenericIsWild

Indicates if a path contains wildcard characters.

Syntax

```
#include <smsutapi.h>

NWBOOLEAN NWSMGenericIsWild (
   UINT32   nameSpaceType,
   void    *string);
```

Parameters

nameSpaceType

> (IN) Specifies the name space type of the string (see "nameSpaceType Values" on page 347).

string

> (IN) Points to the path to search for wildcard characters (see "Wildcard Values" on page 349).

Return Values

The following table lists the return values associated with the function.

TRUE	string contains wildcards.
FALSE	string does not contain wildcards.

Remarks

NWSMGenericIsWild supports strings in UTF-8 format.

The path node can be a parent or a child. Only the terminal path node can contain wildcards.

NWSMGenericIsWild Example

```
#include <smsutapi.h>

NWBOOLEAN isWild;
char stringBuf[15]; /* Arbitrary size */
STRING string = (STRING)stringBuf;

UINT32 nameSpace=DOSNameSpace;
isWild = NWSMGenericIsWild(nameSpace,string);
```

NWSMGenericStr

Concatenates the specified number of strings together.

Syntax

```
#include <smsutapi.h>

void* NWSMGenericStr (
   UINT32   nameSpaceType,
   UINT8    n,
   void     *dest,
   void     *src1,
   void     *src2, ...);
```

Parameters

nameSpaceType

> (IN) Specifies the name space type of the source and destination strings (see "nameSpaceType Values" on page 347).

n

> (IN) Specifies the number of source strings to concatenate.

dest

> (OUT) Points to the concatenated strings (cannot be a NULL pointer).

src1

> (IN) Points to the first string to concatenate.

src2

> (IN) Points to the second string to concatenate.

...

> (IN) Points to other strings to concatenate.

Return Values

The following table lists the return values associated with the function.

NULL	Failure
Nonzero	The pointer points to dest.

Remarks

NWSMGenericStr supports strings in UTF-8 format.

dest must be large enough to hold all the strings.

See Also

NWSMCatStrings (page 239), NWSMFreeString (page 246)

NWSMGenericWildMatch

Indicates if a string matches a search pattern.

Syntax

```
#include <smsutapi.h>

NWBOOLEAN NWSMGenericWildMatch (
   UINT32   nameSpaceType,
   void    *pattern,
   void    *string);
```

Parameters

nameSpaceType

(IN) Specifies the name space type of the string and pattern (see "nameSpaceType Values" on page 347).

pattern

(IN) Points to the pattern to search for (see "Wildcard Values" on page 349).

string

(IN) Points to the string to analyze.

Return Values

The following table lists the return values associated with the function.

TRUE	string meets the pattern specifications.
FALSE	string does not match the pattern.

Remarks

NWSMGenericWildMatch supports strings in UTF-8 format.

pattern can contain the following wild cards: ASTERICKS, QUESTION, SQUESTION, or SPERIOD.

If you need double-byte enabling, call NWSMMatchName.

See Also

NWSMIsWild (page 252)

NWSMGenericWildMatch Example

```
#include <smsutapi.h>
```

```
CCODE isWildMatch;
char patternBuf[15], stringBuf[15]; /* Arbitrary size. */
STRING pattern = (STRING)patternBuf, string = stringBuf;

UINT32 nameSpace=DOSNameSpace;
isWildMatch = NWSMGenericWildMatch(nameSpace,pattern,string);
```

NWSMIsWild

Indicates if a path contains a wildcard.

Syntax

```
#include <smsutapi.h>

NWBOOLEAN NWSMIsWild (
   STRING   string);
```

Parameters

string

 (IN) Specifies the path to search for wildcards (see "Wildcard Values" on page 349).

Return Values

The following table lists the return values associated with the function.

TRUE	string contains a wild card.
FALSE	string does not contain a wild card.

Remarks

NOTE: This function is supported only for backward compatibility, the NWSMGenericIsWild function can be used instead of this, as it provides the same functionality.

Only the terminal path node can have wild cards. The path node can be a parent or a child.

NWSMIsWild Example

```
#include <smsutapi.h>

NWBOOLEAN isWild;
char stringBuf[15]; /* Arbitrary size */
STRING string = (STRING)stringBuf;

isWild = NWSMIsWild(string);
```

NWSMMatchName

Performs a case-sensitive search for long spaces and case-insensitive search for all the other namespaces for a pattern in a string.

Syntax

```
#include <smsutapi.h>

int NWSMMatchName (
   UINT32      nameSpaceType,
   void        *pattern,
   void        *string,
   NWBOOLEAN   returnMatchToPatternEndIfWild);
```

Parameters

nameSpaceType

   (IN) Specifies the name space type of string and pattern. (see "nameSpaceType Values" on page 347).

pattern

   (IN) Points to the pattern to search for (see "Wildcard Values" on page 349).

string

   (IN) Points to the string to search.

returnMatchToPatternEndIfWild

   (IN) Specifies whether string is a directory path:

   TRUE Directory path
   FALSE Not a directory path

Return Values

The following table lists the return values associated with the function.

-1	NWSM_MATCH_UNSUCCESSFULL
0	NWSM_MATCH_SUCCESSFULL
1	NWSM_MATCH_TO_STRING_END
2	NWSM_MATCH_TO_PATTERN_END

Remarks

NWSMMatchName also supports strings in UTF-8 format.

Wild card matching is allowed if the name space supports it.

NWSM_MATCH_TO_STRING_END is returned if string matches the first part of pattern.

NWSM_MATCH_TO_PATTERN_END is returned when part of string matches pattern exactly. For example, when pattern is set to "*ted" and string is set to "sorted_i."

returnMatchToPatternEndIfWild allows NWSMMatchName to return the correct return value for directory paths that matches a pattern to its end.

NWSMMatchName Example

```
UINT32 nameSpaceType = DOSNameSpace;
char patternBuf[15], stringBuf[15]; /* Arbitrary size.*/

STRING pattern = patternBuf, string = stringBuf;
strcpy(pattern, "?es*.*");
strcpy(string, "test.exe");

NWSMMatchName(nameSpaceType, pattern, string, TRUE);
```

NWSMStr

Concatenates a specified number of strings together.

Syntax

```
#include <smsutapi.h>

STRING NWSMStr (
   UINT8   n,
   void   *dest,
   void   *src1,
   void   *src2, ...);
```

Parameters

n

(IN) Specifies the number of source strings to concatenate with dest.

dest

(OUT) Points to the concatenated strings (cannot be a NULL pointer).

src1

(IN) Points to the first string to concatenate.

src2

(IN) Points to the second string to concatenate.

...

(IN) Points to other strings to concatenate.

Return Values

The following table lists the return values associated with the function.

NULL	Failure
Nonzero	The pointer points to dest.

Remarks

NOTE: This function is supported only for backward compatibility, the NWSMGenericStr function can be used instead of this, as it provides the same functionality.

dest must be large enough to hold all the strings.

See Also

NWSMCatStrings (page 239), NWSMFreeString (page 246)

NWSMWildMatch

Indicates if a string matches a search pattern.

Syntax

```
#include <smsutapi.h>

NWBOOLEAN NWSMWildMatch (
   STRING   pattern,
   STRING   string);
```

Parameters

pattern

   (IN) Specifies the pattern to search for (see "Wildcard Values" on page 349).

string

   (IN) Specifies the string to search.

Return Values

The following table lists the return values associated with the function.

TRUE	string meets the pattern's specifications.
FALSE	string does not match the pattern.

Remarks

NOTE: This function is supported only for backward compatibility, the NWSMGenericWildMatch function can be used instead of this, as it provides the same functionality.

pattern can contain the following wild cards: ASTERICKS, QUESTION, SQUESTION, or SPERIOD.

If you need double-byte enabling, call NWSMMatchName.

See Also

NWSMIsWild (page 252)

NWSMWildMatch Example

```
#include <smsutapi.h>

CCODE isWildMatch;
char patternBuf[15], stringBuf[15]; /* Arbitrary size. */
```

```
STRING pattern = (STRING)patternBuf, string = stringBuf;

isWildMatch = NWSMWildMatch(pattern,string);
```

6.6 Miscellaneous Functions

Miscellaneous functions generate a CRC and list the SMDRs.

NWSMFreeNameList

Frees the memory held by an NWSM_NAME_LIST object.

Syntax

```
#include <smsutapi.h>

CCODE NWSMFreeNameList (
   NWSM_NAME_LIST **nameList);
```

Parameters

nameList

(IN) Points to the name list to free on input. Points to NULL on output.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION: nameList is NULL.
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER

NWSMGenerateCRC

Generates a CRC value for the given data or continues the CRC calculation from previous data to the current data.

Syntax

```
#include <smsutapi.h>

UINT32 NWSMGenerateCRC (
   UINT32     size,
   UINT32     crc,
   BUFFERPTR  ptr);
```

Parameters

size

   (IN) Specifies the number of bytes in the buffer pointed to by ptr.

crc

   (IN) Specifies the initial CRC value (-1) or a previously accumulated CRC.

ptr

   (IN) Points to the data used in calculating the CRC.

Return Values

If NWSMGenerateCRC is successful, it returns the new CRC value.

Remarks

NWSMGenerateCRC returns a 32-bit CRC. A CRC value needs to be calculated for the whole data set. However, the whole data set cannot be brought into memory; so the data set is broken into sections and CRC values are generated for each section. CRC values generated for the previous section are passed to NWSMGenerateCRC when calculating the current section's CRC.

Example

```
#include <smsutapi.h>

UINT32 genCRC, size, crc = -1;
BUFFERPTR ptr;

/* Set ptr to point to a buffer and set size to the number of data
bytes in the buffer */

genCRC = NWSMGenerateCRC(size, crc, ptr);
```

6.7 SIDF Functions

SIDF functions simplify the building and parsing of SIDF sections.

For more information about SIDF sections and fields, see Standard ECMA-208 (http://www.ecma-international.org/publications/standards/Ecma-208.htm)

NWSMGetDataSetInfo

Returns the data set information.

Syntax

```
#include <smsutapi.h>

CCODE NWSMGetDataSetInfo (
   BUFFERPTR  *buffer,
   UINT32     *bufferSize,
   NWSM_RECORD_HEADER_INFO
              *recordHeaderInfo);
```

Parameters

buffer

(IN/OUT) Points to the beginning of the data set information section of buffer on input. Points to the section following the data set information on ouput.

bufferSize

(IN/OUT) Points to the amount of data left to read in buffer on input. Points to the amount of data in buffer minus the amount of data just read from buffer on output.

recordHeaderInfo

(OUT) Points to the data set information and scan information.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFB	NWSMUT_OUT_OF_MEMORY
0xFFFBFFF0	NWSMUT_BUFFER_UNDERFLOW: A field cannot fit into buffer.

Remarks

Call NWSMGetRecordHeaderOnly before calling NWSMGetDataSetInfo.

NWSMGetDataSetInfo extracts the data set and scan information from a buffer and places it into recordHeaderInfo.

If the data set information spans buffer, NWSMGetDataSetInfo saves the information to an internal buffer and sets recordHeaderInfo.dataSetInfoRetrieved to DATA_SET_INFO_SPANNED. When the function returns, get the next buffer, call NWSMGetRecordHeaderOnly to get the Data Set Subheader, and call NWSMGetDataSetInfo to get the rest of the data. Continue this process until recordHeaderInfo.dataSetInfoRetrieved is DATA_SET_INFO_COMPLETE.

recordHeaderInfo.recordSize is decremented to reflect the amount of data transferred from the buffer.

If the data set information spans the buffer, buffer points to the end of the buffer on return.

recordHeaderInfo.dataSetName and recordHeaderInfo.scanInformation will be set to NULL or to a valid structure. The memory used by the data set name or scan information will be resized if there is not enough space.

recordHeaderInfo.recordSize is the size of the record minus the amount of data put into scanInformation and dataSetName.

recordHeaderInfo.headerSize returns the size of the data set header or data set subheader.

The following recordHeaderInfo fields are not used by NWSMGetDataSetInfo:

```
archiveDateAndTime
addressOfRecordSize
addressForCRC
crcBegin
crcLength
```

See Also

NWSMGetMediaHeaderInfo

Separates an SIDF media header into its various components.

Syntax

```
#include <smsutapi.h>

CCODE NWSMGetMediaHeaderInfo (
   BUFFERPTR          headerBuffer,
   UINT32             headerBufferSize,
   NWSM_MEDIA_INFO  *mediaInfo);
```

Parameters

headerBuffer

   (IN) Points to the beginning of the buffer containing an SIDF media header.

headerBufferSize

   (IN) Specifies the size of the media header in bytes.

mediaInfo

   (OUT) Points to the unformatted media header information.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFF0	NWSMUT_BUFFER_UNDERFLOW
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW
0xFFFBFFF9	NWSMUT_INVALID_FIELD_ID
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER

NWSMGetMediaHeaderInfo Example

```
/* example using SMS DI */
#define MY_BUFFER_SIZE 1024 /* arbitrary size */

NWSM_MEDIA_INFO mediaInfo;
NWSMSD_CONTROL_BLOCK controlBlock; /* SMS DI structure */
NWSMSD_HEADER_BUFFER *headerBuffer;
NWSMSD_MEDIA_HANDLE mediaHandle;
NWBOOLEAN32 verifyHeader;
NWSMSD_HEADER_BUFFER *mediaHeader;
CCODE ccode, completionStatus;

/* Allocate memory for headerBuffer. maxTransferBufferSize is set by
```

```
NWSMSDSessionOpenForReading */

headerBuffer=(NWSMSD_HEADER_BUFFER *)malloc(sizeof(NWSMSD_HEADER_BUFFE
R) + maxTransferBufferSize - 1);
headerBuffer->bufferSize = maxTransferBufferSize;

/* Set up the media header buffer information */
mediaHeader = (NWSMSD_HEADER_BUFFER *)
malloc(sizeof(NWSMSD_HEADER_BUFFER) +MY_BUFFER_SIZE - 1);
mediaHeader->bufferSize = MY_BUFFER_SIZE;

verifyHeader = TRUE;
ccode = NWSMSDMediaHeaderReturn(sdiConnection, mediaHandle,
verifyHeader, mediaHeader, 0, &completionStatus);

if (ccode)
{   /* handle error here */
   return ccode;
}

 /* Wait for the function to complete its task. The engine can do other
things while waiting. */

while (completionStatus == NWSMSD_WAIT_PENDING)
     MyDelay();

if (completionStatus == 0)
   NWSMGetMediaHeaderInfo((BUFFERPTR)headerBuffer->headerBuffer,
headerBuffer->headerSize, &mediaInfo);
```

NWSMGetRecordHeaderOnly

Returns only the record/subrecord header information.

Syntax

```
#include <smsutapi.h>

CCODE NWSMGetRecordHeaderOnly (
   BUFFERPTR  *buffer,
   UINT32     *bufferSize,
   NWSM_RECORD_HEADER_INFO
              *recordHeaderInfo);
```

Parameters

buffer

   (IN/OUT) Points to the beginning of the SIDF data or subdata set in a transfer buffer on input. Points to the data set data on output.

bufferSize

   (IN/OUT) Points to the size of the transfer buffer minus the transfer buffer header size on input. Points to the size of the data left in the transfer buffer on output.

recordHeaderInfo

   (OUT) Points to the next section (record or subrecord).

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER: The section does not begin with a synchronization field.

Remarks

To get the data set information and data set data, call NWSMGetDataSetInfo.

NWSMGetRecordHeaderOnly indicates one of two conditions: a new record or a record containing data that spans a transfer buffer.

A new record is indicated as follows:
```
recordHeaderInfo->isSubRecord = FALSE;
recordHeaderInfo->dataSetInfoRetrieved = DATA_SET_INFO_NOT_STARTED;
```

A record containing spanned data is indicated as follows:
```
recordHeaderInfo->isSubRecord = TRUE;
recordHeaderInfo->dataSetInfoRetrieved = DATA_SET_INFO_DOES_NOT_EXIST;
```

See Also

NWSMGetRecordHeaderOnly Example

```
/* Simplified example from the FILES.C file. It assumes that there is a
media transfer buffer, and that it
starts on the first record. */

UINT32 bufferSize, maxTransferBufferSize, transferBufferDataOffset;
BUFFERPTR bufferPtr;
NWSM_RECORD_HEADER_INFO recordHeaderInfo = {0};

while(there are buffers on the media)
{

/* This function is not defined. It gets the next transfer buffer from
the media and points to the beginning of
the transfer buffer. */
   GetTheNextBuffer(&bufferPtr);

/* Point to the transfer buffer data. maxTransferBufferSize and
transferBufferDataOffset are returned by
NWSMSDSessionOpenForReading.*/
   bufferSize = maxTransferBufferSize - transferBufferDataOffset;
   bufferPtr += bufferSize;

while (bufferSize) /* While there is data in the transfer buffer. */
{     /* Get the next record/subrecord from the transfer buffer. */
      NWSMGetRecordHeaderOnly(&bufferPtr, &bufferSize,
&recordHeaderInfo);

     /* If a record/subrecord, get the data from the transfer buffer.
        Note: DATA_SET_INFO_SPANNED is set by NWSMGetDataSetInfo. */
    if
((recordHeaderInfo.dataSetInfoRetrieved==DATA_SET_INFO_NOT_STARTED)
       || (recordHeaderInfo.dataSetInfoRetrieved ==
DATA_SET_INFO_SPANNED))
         NWSMGetDataSetInfo(&bufferPtr, &bufferSize,
&recordHeaderInfo);

    if (!bufferSize)
      break; /* All data retrieved from current transfer buffer. Break
and
              get the next transfer buffer. */

   if (recordHeaderInfo.dataSetInfoRetrieved ==
DATA_SET_INFO_COMPLETE)
     { /* dataSetName and scanInformation can now be used. */
     }
      /* Update the buffer's information. */
      bufferPtr += recordHeaderInfo.recordSize;
```

```
        bufferSize -= recordHeaderInfo.recordSize;
    }   /* end while(bufferSize) */
}
```

NWSMGetSessionHeaderInfo

Separates an SIDF session header into it various components.

Syntax

```
#include <smsutapi.h>

CCODE NWSMGetSessionHeaderInfo (
   BUFFERPTR            headerBuffer,
   UINT32               headerBufferSize,
   NWSM_SESSION_INFO  *sessionInfo);
```

Parameters

headerBuffer

    (IN) Points to the buffer containing the SIDF session header data that can be returned by NWSMSDOpenSessionForReading.

headerBufferSize

    (IN) Specifies the size of headerBuffer in bytes.

sessionInfo

    (OUT) Points to the unformatted session information.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW
0xFFFBFFF9	NWSMUT_INVALID_FIELD_ID
0xFFFBFFF0	NWSMUT_BUFFER_UNDERFLOW

NWSMPadBlankSpace

Inserts a blank space section if there is room in the buffer, or fills the unused area of a buffer with zeros.

Syntax

```
#include <smsutapi.h>

void NWSMPadBlankSpace (
   BUFFERPTR   bufferPtr,
   UINT32      unusedSize);
```

Parameters

bufferPtr

   (OUT) Points to the area in the buffer to fill with blank spaces or zeros.

unusedSize

   (IN) Specifies the size of the area in buffer to pad.

Remarks

If unusedSize is less than MIN_BLANK_SPACE_SECTION_SIZE, the unused area is filled with zeros. If unusedSize is larger than or equal to MIN_BLANK_SPACE_SECTION_SIZE, the Blank Space section is put into the unused area.

NWSMPadBlankSpace Example

```
#define BUFFER_SIZE 2048 /* Arbitrary size. */
char buf[BUFFER_SIZE];
BUFFERPTR bufferPtr = buf;
UINT32 unusedSize = BUFFER_SIZE;

/* Put data into buffer pointed to by bufferPtr. */
/* Update buffer information. */
bufferPtr = bufferPtr + amount of data put into buffer;
unusedSize -= (bufferPtr - buf);

/* Pad unused buffer area. */
NWSMPadBlankSpace(bufferPtr, unusedSize);
```

NWSMSetMediaHeaderInfo

Formats media header information into an SIDF compliant media header.

Syntax

```
#include <smsutapi.h>

CCODE NWSMSetMediaHeaderInfo (
   NWSM_MEDIA_INFO  *mediaInfo,
   BUFFERPTR          buffer,
   UINT32             bufferSize,
   UINT32               *headerSize);
```

Parameters

mediaInfo

   (IN) Points to the media header information to format.

buffer

   (OUT) Points to the buffer to receive the formatted media header information.

bufferSize

   (IN) Specifies the size of buffer.

headerSize

   (OUT) Points to the size (in bytes) of the formatted media header.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFF0	NWSMUT_BUFFER_UNDERFLOW: The buffer is out of space.
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER: One or more of the parameters were set to an invalid value.

See Also

NWSMGetMediaHeaderInfo (page 263)

Example

```
/* Some functions and structures are defined by SMS DI.
#include <smssdapi.h>
#include <smsdefns.h>
#define MY_BUFFER_SIZE 1024 /* arbitrary buffer size */
```

```
CCODE ccode;
NWSM_MEDIA_INFO mediaInfo;
NWSMSD_HEADER_BUFFER *mediaHeader;
UINT32 dataFormatType, mediumCreatedDateTime, setCreatedDateTime,
mediaNumber = 0;
NWSMSD_MEDIA_HANDLE mediaHandle;

/* Get date and time of the media set's creation date and time */
setCreatedDateTime = NWSMGetCurrentDateAndTime();

/* Setup the media header buffer information. */
mediaHeader = (NWSMSD_HEADER_BUFFER *)
malloc(sizeof(NWSMSD_HEADER_BUFFER) + MY_BUFFER_SIZE);
mediaHeader->bufferSize = MY_BUFFER_SIZE + 1;
mediaHeader->overflowSize = 0;

/* Create the media header information */
mediumCreatedDateTime = NWSMGetCurrentDateAndTime();
NWSMDOSTimeToECMA(mediumCreatedDateTime, &(mediaInfo.timeStamp));
strcpy(mediaInfo.label, "The media set label");
mediaInfo.number = ++mediaNumber;

/* Format the media header information according to SIDF. */
NWSMSetMediaHeaderInfo(&mediaInfo, (BUFFERPTR)mediaHeader-
>headerBuffer, mediaHeader->bufferSize,
&mediaHeader->headerSize);

/* Label the media. */
dataFormatType = NWSMSD_DFT_SIDF;
ccode = NWSMSDMediaLabel(sdiConnection, mediaHandle, dataFormatType,
mediaHeader, 0, &completionStatus);
```

NWSMSetNewRecordHeader

Creates a record or a subrecord header.

Syntax

```
#include <smsutapi.h>

CCODE NWSMSetNewRecordHeader(
   BUFFERPTR  *buffer,
   UINT32     *bufferSize,
   UINT32     *bufferData,
   NWBOOLEAN   setCRC,
   NWSM_RECORD_HEADER_INFO
              *recordHeaderInfo);
```

Parameters

buffer

(OUT) Points to the beginning of the buffer to contain the SIDF data set header on input. Points to where the data set data should be placed on output.

bufferSize

(IN/OUT) Points to the amount of free space in buffer.

bufferData

(IN/OUT) Points to the total number of bytes written to buffer.

setCRC

(IN) Specifies the CRC flag:

CRC_NO The CRC field is set to zero.
CRC_YES A CRC value is calculated for the entire data set.
CRC_LATER The engine will generate the CRC value.

recordHeaderInfo

(IN/OUT) Points to the data for the record header and data set information.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER
0xFFFBFFFA	NWSM_BUFFER_OVERFLOW
0xFFFBFFF0	NWSM_BUFFER_UNDERFLOW

Remarks

If a record is being created, set recordHeaderInfo.isSubRecord to FALSE. When NWSMSetNewRecordHeader is called, a data set header is created and placed into buffer. The data set information is formatted according to SIDF and placed, along with the data set' s data, after the header.

If a subrecord is being created, set recordHeaderInfo.isSubRecord to TRUE. When NWSMSetNewRecordHeader is called, a data set subheader is created and placed into buffer. Call NWSMUpdateRecordHeader to place the rest of the data into buffer.

If buffer already contains 36 KB of data, bufferData must be set to 32 KB. NWSMSetNewRecordHeader will add the number of bytes copied into buffer to the current value.

If CRC is specified, the CRC data for the last section field is set to zero.

The following figure shows the buffer area represented by bufferData and bufferSize:
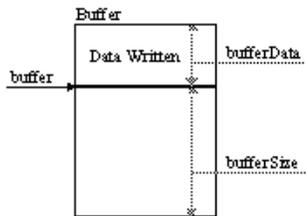
See Also

NWSMSetSessionHeaderInfo

Formats the session header information into an SIDF compliant session header.

Syntax

```
#include <smsdefns.h>
#include <smsutapi.h>

CCODE NWSMSetSessionHeaderInfo (
   NWSM_SESSION_INFO  *sessionInfo,
   BUFFERPTR           buffer,
   UINT32              bufferSize,
   UINT32             *headerSize);
```

Parameters

sessionInfo

   (IN) Points to the session header information to be formatted.

buffer

   (OUT) Points to the buffer to receive the formatted session information.

bufferSize

   (IN) Specifies the size of buffer.

headerSize

   (OUT) Points to the size of the formatted header.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFF0	NWSMUT_BUFFER_UNDERFLOW: An internal error occurred.
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER: An internal error occurred.

Remarks

If you are using SMS DI, call NWSMSDOpenSessionForWriting to write the header out to the medium.

To get the SMDR name, call NWSMListSMDRs.

See Also

NWSMGetSessionHeaderInfo (page 268), NWSMListSMDRs (page 301)

NWSMSetSessionHeaderInfo Example

```
#include <smstsapi.h>
#include <smsutapi.h>
#include <smsdefns.h>

NWSM_SESSION_INFO sessionInfo;
NWSMSD_HEADER_BUFFER *sessionHeaderInfo;
UINT32 sessionDateTime;

/* Setup the session header buffer information. maxTransferBufferSize
is set by NWSMSDSessionOpenForWriting */
sessionHeaderInfo = (NWSMSD_HEADER_BUFFER *)
malloc(sizeof(NWSMSD_HEADER_BUFFER) + maxTransferBufferSize - 1);
sessionHeaderInfo->bufferSize = maxTransferBufferSize;

/* Setup the session header information. */
sessionDateTime = NWSMGetCurrentDateAndTime();
NWSMDOSTimeToECMA(sessionDateTime, &(sessionInfo.timeStamp));
strcpy(sessionInfo.description, "Backup file server");
strcpy(sessionInfo.softwareName, "SBackup");
strcpy(sessionInfo.softwareType, "SMS");
strcpy(sessionInfo.softwareVersion, "Ver 4.x");
sessionInof.serviceName = name from NWSMTSListTargetServices or
NWSMTSScanTargetServices;
NWSMTSGetTargetServiceType(connection, (STRING)sessionInfo.sourceName,
(STRING)sessionInfo.sourceType,
(STRING)sessionInfo.sourceVersion);

/* Format the session information according to SIDF, and put the
resulting data into the session header buffer. */
NWSMSetSessionHeaderInfo(&sessionInfo, sessionHeaderInfo-
>headerBuffer, sessionHeaderInfo->bufferSize,
&sessionHeaderInfo->headerSize);
```

NWSMUpdateRecordHeader

Updates a record/subrecord's size and CRC information.

Syntax

```
#include <smsutapi.h>

CCODE NWSMUpdateRecordHeader (
   NWSM_RECORD_HEADER_INFO  *recordHeaderInfo);
```

Parameters

recordHeaderInfo

  (IN/OUT) Points to the record/subrecord to update on input.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful

Remarks

Call NWSMUpdateRecordHeader after inserting data into a transfer buffer.

See Also

NWSMGetRecordHeaderOnly (page 265), NWSMSetNewRecordHeader (page 272)

6.8 SMDF Functions

SMDF functions simplify the building and parsing of SIDF fields.

◆ "SMDFSubUINT64" on page 296

SMDFAddUINT64

Adds two UINT64 values together.

Syntax

```
#include <smsutapi.h>

CCODE SMDFAddUINT64 (
   UINT64  *term1,
   UINT64  *term2,
   UINT64  *sum);
```

Parameters

term1

   (IN) Points to the first term to be added.

term2

   (IN) Points to the second term to be added.

sum

   (OUT) Points to the sum of term1 and term2.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW: The sum of term1 and term2 produces a number to large to be contained in term1

SMDFDecrementUINT64

Subtracts a UINT32 value from a UINT64 value.

Syntax

```
#include <smsutapi.h>

CCODE SMDFDecrementUINT64 (
   UINT64  *term1,
   UINT32   term2);
```

Parameters

term1

(IN/OUT) Points to the value of term1 on input. Points to the results of term1 - term2 on output.

term2

(IN) Specifies the amount term1 is to be decremented by.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFF0	NWSMUT_BUFFER_UNDERFLOW
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW

SMDFDecrementUINT64 Example

```
UINT64 term1;
UINT32 term2 = 0xF;

SMDFPutUINT64(&term1, 0xFFF);
SMDFDecrementUINT64(&term1, term2);
```

SMDFGetFields

Separates an SIDF section into its various components.

Syntax

```
#include <smsutapi.h>

CCODE SMDFGetFields (
   UINT32                   headFID,
   NWSM_GET_FIELDS_TABLE   table[ ],
   BUFFERPTR               *buffer,
   UINT32                  *bufferSize);
```

Parameters

headFID

(IN) Specifies the FID value of the first field in the section that the engine expects to find in buffer.

table

(IN/OUT) Points to the table to receive the parsed section.

buffer

(IN/OUT) Points to the beginning of the section to be parsed in the buffer on input. Points to the next section on output.

bufferSize

(IN/OUT) Points to the size of buffer on input. Points to the buffer's size minus the size of the returned section on ouput.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW: buffer is to small to contain the fields of table.
0xFFFBFFF0	NWSMUT_BUFFER_UNDERFLOW: A field's data spans buffer.
0xFFFBFFF9	NWSMUT_INVALID_FIELD_ID: headFid does not match the section FID in buffer.

Remarks

Before you call SMDFGetFields, set found to FALSE and fid to the field to look for.

SMDFGetFields parses a section into a table by calling SMDFGetNextField repetitively. SMDFGetFields does not handle buffer overflow or buffer underflow, but returns a completion code indicating the condition.

If SMDFGetFields returns an error, buffer and bufferSize may not contain their original values.

Each element in the table array represents one field in the section. The order of the elements, except the first and last elements (and where noted by SIDF), is not important to SMDFGetFields. If SMDFGetFields encounters a field that is not defined in table, the field is ignored.

See Also

SMDFGetFields Example

```
#define SECTION_FID 0
#define NAME_INDEX 1
UINT16 syncData;
/* buffer and bufferSize are set elsewhere in the program. */
BUFFERPTR origBufferPtr = buffer;
UINT32 origBufferSize = bufferSize;

NWSM_GET_FIELDS_TABLE table[] =
{
   /* Section FID is FAKE_FID */
   { FAKE_FID, NULL, sizeof(UINT16), FALSE },
   { NWSM_SOURCE_NAME, NULL, NWSM_MAX_TARGET_SRVC_NAME_LEN, FALSE },
   /* End of table */
   { NWSM_END }
};

/* Link the data fields to the variables */
table[SECTION_FID].data = &syncData;
table[NAME_INDEX].data = calloc(1, NWSM_MAX_TARGET_SRVC_NAME_LEN);

if (SMDFGetFields(FAKE_FID, table, &buffer, &bufferSize) != 0)
{
   /* Error occurred, reset buffer information to known values. */
   buffer = origBufferPtr;
   bufferSize = origBufferSize;
}
```

SMDFGetNextField

Separates an SIDF compliant field into its various components.

Syntax

```
#include <smsutapi.h>

CCODE SMDFGetNextField (
   BUFFERPTR        buffer,
   UINT32           bufferSize,
   SMDF_FIELD_DATA  *field);
```

Parameters

buffer

   (IN) Points to the buffer containing the field to be parsed.

bufferSize

   (IN) Specifies the size of buffer.

field

   (OUT) Points to the structure to receive the parsed data.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful

Remarks

SMDFGetNextField parses the next field in buffer into its various field components. If buffer contains a partial field (because the field spans the buffer), field.dataOverFlow is set to a positive value. You must save this data, call the function again with the next buffer, and append the spanned data.

The FID from buffer is converted to a byte stream when it is placed into field.fid.

See Also

SMDFGetNextField Example

```
CCODE GetFields(UINT32 sectionFID, NWSM_GET_FIELDS_TABLE section[],
BUFFERPTR *buffer, UINT32 *bufferSize)
{
   CCODE ccode;
```

```
   SMDF_FIELD_DATA field;
   int index;

   /* Get the first field from buffer. */
   if ((ccode = SMDFGetNextField(*buffer, *bufferSize, &field)) != 0)
      goto Return;

 /* If the first field is not the section field, return. */
   if (field.fid != sectionFID)
   {
      ccode = NWSMUT_INVALID_FIELD_ID;
      goto Return;
    }

   /* Adjust the buffer information to point to the next field. */
   *buffer += field.bytesTransfered;
   *bufferSize -= field.bytesTransfered;


   /* Search through buffer for the fields specified in section until
you find the section's ending field. */
   while(1)
   {
      if ((ccode = SMDFGetNextField(*buffer, *bufferSize, &field)) !=
0)
         goto Return;

      /* Adjust the buffer information to point to the next field. */
      *buffer += field.bytesTransfered;
      *bufferSize -= field.bytesTransfered;

      /* If the section's last field is found, break. */
      if (field.fid == sectionFID)
         break;

      /* See if field is in section; if it is, put field's information
into section. The code will loop until field
matches a corresponding element in section, or when all elements in
section are searched.*/

      for (index = 0; section[index].fid != NWSM_END; index++)
      {
        /* If the field was previously found, break and get the next
field. */
         if (section[index].found)
            break;

      /* Put the field into section. */
         if (field.fid == section[index].fid)
         {
            UINT32 dataSize;

       /* Check if the field's data will fit into section's buffer */
            SMDFGetUINT64(&field.dataSize, &dataSize);
```

```
                if (dataSize > section[index].dataSize)
                {
                   ccode = NWSMUT_BUFFER_OVERFLOW;
                   goto Return;
                }

                memmove(section[index].data, field.data, dataSize);
                section[index].dataSize = dataSize;
                section[index].found = TRUE;
                break;
            }
        } /* end of for loop */
    } /* end of while loop */

Return:
    return(ccode);
}
```

SMDFGetUINT64

Returns the lower four bytes from a UINT64 value.

Syntax

```
#include <smsutapi.h>

CCODE SMDFGetUINT64 (
   UINT64  *src,
   UINT32  *dest);
```

Parameters

src

   (IN) Points to the UINT64 value.

dest

   (OUT) Points to the lower 4 bytes of the UINT64 value.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW: The upper 4 bytes of src are nonzero.

See Also

SMDFSetUINT64 (page 295)

SMDFIncrementUINT64

Adds a UINT32 value to a UINT64 value.

Syntax

```
#include <smsutapi.h>

CCODE SMDFIncrementUINT64 (
   UINT64  *term1,
   UINT32   term2);
```

Parameters

term1

> (IN/OUT) Points to the value of the first term on input. Points to the sum of term1 and term2 on ouptut.

term2

> (IN) Specifies the second term.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW: The sum of term1 and term2 produces a number to large to be contained in term1.

SMDFPutFields

Formats data into an SIDF section.

Syntax

```
#include <smsutapi.h>

CCODE SMDFPutFields (
    NWSM_FIELD_TABLE_DATA    table[ ],
    BUFFERPTR               *buffer,
    UINT32                  *bufferSize,
    UINT32                   crcFlag);
```

Parameters

table[ ]

(IN) Points to the table that represents the section to put into buffer.

buffer

(OUT) Points to the part of the buffer where the section should be placed on input. Points to the beginning of the next section on output.

bufferSize

(IN/OUT) Points to the amount of free space in buffer.

crcFlag

(IN) Specifies the CRC flag:

CRC_YES Generate CRC now
CRC_NO Do not generate CRC
CRC_LATER Generate the CRC at a later time

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFF0	NWSMUT_BUFFER_UNDERFLOW: The offset to end value cannot fit into the provided space.
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW: The buffer is too small. Increase the buffer size and call SMDFPutFields again.
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER

Remarks

SMDFPutFields checks for boundary conditions such as field and buffer overflow or underflow, but it does not handle buffer overflow or underflow conditions or check for valid FIDs.

SMDFPutFields converts table into a section and writes it into buffer by calling SMDFPutNextField.

SMDFPutFields sets the size of the data to 4 and the field data to 0xA55A (synchronization data). The data for Offset to End field is set if table contains this field.

crcFlag affects the data portion of the last field in the section. If no CRC_NO is passed, the CRC data is zero. If CRC_LATER is passed, the CRC data is 0xFFFFFFFF. You must decrement buffer by four bytes (UINT32) before inserting its CRC value into buffer.

See Also

SMDFPutFields Example

```
/* This example shows how to use the structures. */

#define DEBUG_CODE 1
#define OFFSET_TO_END_INDEX 4
#define VOLUME_NAME_INDEX 5
#define HEADER_STRING "NWSM_VOLUME_RESTRICTIONS"
#define HEADER_STRING_LEN sizeof(NWSM_HEADER_DEBUG_STRING)

char buffer[255]; /* arbitrary size */
UINT32 bufferSize = 255;
UINT64 offsetToEnd = UINT64_ZERO;

NWSM_FIELD_TABLE_DATA volumeRestrictionHeaderTable[] =
{
   /* Signal SMDFPutFields that the second field in the table is the
beginning of the section.
Create the synchronization data for that field. */
   { { NWSM_BEGIN } },

   /* The next field contains the section FID, used to construct the
start and end Fields of the section. */
   { {NWSM_VOLUME_RESTRICTIONS, UINT64_ZERO, NULL, 0, UINT64_ZERO }, 0,
NULL, 0 },

   /* The field is not required, but it is helpful when searching. It
places a string describing the section into
the data stream. */
   { {NWSM_HEADER_DEBUG_STRING, UINT64_ZERO, HEADER_STRING},
HEADER_STRING_LEN, NULL, HEADER_STRING_LEN},

   /* For this field we want the address of the data area in buffer */
   { { NWSM_OFFSET_TO_END, UINT64_ZERO, NULL, 0, UINT64_ZERO }, 4,
GET_ADDRESS, 4},

   /* This field needs a string, but the information will not be known
until the program is running. */
   { { NWSM_VOLUME_NAME, UINT64_ZERO, NULL, 0, UINT64_ZERO }, 0, NULL,
```

```
NWSM_VARIABLE_SIZE},

   /* This field is required to let the function know that this the end
of the table. The section header field does
not repeat. */
   { {NWSM_END } }
},
volumeRestrictionEntryTable[] =
/* Another table. Note that this one does not have a "beginning" field.
*/
{
   #if defined(NETWARE_V320)
   { { NWSM_VOLUME_RSTRCTNS_NAME, UINT64_ZERO, NULL, 0, UINT64_ZERO },
0, NULL, NWSM_VARIABLE_SIZE },
   #else
      { { NWSM_VOLUME_RSTRCTNS_ID, UINT64_ZERO, NULL, 0, UINT64_ZERO },
0, NULL, NWSM_VARIABLE_SIZE },
   #endif
      { { NWSM_VOLUME_RSTRCTNS_LIMIT, UINT64_ZERO, NULL, 0, UINT64_ZERO
}, 0, NULL, 0 },
      { {    NWSM_END } }
};

/* Set up the variable data for sourceVersion to the table. Set
volumeName somewhere in the program. */
SMDFPutUINT64(
&volumeRestrictionHeaderTable[VOLUME_NAME_INDEX].field.dataSize,
      (UINT32)strlen(volumeName));
volumeRestrictionHeaderTable[VOLUME_NAME_INDEX].field.data =
volumeName;
volumeRestrictionHeaderTable[VOLUME_NAME_INDEX].sizeOfData =
      (UINT32)strlen(volumeName);
volumeRestrictionHeaderTable[VOLUME_NAME_INDEX].dataSizeMap =
      (UINT8)strlen(volumeName);
SMDFPutFields(volumeRestrictionHeaderTable, &buffer, &bufferSize,
CRC_NO);

/* Now set the offset to end value. */
SMDFPutUINT64(&offsetToEnd, buffer -
       ((BUFFERPTR)volumeRestrictionHeaderTable[
       OFFSET_TO_END_INDEX].addressOfData +

volumeRestrictionHeaderTable[OFFSET_TO_END_INDEX].dataSizeMap));

memcpy(volumeRestrictionHeaderTable[OFFSET_TO_END_INDEX].addressOfData
,
      &offsetToEnd,
volumeRestrictionHeaderTable[OFFSET_TO_END_INDEX].dataSizeMap);
```

SMDFPutNextField

Formats data into an SIDF compliant field.

Syntax

```
#include <smsutapi.h>

CCODE SMDFPutNextField (
   BUFFERPTR          buffer,
   UINT32             bufferSize,
   SMDF_FIELD_DATA    *field,
   UINT8              dataSizeMap,
   UINT32             sizeOfData);
```

Parameters

buffer

   (IN) Points to the buffer to receive the field.

bufferSize

   (IN) Specifies the amount of free space in buffer.

field

   (IN) Points to the field information to format.

dataSizeMap

   (IN) Specifies where to find the data size for field.data.

sizeOfData

   (IN) Specifies how many bytes of data are in field.data.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER

Remarks

SMDFPutNextField puts one field into a buffer and indicates how many bytes were moved into buffer.

Even if SMDFPutNextField successfully completes, you should always check for a data overflow condition. If the data overflows, field.dataOverflow contains the amount of data that was not transferred.

If field.data contains part of the total field data, you must put the rest of the data into the buffer. To figure out where to put the rest of the data, track the offset into buffer through field.bytesTransferred.

The value of dataSizeMap depends upon the amount of data in field.data. If the total amount of data is less than 128 bytes, set dataSizeMap to the data's size. If the total amount of data is 128 bytes or more, set dataSizeMap to NWSM_VARIABLE_SIZE and field.dataSize to the total amount of data that the field will contain.

field.dataSize specifies the total amount of data the field will contain. This value may or may not be the size of data because data might point to only part of the field's data. For example, if the data set's size is 64 KB, but only 32 KB is being transferred, this field contains 64 KB.

See Also

SMDFPutNextField Example

```
/* Places a whole section with no offset to end field. */

PutSection(BUFFERPTR buffer, UINT32 bufferSize, NWSM_FIELD_TABLE_DATA
*section)
{
   UINT16 sync = 0xA55A, i;
   SMDF_FIELD_DATA lastField = {{0},0};
   BUFFERPTR begin = buffer;
   UINT32 crc = 0xFFFFFFFF;

   /* The first element is the section fid. */
   section->sizeOfData = section->dataSizeMap = sizeof(UINT16);
   section->data = &sync;
   SMDFPutNextField(buffer, bufferSize, &section->field, section-
>dataSizeMap, section->sizeOfData);
   buffer += section->field.bytesTransferred;
   bufferSize -= section->field.bytesTransferred;

   for (i = 1; section[i].field.fid != NWSM_END; i++)
   {
      SMDFPutNextField(buffer, bufferSize, &section[i].field,
           section[i].dataSizeMap, section[i].sizeOfData);
      buffer += section[i].field.bytesTransferred;
      bufferSize -= section[i].field.bytesTransferred;
   }

   /* Set the last field into the buffer */
   lastField.fid = section[0].field.fid;
   crc = NWSMGenerateCRC(buffer - begin, crc, begin);
   lastField.data = &crc;
   SMDFPutUINT64(&lastField.dataSize, sizeof(UINT32));
   SMDFPutNextField(buffer, bufferSize, &lastField,
(UINT8)sizeof(UINT16),
```

```
                    (UINT32)sizeof(UINT16));
        }
```

SMDFPutUINT64

Converts a UINT32 data type to a UINT64 data type.

Syntax

```
#include <smsutapi.h>

void SMDFPutUINT64 (
   UINT64  *dest,
   UINT32   src);
```

Parameters

dest

(OUT) Points to the UINT64 equivalent of src.

src

(IN) Specifies the value to be converted to a UINT64.

SMDFSetUINT32Data

Copies four or less bytes from a buffer to a UINT32 variable.

Syntax

```
#include <smsutapi.h>

CCODE SMDFSetUINT32Data (
    UINT64     *dataSize,
    BUFFERPTR   buffer,
    UINT32     *data);
```

Parameters

dataSize

   (IN) Points to the number of bytes the data size occupies (maximum of 4).

buffer

   (IN) Points to the data to copy to data.

data

   (OUT) Points to the data from buffer.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW: dataSize is more than 4.

SMDFSetUINT64

Copies eight or less bytes from a buffer to a UINT64 variable.

Syntax

```
#include <smsutapi.h>

CCODE SMDFSetUINT64 (
   UINT64  *data,
   void    *buffer,
   UINT16   dataSize);
```

Parameters

data

(OUT) Points to the data copied from buffer.

buffer

(IN) Points to the data to copy.

dataSize

(IN) Specifies the number of bytes to copy (maximum of 8).

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW: dataSize is greater than 8.

Remarks

SMDFSetUINT64 calls memcpy to copy the data.

If buffer is NULL, data is set to 0.

See Also

SMDFGetUINT64 (page 285)

SMDFSubUINT64

Subtracts two UINT64 values.

Syntax

```
#include <smsutapi.h>

CCODE SMDFSubUINT64 (
    UINT64  *term1,
    UINT64  *term2,
    UINT64  *diff);
```

Parameters

term1

    (IN) Points to the left-hand operand.

term2

    (IN) Points to the right-hand operand.

diff

    (OUT) Points to the result of term1 - term2.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFF0	NWSMUT_BUFFER_UNDERFLOW: term2 is larger than term1.

See Also

SMDFDecrementUINT64 (page 279)

6.9 SMDR Functions

This section contains SMDR functions.

- "NWSMConvertError" on page 297
- "NWSMGetRequestorVersionInfo" on page 298
- "NWSMGetSMSModuleVersionInfo" on page 299
- "NWSMGetResponderVersionInfo" on page 300
- "NWSMListSMDRs" on page 301

NWSMConvertError

Returns the string that represents the specified completion code.

Syntax

```
#include <sms.h>

CCODE NWSMConvertError (
   UINT32   connection,
   CCODE    error,
   char    *message);
```

Parameters

connection

   (IN) Specifies the TSA or SMS DI connection handle.

error

   (IN) Specifies the completion code to translate.

message

   (OUT) Points to the string that describes error (initialize to
   NWSM_MAX_ERROR_STRING_LEN + 1 bytes).

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFEFFEF	NWSMDR_INVALID_MESSAGE_NUMBER

Remarks

NWSMConvertError accepts error codes that begin with NWSMTS, NWSMDR, and NWSMUT.

NWSMGetRequestorVersionInfo

Queries the local SMDR for its version information.

Syntax

```
#include <sms.h>

CCODE NWSMGetRequestorVersionInfo (
   UINT32                          connection,
   NWSM_MODULE_VERSION_INFO *versionInfo);
```

Parameters

connection

   (IN) Specifies the connection information that NWSMConnectToTSA returned.

versionInfo

   (OUT) Points to a NWSM_MODULE_VERSION_INFO structure.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE

Remarks

You are responsible for freeing the memory held by versionInfo.

NWSMGetSMSModuleVersionInfo

Queries the connected SMS module for its version information.

Syntax

```
#include <sms.h>

CCODE NWSMGetSMSModuleVersionInfo (
   UINT32                     connection,
   NWSM_MODULE_VERSION_INFO  *info);
```

Parameters

connection

   (IN) Specifies a TS API or SMS DI connection handle.

info

   (OUT) Points to a NWSM_MODULE_VERSION_INFO structure containing information
   about the module.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFE9	NWSMDR_INVALID_CONTEXT

Remarks

If a TS API connection handle is passed to connection, you must be connected to the TSA and
Target Service.

NWSMGetResponderVersionInfo

Queries the responder used by the specified connection for its version information.

Syntax

```
#include <sms.h>

CCODE NWSMGetResponderVersionInfo (
   UINT32                        connection,
   NWSM_MODULE_VERSION_INFO  *versionInfo);
```

Parameters

connection

   (IN) Specifies the connection information that NWSMConnectToTSA returned.

versionInfo

   (OUT) Points to a NWSM_MODULE_VERSION_INFO structure.

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFE9	NWSMDR_INVALID_CONTEXT

Remarks

If the only SMDR is local, its version information is returned.

You are responsible for freeing the memory held by versionInfo.

NWSMListSMDRs

Returns the names of all active SMDRs.

Syntax

```
#include <smsdrapi.h>

CCODE NWSMListSMDRs (
   char *pattern,
   NWSM_NAME_LIST **tsaNameList);
```

Parameters

pattern

   (IN) Specifies the pattern:

   "*" Return all SMDR names

   "*xxxx" Return all SMDR names that end with "xxxx"

   "xxxx*" Return all SMDR names that begin with "xxxx"

   "xxxx" Find the SMDR named "xxxx"

tsaNameList

   (OUT) Points to a block of memory containing the list of specified SMDRs (cannot be a NULL pointer but can be a pointer set to a NULL pointer).

Return Values

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFBFFFB	NWSMUT_OUT_OF_MEMORY
0xFFFEFFF6	NWSMDR_NO_SUCH_SMDR
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFE9	NWSMDR_INVALID_CONTEXT

Remarks

For clusters, the virtual NCP server name is listed.

After an SMDR name is selected, it is passed to NWSMListTSAs to find the available TSAs that are local to that SMDR. The SMDR name is the same as the Target Service's name. NWSMListSMDRs may speed up the listing of SMS modules.

Call NWSMFreeNameList to free the memory allocated to nameList.

Utility Field Macros

<div style="text-align: right; font-size: 3em;">7</div>

Field Macros are macros that manipulate and return information about SIDF data at the bit level.

In the field macros section, it is important to remember that formatted FID values are put into a section as a byte stream where the most significant byte is first, the second most significant byte is next, etc.

Where noted, the macros depend upon longFid (a local variable) being set before the macros are called.

If the FID is long, the engine sets longFID to TRUE. If the FID is not long, longFID is set to FALSE. The following function is used to set longFID:

```
int IsLong(unsigned char *fid)
{
   if(*fid & 0x80)
   {
      //fid is a Standard or OS FID
      if (*fid == 0x80 || ((*fid & 0xC0) == 0x80))
         ++fid; // fid is Standard or OS FID
      else
         fid += 2; // fid is Developer FID

      if (*fid & 0x80)
            return TRUE; // fid is a long FID
   else
            return FALSE; //fid is a Short FID
   }

   else
   return FALSE; //fid is Small FID
}
```

The function is used as follows:

```
UINT8 longFid;
char *fid;
UINT32 fidValue = 0x8000C0;
...
fid = (char *)&fidValue;
longFid = IsLong(fid);
```

A special note is needed for macro parameters. The contents of non-pointer parameters in macros can be changed. The parameter prototypes will not show as pointer types because the macro does not use a pointer to change the parameter's contents. For example, macro "typedef Demo(param) ((param) = 0)" changes the contents of param.

SIDF_GetFixedSize

Indicates if a FID has a fixed or variable data size.

Syntax

```
#include <sidf.h>

int SIDF_GetFixedSize (
    UINT32   fid);
```

Parameters

fid

   (IN) Specifies the FID value to be identified.

Return Values

If SIDF_GetFixedSize is successful, it returns nonzero if the FID is fixed and zero if it is not fixed.

The following table lists the return values associated with the function.

0x00000000	Not fixed: fid is not a fixed FID.
nonzero	Fixed size value: size of fixed fid.

Remarks

longFid must be set before you call this macro.

Example

```
UINT32 fid;
BUFFERPTR bufferPtr = buffer;
UINT8 longFid;

longFid = IsLong((char *)&fid);
if (SMDFFixedFid(fid))
   /* FID is fixed. */
   ...
else
   /* FID is variable. */
   ...
```

SMDFSizeOfFID

Determines the size of a FID.

Syntax

```
#include <smsutapi.h>

int SMDFSizeOfFID (
   UINT32   fid);
```

Parameters

fid

   (IN) Specifies the FID to analyze.

Return Values

The following table lists the return values associated with the function.

nonzero	FID size (1 through 4 bytes)

Remarks

The return value does not describe the size of the data associated with the FID, but only the FID's size.

SMDFSizeOfFieldData

Returns the size information in data size format 2 for a given data size.

Syntax

```
#include <smsutapi.h>

int SMDFSizeOfFieldData (
   UINT64  dataSize,
   UINT8   dataSizeMap);
```

Parameters

dataSize

> (IN) Specifies the size of the data size descriptor.

dataSizeMap

> (OUT) Specifies a size format 2 descriptor that indicates the number of size bytes that should follow the descriptor.

Return Values

The following table lists the return values associated with the function.

0x0	Invalid data size value. The data's size is between 0 and 127 bytes (size format 1 should be used).
0x1	The data size value will occupy one byte.
0x2	The data size value will occupy 2 bytes.
0x4	The data size value will occupy 4 bytes.
0x8	The data size value will occupy 8 bytes.

Remarks

For data size values that fall within the range of Size Format 2, SMDFSizeOfFieldData returns a size descriptor and indicates the number of bytes the data size value will occupy. These values are used to fill in the data size values of an SIDF field.

dataSizeMap returns one of the following values:

0x00 dataSize contains an invalid value
0x80 20 size bytes follow
0x81 21 size bytes follow
0x82 22 size bytes follow
0x83 23 size bytes follow

SMDFBitNIsSet

Checks if a bit is set.

Syntax

```
#include <smsutapi.h>

int SMDFBitNIsSet (
   unsigned char   c);
```

Parameters

c

   (IN) Specifies the bit to test.

Return Values

The following table lists the return values associated with the function.

0x00000000	Bit N is not set.
nonzero	Bit N is set.

Remarks

SMDFBitNIsSet is actually a set of macros where N stands for the bit to be tested. N ranges from 1 to 6 as shown below:

```
SMDFBit1IsSet Checks if bit 0 is set
SMDFBit6IsSet Checks if bit 6 is set
```

SMDFSetBitN

Sets bit N.

Syntax

```
#include <smsutapi.h>

void SMDFSetBitN (
    unsigned char   c);
```

Parameters

c

> (IN/OUT) Specifies the bit to set.

Remarks

Bits 6 and 7 are not affected by SMDFSetBitN.

SMDFSetBitN is actually a set of macros where N stands for the number of the bit to be set. N ranges from 1 to 6 as shown below:

```
SMDFSetBit1 Set bit 0
SMDFSetBit6 Set bit 5
```

SMDFSizeOfUINT32Data

Determines the number of bytes the data occupies.

Syntax

```
#include <smsutapi.h>

int SMDFSizeOfUINT32Data (
   UINT32   sizeOfData);
```

Parameters

sizeOfData

   (IN) Specifies the size of the data.

Return Values

If SMDFSizeOfUINT32Data is successful, it returns the size of the data.

SMDFSizeOfUINT32Data0

Determines the number of bytes the FID's data size descriptor occupies.

Syntax

```
#include <smsutapi.h>

int SMDFSizeOfUINT32Data0 (
   UINT32   fid);
```

Parameters

dataSize

(IN) Specifies the data's size.

Return Values

If SMDFSizeOfUINT32Data0 is successful, it returns the size of the data size descriptor.

SMDFSizeOfUINT64Data

Determines the number of bytes used by a UINT64 value.

Syntax

```
#include <smsutapi.h>

UINT32 SMDFSizeOfUINT64Data (
   UINT64   data);
```

Parameters

data

   (IN) Specifies the data to count.

Return Values

The following table lists the return values associated with the function.

0x00000000	The value of data is zero.
0x1	data has a value that uses only the lowest byte.
0z2	data has a value that uses only the lower 2 bytes.
0z4	data has a value that uses only the lower 4 bytes.
0z8	data has a value that uses all 8 bytes.

Remarks

The value returned by SMDFSizeOfUINT64Data is rounded to the next highest power of two. For example, if data uses five bytes, SMDFSizeOfUINT64Data returns 8.

SMDFZeroUINT64

Sets a UINT64 variable to zero.

Syntax

```
#include <smsutapi.h>

void SMDFZeroUINT64 (
   UINT64  *data);
```

Parameters

data

> (OUT) Specifies the variable to be set.

Utility Library Structures

<div style="text-align: right; font-size: 4em;">8</div>

This documentation alphabetically lists the Utility Library structures and describes their purpose, syntax, and fields.

ECMATime

Contains the date and time information that conforms to the ISO/IEC-13346 (section 7.3) standard for date and time.

Syntax

```
typedef struct
{
   UINT16   typeAndTimeZone;
   INT16    year;
   UINT8    month;
   UINT8    day;
   UINT8    hour;
   UINT8    minute;
   UINT8    second;
   UINT8    centiSecond;
   UINT8    hundredsOfMicroseconds;
   UINT8    microSeconds;
   UINT32   reserved;
} ECMATime;
```

Fields

typeAndTimeZone

Specifies the type of time zone and a time zone value (see "Time Zone Values" on page 349).

year

Specifies the year value (1-9999).

month

Specifies the month value (1-12).

day

Specifies the day value (1-31).

hour

Specifies the hour value (0-23).

minute

Specifies the minute value (0-59).

second

Specifies the second value (0-60 for a local type and 0-59 for other types).

centiSecond

Specifies the hundredths of second value (0-99).

hundredsOfMicroseconds

Specifies the hundreds of microsecond value (0-99).

microSeconds

　　Specifies the microsecond value (0-99).

NWSM_DATA_SET_NAME

Describes the path to a data set.

Syntax

```
typedef struct
{
   UINT32       nameSpaceType;
   UINT32       selectionType;
   UINT16       count;
   UINT16 HUGE  *namePositions;
   UINT16 HUGE  *separatorPositions;
   UINT16       nameLength;
   STRING       name;
} NWSM_DATA_SET_NAME;
```

Fields

nameSpacetype

> Specifies the name space type of name (see "nameSpaceType Values" on page 347).

selectionType

> Specifies how the data set is selected.

count

> Specifies the size of the namePositions and separatorPositions array.

namePositions

> Specifies part of an array that contains the beginning position of each path node in name.

separatorPositions

> Specifies part of an array that contains the index location of the beginning of the separator in name.

nameLength

> Specifies the size of name in bytes.

name

> Specifies the data set's name as it appears in the specified name space.

Remarks

NWSM_DATA_SET_NAME is used to copy from or append to list elements that are part of the NWSM_DATA_SET_NAME_LIST or NWSM_SELECTION_LIST structure and shows the name space, path node and separator positions, path length, and path.

You must set selectionType to zero if an element is appended to an NWSM_DATA_SET_NAME_LIST. If an element is being appended to an NWSM_SELECTION_LIST, it cannot be zero.

NWSM_EXTENSION_INFORMATION

Describes an extension information.

Syntax

```
typedef struct
{
   UINT32   extensionTag ;
   UINT32   tagVersion ;
   UINT32   extensionSize;
   void     info;
} NWSM_EXTENSION_INFORMATION;
```

Fields

extensionTag

> Specifies the extension tag value. See Section 9.5, "ExtensionTag Values," on page 350.

tagVersion

> Specifies the extension tag version. See Section 9.5, "ExtensionTag Values," on page 350.

extensionSize

> Contains the extension size in bytes as encountered in the encoded extension buffer.

info

> Points to the extension information. Encoded as per the extension tag and version structure definition.

Remarks

extensionSize points to the number of bytes of information that was used in the encoded extension buffer to represent the extension. This does not represent the size of the info field.

The info field can be type casted to the extensionTag structure whose version is equal to or lower than tagVersion field.

info field may contain a NULL if the extension was not encoded.

Extension tag structures that contain pointers fields may contain NULL if no information is encoded regarding the same.

NWSM_FIELD_TABLE_DATA

Syntax

```
typedef struct
{
   SMDF_FIELD_DATA   field;
   UINT32            sizeOfData;
   void             *addressOfData;
   UINT8             dataSizeMap;
   UINT8             reserved[3];
} NWSM_FIELD_TABLE_DATA;
```

Fields

field

Specifies the field's data.

sizeOfData

Specifies the number of bytes in fieldData (must be the same as fieldDataSize).

addressOfData

Specifies where the data was placed into buffer (optional).

dataSizeMap

Specifies if dataSizeMap or fieldDataSize contains the total size of the data to be put into field.

Remarks

Two fields that are not part of the section must be created to indicate that the first field of the section needs synchronization data, and the other field to signal the end of the section. These fields are the beginning field and the ending field. Only the fid value for these fields are specified. See SMDF_FIELD_DATA on the next page for more information.

addressOfData is valid if it is set to GET_ADRESS before you call SMDFPutFields. You can use this field to store data at a later time.

If the total amount of data to be put into field is less than 128 bytes, set dataSizeMap to the data's size. If the total amount of data is 128 bytes or more, set dataSizeMap to NWSM_VARIABLE_SIZE and field.dataSize to the total amount of data that the field will contain.

NWSM_GET_FIELDS_TABLE

Contains information about each table entry.

Syntax

```
typedef struct
{
   UINT32      fid;
   void       *data;
   UINT32      dataSize;
   NWBOOLEAN   found;
} NWSM_GET_FIELDS_TABLE;
```

Fields

fid

Specifies the FID of the field the you are looking for.

data

Specifies the field's data.

dataSize

Specifies the size of data in bytes.

found

Specifies a flag that indicates if a FID that matches fid was found (initialize to FALSE):

TRUE Match was found
FALSE No match was found

Remarks

Set fid to NWSM_RECORD_SIZE. The fid value of the first element must be set to the section FID. The fid value of the last element must be set to NWSM_END to indicate the end of the table.

data is allocated by the calling function. If the buffer is too small, BUFFER_OVERFLOW is returned.

NWSM_LIST

Contains list information.

Syntax

```
typedef struct NWSM_LIST_STRUCT
{
   UINT8                    marked;
   struct NWSM_LIST_STRUCT  *prev;
   struct NWSM_LIST_STRUCT  *next;
   void                     *otherInfo;
   BUFFER                    text[1];
} NWSM_LIST;
```

Fields

marked

  Specifies if the data is valid or has been touched.

prev

  Specifies the previous list element.

next

  Specifies the next list element.

otherInfo

  Specifies either a buffer containing user-defined information or an integer value (optional).

text

  Specifies further information about the element.

Remarks

Do not modify marked.

The size of text is determined by the amount of text to be stored.

SMS uses NWSM_LIST for scan types. text contains the scan type string, and otherInfo contains the scan type bit map.

NWSM_LIST_PTR

Specifies the beginning and ending list elements containing list information.

Syntax

```
typedef struct
{
   NWSM_LIST  *head;
   NWSM_LIST  *tail;
   int        (*sortProc)();
   void       (*freeProcedure)(void *memoryPointer);
} NWSM_LIST_PTR;
```

Fields

head

   Specifies the first element as returned by NWSMGetListHead.

tail

   Specifies the last element.

sortProc

   Specifies the function that compares the text portion of an element.

freeProcedure

   Specifies a function that frees the memory allocated to an element's otherInfo field.

Remarks

The following figure shows the relationship between NWSM_LIST_PTR and NWSM_LIST:
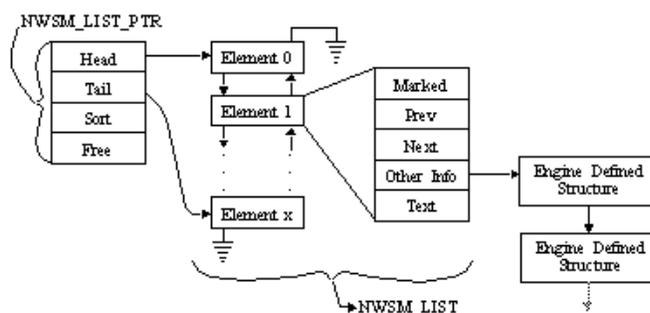
For example, suppose that a program needs to display a list of resources and resource information (free space, block size, etc.) when the user selects a resource name. The list functions can be used to put the resource name and resource information into the above structure to associate them as a unit.

Call NWSMGetListHead to get the list head, rather than traversing the list.

To get the list tail, find the list head and traverse the list until you find NULL.

The function specified by sortProc performs a case insensitive sort. sortProc is always set by the utilities library, but you can reset this pointer to another sort function. The prototype of sortProc is:

```
int sortProc(char *s1, char *s2);
```

The return values are:

```
>0   String s1 is greater than s2.
0    The strings match.
<0   String s1 is less than s2.
```

s1 and s2 are NULL-terminated strings.

Call NWSMInitList to initialize freeProcedure. The function pointed to by this field is called when you call NWSMDestroyList.

If otherInfo contains a non-allocated value such as an integer, freeProcedure must be set to NULL. It's prototype is:

```
void freeProcedure(void *ptr);
```

NWSM_MEDIA_INFO

Syntax

```
typedef struct
{
   ECMATIME    timeStamp;
   ECMATIME    setTimeStamp;
   BUFFER      label[NWSM_MAX_MEDIA_LABEL_LEN];
   UINT32      number;
} NWSM_MEDIA_INFO;
```

Fields

timeStamp

Specifies the date and time the media header was written in ECMA date and time standards.

setTimeStamp

Specifies the date and time the media set was created.

label

Specifies the name of the media set (NWSM_MEDIA_LABEL_LEN includes the NULL terminator).

number

Specifies the position of the medium within the media set (beginning with 1).

Remarks

If a media set contains three media, and the back up session started on July 3, 1991 at 11:30 p.m., the time stamp for each medium would indicate the following as each media header is written:

```
medium 1   July 3, 1991 11:300 p.m.
medium 2   July 4, 1991 12:00 a.m.
medium 3   July 4, 1991, 12:30 a.m.
```

SMS DI users should initialize the first timeStamp value to setTimeStamp. SMS DI will automatically update each time stamp value for subsequent media.

setTimeStamp is the same for all media in the media set.

NWSM_MODULE_VERSION_INFO

Contains information about the module.

Syntax

```
typedef struct
{
   char     moduleFileName[256];
   UINT8    moduleMajorVersion,
   UINT8    moduleMinorVersion;
   UINT16   moduleRevisionLevel;
   char     baseOS[64];
   UINT8    baseOSMajorVersion,
   UINT8    baseOSMinorVersion;
   UINT16   baseOSRevisionLevel;
} NWSM_MODULE_VERSION_INFO;
```

Fields

moduleFileName

Specifies the file name of the module.

moduleMajorVersion

Specifies the whole number portion of the version number.

moduleMinorVersion

Specifies the decimal portion of the version number.

moduleRevisionLevel

Specifies the letter portion of the version number.

baseOS

Specifies the OS the module is running on.

baseOSMajorVersion

Specifies the whole number portion of the OS version number.

baseOSMinorVersion

Specifies the decimal portion of the OS version number.

baseOSRevisionLevel

Specifies the letter portion of the OS version number.

Remarks

If the version number is 1.3a, moduleMajorVersion, moduleMinorVersion, and moduleRevisionLevel return "1," "3," and "a" respectively.

NWSM_RECORD_HEADER_INFO

Syntax

```
typedef struct
{
   NWBOOLEAN                isSubRecord;
   UINT32                   headerSize;
   UINT32                   recordSize;
   NWSM_DATA_SET_NAME_LIST  *dataSetName;
   NWSM_SCAN_INFORMATION    *scanInformation;
   ECMATime                 archiveDateAndTime;
   UINT32                   *addressOfRecordSize;
   UINT32                   *addressForCRC;
   BUFFERPTR                crcBegin;
   UINT32                   crcLength;
   UINT32                   dataSetInfoRetrieved;
   UINT32                   saveBufferSize;
   BUFFERPTR                saveBuffer;
} NWSM_RECORD_HEADER_INFO;
```

Fields

headerSize

Specifies the total size of the data set header or subheader.

recordSize

Specifies the size of the data in the record (the data set information and the portion of the data set's data in the record).

isSubRecord

Specifies if the record is a subrecord:

TRUE NWSMUT_BUFFER_OVERFLOW was returned from NWSMSetNewRecordHeader
FALSE Need a header for a new data set

dataSetName

Specifies the data set name space type as returned by NWSMTSScanDataSetBegin or NWSMTSScanNextDataSet.

scanInformation

Specifies the data set's scan information as returned by NWSMTSScanDataSetBegin or NWSMTSScanNextDataSet.

archiveDateAndTime

Specifies the date and time the data set was added to the buffer.

addressOfRecordSize

Specifies the record size value (UINT32) in the SIDF data set header or subheader.

addressForCRC

Specifies the address for the data set header CRC value:

crcBegin

Specifies where the CRC will be applied.

crcLength

Specifies the number of bytes the CRC was applied to (not valid until NWSMUpdateRecordHeader is called).

dataSetInfoRetrieved

Is used internally by NWSMGetDataSetInfo and NWSMGetRecordHeaderOnly.

saveBufferSize

Is used internally by NWSMGetDataSetInfo and NWSMGetRecordHeaderOnly.

saveBuffer

Is used internally by NWSMGetDataSetInfo and NWSMGetRecordHeaderOnly.

Remarks

recordSize is the size of the record minus the amount of data put into scanInformation and dataSetName. It is decremented to reflect the amount of data transferred from the buffer to the recordHeaderInfo parameter.

When the data set data is added to the record, you must update recordSize to reflect the record's new size.

isSubRecord remains TRUE until all of a data set's data is written to the buffer.

The time specified by archiveDateAndTime may not be the same as scanInformation.archiveDateAndTime.

addressForCRC is usually used by NWSMUpdateRecordHeader to update the CRC value after the data is inserted into the transfer buffer.

The CRC for the data set information is calculated and placed into the section automatically if setCRC is CRC_YES or CRC_LATER and no pointer is returned for addressForCRC.

crcBegin is usually used by NWSMUpdateRecordHeader to update the CRC value after the data is inserted into the transfer buffer.

The values that can be set for dataSetInfoRetrieved is listed in the following table.

Value	Description
0x00000000	DATA_SET_INFO_NOT_STARTED: The data set information has not been processed yet as set by NWSMGetRecordHeaderOnly. The retrieval of the data from the new record has not begun. Call NWSMGetDataSetinfo to retrieve this data from the transfer buffer.
0x00000001	DATA_SET_INFO_SPANNED: The data set name list or scan information spans the buffer. Call NWSMGetDataSetInfo with the same recordHeaderInfo.

Value	Description
0x00000002	DATA_SET_INFO_COMPLETE: dataSetName and scanInformation specify complete structures. Until dataSetInfoRetrieved returns this value, dataSetName and scanInformation cannot be used.
0x00000003	DATA_SET_INFO_DOES_NOT_EXIST: The data set information starts in the next transfer buffer. In a backup session, if the data set information cannot fit into the space left in the transfer buffer, the data set information is placed into the next transfer buffer, preceded by a subrecord header as set by NWSMGetRecordHeaderOnly. The retrieval of the data has begun. Call NWSMGetDataSetInfo to get the next portion of the data from the transfer buffer.

Set dataSetName and scanInformation to NULL or to a valid structure. The memory used by the data set name or scan information will be resized if there is not enough space.

The following fields in recordHeaderInfo are not used by NWSMGetRecordHeaderOnly:

```
dataSetName
scanInformation
addressOfRecordSize
addressForCRC
crcBegin
crcLength
saveBufferSize
saveBuffer
```

NWSM_RESOURCE_INFO_EXTN_NETWARE_DATA_1

Describes the NetWare file systems primary resource meta data.

Syntax

```
typedef struct
{
   UINT16     blockSize;
   UINT32     totalBlocks;
   UINT32     freeBlocks;
   NWBOOLEAN  resourceIsRemovable;
   UINT32     purgableBlocks;
   UINT32     notYetPurgableBlocks;
   UINT32     migratedSectors;
   UINT32     preCompressedSectors;
   UINT32     compressedSectors;
} NWSM_RESOURCE_INFO_EXTN_NETWARE_DATA_1;
```

Fields

blockSize

> Contains the resource's disk block size.

totalBlocks

> Contains the total number of blocks on the resource.

freeBlocks

> Contains the number of free blocks on the resource.

resourceIsRemovable

> Contains a flag indicating whether the resource is removable.
>
> TRUE Resource is removable.
>
> FALSE Resource is not removable.

purgableBlocks

> Contains the total number of blocks set aside as purgeable blocks.

notYetPurgeableBlocks

> Contains the number of blocks not marked to be purged.

migratedSectors

> Contains the number of migrated sectors.

precompressedSectors

> Contains the number of sectors used by all data sets before they were compressed.

compressedSectors

Contains the number of sectors used by all compressed data sets.

NWSM_RESOURCE_INFO_EXTN_UNIX_DATA_1

Describes the Linux file system primary resource meta data.

Syntax

```
typedef struct
{
   char      *mnt_fsname;
   char      *mnt_type;
   char      *mnt_opts;
} NWSM_RESOURCE_INFO_EXTN_UNIX_DATA_1;
```

Fields

mnt_fsname

   Name of the mounted file system.

mnt_type

   Type of mount.

mnt_opts

   Mount options used to mount the file system.

Remarks

See SLES man pages getmntent(3), mount(8), and mntent.h header for various values of these fields.

NWSM_RESOURCE_INFO_EXTN_UNSUPPORTE D_DATA_1

Describes the unsupported TSA options.

Syntax

```
typedef struct
{  UINT32     unsupportedBackupOptions;
   UINT32     unsupportedRestoreOptions;
} NWSM_RESOURCE_INFO_EXTN_UNSUPPORTED_DATA_1;
```

Fields

unsupportedBackupOptions

Contains a bit map that represents the TSA's unsupported backup options.

unsupportedRestoreOptions

Contains a bit map that represents the TSA's unsupported restore options.

Remarks

See NWSMTSGetUnsupportedOptions for a list of unsupported backup and restore options.

NWSM_SCAN_INFO_EXTN_NFS_DATA_1

Describes the NFS file system meta data.

Syntax

```
typedef struct
{
   UINT32   nfs_st_mode;
   UINT32   nfs_st_nlinks;
   UINT32   nfs_st_nlinks;
   UINT32   nfs_st_gid;
   UINT32   nfs_st_ctime;
} NWSM_SCAN_INFO_EXTN_NFS_DATA_1;
```

Fields

nfs_st_mode

> Contains the file permission mode as defined by stat.h in the IEEE Std 1003.1.

nfs_st_nlinks

> Contains the number of links to the data set as defined by stat.h in the IEEE Std 1003.1

nfs_st_uid

> Contains the user ID of the data set as defined by stat.h in the IEEE Std 1003.1

nfs_st_gid

> Contains the group ID of the data set as defined by stat.h in the IEEE Std 1003.1

nfs_st_ctime

> Contains the time of last file status change as defined by stat.h in the IEEE Std 1003.1

NWSM_SESSION_INFO

Syntax

```
typedef struct
{
   ECMATime    timeStamp;
   UINT32      sessionID;
   BUFFER      description;
   BUFFER      softwareName;
   BUFFER      softwareType;
   BUFFER      softwareVersion;
   BUFFER      sourceName;
   BUFFER      sourceType;
   BUFFER      sourceVersion;
   BUFFER      sidfSourceNameType;
   BUFFER      sidfSourceName;
} NWSM_SESSION_INFO;
```

Fields

timeStamp

Specifies the date and time the session was created.

sessionID

Specifies the unique ID that identifies the session.

description

Specifies a user-defined string including a NULL terminator (maximum of NWSM_MAX_DESCRIPTION_LEN).

softwareName

Specifies the name of the software servicing the target (maximum of NWSM_MAX_SOFTWARE_NAME_LEN).

softwareType

Specifies the type of the software doing the backup (maximum of NWSM_MAX_SOFTWARE_TYPE_LEN).

softwareVersion

Specifies the software version string (maximum of NWSM_MAX_SOFTWARE_VER_LEN).

sourceName

Specifies the target service's name as returned by NWSMTSGetTargetServiceType (page 71) (maximum of NWSM_MAX_TARGET_SRVC_NAME_LEN).

sourceType

Specifies the target's type string as returned by NWSMTSGetTargetServiceType (page 71) (maximum of NWSM_MAX_TARGET_SRVC_TYPE_LEN).

sourceVersion

Specifies the target's version string as returned by NWSMTSGetTargetServiceType (page 71) (maximum of NWSM_MAX_TARGET_SRVC_VER_LEN).

sidfSourceNameType

Specifies the type of string contained in sidfSourceName (maximum of NWSM_MAX_SIDF_SRC_NAME):

SMS SMS string
common Non-SMS string

sidfSourceName

Specifies the name of the target (maximum of NWSM_MAX_SIDF_SRC_NM_TYPE_LEN).

Remarks

The two kinds of format used by sidfSourceName is listed in the following table.

Format	Description
SMS	SMDR_name.TSA_name.Target_Service_Name
common	String can contain anything

To get the SMDR name, refer to NWSMListSMDRs (page 301).

SMDF_FIELD_DATA

Syntax

```
typedef struct
{
   UINT32  fid;
   UINT64  dataSize;
   void    *data;
   UINT32  bytesTransfered;
   UINT64  dataOverflow;
} SMDF_FIELD_DATA;
```

Fields

fid

   Specifies a FID value.

dataSize

   Specifies the size of the data in bytes.

data

   Specifies all of the field's data.

bytesTransferred

   Specifies the number of bytes moved into buffer.

dataOverflow

   Specifies the number of bytes that could not be transferred into buffer.

Remarks

The FID value of the first table field must be NWSM_BEGIN. The FID value of the field that marks the end of the section must be NWSM_END.

dataSize is used only for data sizes over and including 128 bytes (size format 2) and is invalid if there are less than 128 bytes of data. For data sizes under 128 bytes, see dataSizeMap of NWSM_FIELD_TABLE_DATA (page 318).

bytesTransferred does not need to be initialized to any value.

UINT64

Contains an 8-byte unsigned integer value.

Syntax

```
typedef struct
{
    UINT16   v[4];
} UINT64;
```

Return Values

9

This chapter lists the return values used with Target Services and Utility Library.

9.1 Target Services Values

This section lists the values that Target Services can return.

9.2 Target Services Generic Open Mode Values

The two types of generic open mode values are:

9.2.1 Generic Backup Open Mode Values

The generic backup open mode strings are built by the engine from the values given in the following table.

Value	Modes	Description
Numeric Modes:		
0x0001	NWSM_USE_LOCK_MODE_IF_DW_FAILS	Open the data set using lock mode if a deny write open mode fails.
0x0002	NWSM_NO_LOCK_NO_PROTECTION	Circumvents lock and write protection. The engine should only use this mode when data integrity is not required.
0x0003	NWSM_OPEN_READ_ONLY	The dataset is opened in Read only mode and the data is of the last saved state. This mode circumvents lock and write protection and the engine should only use it when data integrity is not required.

9.2.2 Generic Restore Open Mode Values

The generic restore open mode strings are built by the engine from the values given in the following table.

Value	Modes	Description
Numeric Modes:		
0x0000	Invalid	Invalid
0x0001	NWSM_OVERWRITE_DATA_SET	Open the data set and allow it to be overwritten.
0x0002	NWSM_DO_NOT_OVERWRITE_DATA_SET	Do not open the data set if it exists on the Target Service. NWSM_DO_NOT_OVERWRITE_DATA_SET has lower precedence than excluding or including a data set. That is, if a data set is marked as do not overwrite and is also marked as included in the selection list, the data set will be overwritten. To avoid overwriting an included data set, it must be excluded.
0x0003	NWSM_CREATE_PARENT_HANDLE	Create a parent handle. If the parent does not exist on the Target Service, it is created. If the parent exists, it is not overwritten.
0x0004	NWSM_UPDATE_DATA_SET	If the data set on the media is newer than the Target Service's copy, update the Target Service's copy. This mode is applicable only for files.
Bitmap Modes:		
0x0040	NWSM_CLEAR_MODIFY_FLAG_RESTORE	Clear the modify flag after the data set is restored.
0x0080	NWSM_RESTORE_MODIFY_FLAG	Set the data set's modify flag to what it was before the backup session began.

9.2.3 TSA-Specific Open Mode Values

For the NetWare TSAs, the returned strings represent the following TSA-specific open modes values:

The following table lists the TSA-specific open modes values.

Value	Modes	Description
0x0100	NWSM_NO_DATA_STREAMS	Do not read or write the data set's data streams
0x0200	NWSM_NO_EXTENDED_ATTRIBUTES	Do not read or write the data set's extended attribute
0x0400	NWSM_NO_PARENT_TRUSTEES	If the data set is a parent, do not read or write its trustee information

Value	Modes	Description
0x0800	NWSM_NO_CHILD_TRUSTEES	If the data set is a child, do not read or write its trustee information.
0x1000	NWSM_NO_VOLUME_RESTRICTIONS	Do not read or write the data set's resource restriction information.
0x2000	NWSM_NO_DISK_SPACE_RESTRICTIONS	Do not read or write the data set's space restrictions.
0x8000	NWSM_DELETE_EXISTING_TRUSTEES	Delete the existing trustees during a restore from a data set before restoring the backed up trustees.
0x10000	NWSM_EXPAND_COMPRESSED_DATA_SET	Expand data sets that are currently compressed on the host. This option is used only during backup.
0x20000	NWSM_EXCLUDE_MIGRATED_DATA	Do not backup migrated data sets.
0x4000	NWSM_INCLUDE_MIGRATED_DATA	Restore migrated streams for a data set on the primary storage medium. The default behavior is to skip migrated streams during a restore.
0x40000	NWSM_PRESERVE_ACCESS_TIME	Preserves the access time of the data set after a backup session.
0x80000	NWSM_NO_HARDLINK_DATA	Do not backup the data of consecutive hard link nodes of a particular hardlink network, except for the first encountered node.

9.3 Target Service Return Values

This section lists the return values returned by the TSAPI and SMDR, and TSANDS return values that directly map to the NDS return values, see .

9.3.1 TSAPI and SMDR Return Values

The following table lists completion codes for the TS API and SMDR. (completion codes beginning with 0xFFFD, 0xFFFE, and 0xFFFF).

Value	Error String	Description
0xFFFDFFAD	NWSMTS_UNSUPPORTED_OPTION	One of the selected options is not supported by the target service
0xFFFDFFAE	NWSMTS_CLUSTER_TARGET_HAS_NO_VOLUMES	The cluster pool does not contain any resources
0xFFFDFFAF	NWSMTS_CLUSTER_TARGET_DOES_NOT_EXIST	Cluster target does not exist
0xFFFDFFB0	NWSMTS_INVALID_MESSAGE_NUMBER	Message number is invalid

Value	Error String	Description
0xFFFDFFB1	NWSMTS_INTERNAL_ERROR	An internal TSA error occurred, see the error log for more details
0xFFFDFFB2	NWSMTS_COMPRESSION_CONFLICT	Attempted to put compressed data on a noncompressed resource
0xFFFDFFB3	NWSMTS_MAX_CONNECTIONS	All available connections to the TSA are in use
0xFFFDFFB4	NWSMTS_REDIRECT_TRANSPORT	Indicates reconnection requirement
0xFFFDFFB5	NWSMTS_WRITE_ERROR	An error occurred while writing to a file
0xFFFDFFB6	NWSMTS_WRITE_ERROR_SHORT	An error occurred while writing to a file. Could not write all of the data of current request
0xFFFDFFB7	NWSMTS_WRITE_EA_ERR	Unable to write the extended attribute information
0xFFFDFFB8	NWSMTS_VALID_PARENT_HANDLE	A valid parent handle was created
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION	The requested function is not supported by this TSA
0xFFFDFFBA	NWSMTS_TSA_NOT_FOUND	Invalid or inactive TSA specified
0xFFFDFFBB	NWSMTS_TRANSPORT_PACKET_SIZE_ER	The read/write request exceeds 128K
0xFFFDFFBC	NWSMTS_TRANSPORT_FAILURE	The transport mechanism failed
0xFFFDFFBD	NWSMTS_SET_FILE_INFO_ERR	Unable to set file information
0xFFFDFFBE	NWSMTS_SELECTION_TYPE_NOT_USED	Selection type is not used
0xFFFDFFBF	NWSMTS_SCAN_TYPE_NOT_USED	Scan type is not used
0xFFFDFFC0	NWSMTS_SCAN_TRUSTEE_ERR	Unable to scan for the trustees information. Running DSRepair may resolve the issue
0xFFFDFFC1	NWSMTS_SCAN_NAME_SPACE_ERR	Unable to scan name-space specific information
0xFFFDFFC2	NWSMTS_SCAN_IN_PROGRESS	Cannot alter the resource list while a scans is in progress
0xFFFDFFC4	NWSMTS_SCAN_ERROR	The scan failed; the probable cause is an invalid path
0xFFFDFFC3	NWSMTS_SCAN_FILE_ENTRY_ERR	Unable to scan file entry information
0xFFFDFFC5	NWSMTS_RESOURCE_NAME_NOT_FOUND	No resource name is found or all resource names have been found
0xFFFDFFC6	NWSMTS_READ_ERROR	Cannot read the file
0xFFFDFFC7	NWSMTS_READ_EA_ERR	Unable to read the extended attributes
0xFFFDFFC8	NWSMTS_OVERFLOW	A UINT64 value overflowed

Value	Error String	Description
0xFFFDFFC9	NWSMTS_OUT_OF_MEMORY	The file server is out of memory or the memory allocation failed
0xFFFDFFCA	NWSMTS_OUT_OF_DISK_SPACE	Cannot restore the data, because the target service is out of disk space
0xFFFDFFCB	NWSMTS_OPEN_MODE_TYPE_NOT_USED	The open mode option is not used
0xFFFDFFCC	NWSMTS_OPEN_ERROR	Cannot open a file
0xFFFDFFCD	NWSMTS_OPEN_DATA_STREAM_ERR	The possible instances when TSA500 can encounter this error are: Returning the meta data information for the file object. Returning the directory base information for a specified file path. Opening a file. In either case, the engine should continue with the next data set that it can backup. The possible instance when TSA600 can encounter this error is from the file system open call while opening a file system object. In this case, the engine should continue with the next data set that it can backup.
0xFFFDFFCE	NWSMTS_NO_SUCH_PROPERTY	No such property.
0xFFFDFFCF	NWSMTS_NO_SEARCH_PRIVILEGES	No search privilege on the client service
0xFFFDFFD0	NWSMTS_NO_MORE_NAMES	No more entries in the list or name space does not exist
0xFFFDFFD1	NWSMTS_NO_MORE_DATA_SETS	There are no more data sets to scan
0xFFFDFFD2	NWSMTS_NO_MORE_DATA	No more data exists
0xFFFDFFD3	NWSMTS_NO_CONNECTION	The specified connection is invalid or does not exist
0xFFFDFFD4	NWSMTS_NOT_READY	The specified server is unable to service the request at this time
0xFFFDFFD5	NWSMTS_NAME_SP_PATH_NOT_UPDATED	The name space path has not been updated
0xFFFDFFD6	NWSMTS_LOGOUT_ERROR	Unable to logout
0xFFFDFFD7	NWSMTS_LOGIN_DENIED	Login denied
0xFFFDFFD8	NWSMTS_INVALID_SEQUENCE_NUMBER	The sequence number is invalid
0xFFFDFFD9	NWSMTS_INVALID_SELECTION_TYPE	Invalid selection type. The selection type was either less than zero or greater than thirty one

Value	Error String	Description
0xFFFDFFDA	NWSMTS_INVALID_SEL_LIST_ENTRY	An invalid selection list entry was passed
0xFFFDFFDB	NWSMTS_INVALID_SCAN_TYPE	Invalid scan type was used. The scan type was either less than zero or greater than thirty one
0xFFFDFFDC	NWSMTS_INVALID_PATH	An invalid path was used
0xFFFDFFDD	NWSMTS_INVALID_PARAMETER	One or more of the parameters are NULL or invalid
0xFFFDFFDE	NWSMTS_INVALID_OPEN_MODE_TYPE	Invalid open mode option type. The option type is less than zero or greater than 23
0xFFFDFFDF	NWSMTS_INVALID_OBJECT_ID	The object's backed up id and name does not match the current object's id and name
0xFFFDFFE0	NWSMTS_INVALID_NAME_SPACE_TYPE	The name space type does not exist or is invalid
0xFFFDFFE1	NWSMTS_INVALID_MESSAGE_NUMBER	The message number is invalid
0xFFFDFFE2	NWSMTS_INVALID_HANDLE	The handle is tagged invalid or is set to zero
0xFFFDFFE3	NWSMTS_INVALID_DATA_SET_TYPE	Data set type is invalid
0xFFFDFFE4	NWSMTS_INVALID_DATA_SET_NAME	The data set name is invalid
0xFFFDFFE5	NWSMTS_INVALID_DATA_SET_HANDLE	The data set handle is invalid
0xFFFDFFE6	NWSMTS_INVALID_DATA	The data set is invalid
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL	Invalid connection handle was passed
0xFFFDFFE8	NWSMTS_GET_VOL_NAME_SPACE_ERR	Unable to get the name space information that is supported by the resource
0xFFFDFFE9	NWSMTS_GET_SERVER_INFO_ERR	Unable to get the file server's information
0xFFFDFFEA	NWSMTS_GET_NAME_SPACE_SIZE_ERR	Unable to get name space size information
0xFFFDFFEB	NWSMTS_GET_NAME_SPACE_ENTRY_ERR	Unable to get the name space entry name
0xFFFDFFEC	NWSMTS_GET_ENTRY_INDEX_ERR	Unable to get the entry index
0xFFFDFFED	NWSMTS_GET_DATA_STREAM_NAME_ERR	Unable to get the data stream's name
0xFFFDFFEE	NWSMTS_GET_BIND_OBJ_NAME_ERR	Unable to get the name of a bindery object

Value	Error String	Description
0xFFFDFFEF	NWSMTS_EXPECTING_TRAILER	Received a Record or Subrecord Trailer, but could not locate the trailer field
0xFFFDFFF0	NWSMTS_EXPECTING_HEADER	Received a Record or Subrecord Header, but could not locate the header field
0xFFFDFFF1	NWSMTS_DELETE_ERR	Cannot delete a data set
0xFFFDFFF2	NWSMTS_DATA_SET_NOT_FOUND	No data set was found or the resource is not available.
0xFFFDFFF3	NWSMTS_DATA_SET_IS_OPEN	Attempted to open an already data set or attempted to alter a scan while a data set is open
0xFFFDFFF4	NWSMTS_DATA_SET_IS_OLDER	The existing data set on the target data set is newer the one on the media, the data set will not be restored
0xFFFDFFF5	NWSMTS_DATA_SET_IN_USE	Data set is currently in use and cannot be accessed
0xFFFDFFF6	NWSMTS_DATA_SET_EXECUTE_ONLY	Can only execute the file
0xFFFDFFF7	NWSMTS_DATA_SET_EXCLUDED	The data set is excluded by the selection list
0xFFFDFFF8	NWSMTS_DATA_SET_ALREADY_EXISTS	The data set name already exists
0xFFFDFFF9	NWSMTS_CREATE_ERROR	Cannot create a file
0xFFFDFFFA	NWSMTS_CREATE_DIR_ENTRY_ERR	Cannot create directory entry
0xFFFDFFFB	NWSMTS_CLOSE_BINDERY_ERROR	Cannot close the bindery
0xFFFDFFFC	NWSMTS_CANT_ALLOC_DIR_HANDLE	Cannot allocate a directory handle
0xFFFDFFFD	NWSMTS_BUFFER_UNDERFLOW	Buffer underflow, unable to get entire field
0xFFFDFFFE	NWSMTS_BINDERY_OBJECT_NAME_ERR	Unable to get bindery object name
0xFFFDFFFF	NWSMTS_ACCESS_DENIED	Invalid user name or authentication
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION	Invalid connection handle was passed to the SMDR
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER	One or more of the parameters is NULL or invalid
0xFFFEFFFD	NWSMDR_OUT_OF_MEMORY	SMDR memory allocation failed
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE	The transport mechanism has failed
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION	The requested function is not supported by SMDR
0xFFFEFFFA	NWSMDR_MODULE_ALREADY_EXPORTED	The module is already exported by SMDR

Value	Error String	Description
0xFFFEFFF9	NWSMDR_DECRYPTION_FAILURE	The decryption mechanism of the SMDR failed
0xFFFEFFF8	NWSMDR_ENCRYPTION_FAILURE	The encryption mechanism of the SMDR failed
0xFFFEFFF7	NWSMDR_TSA_NOT_LOADED	The requested TSA is not loaded
0xFFFEFFF6	NWSMDR_NO_SUCH_SMDR	The specified SMDR does not exist
0xFFFEFFF5	NWSMDR_SMDR_CONNECT_FAILURE	SMDR connection failure. Load the SMDR on the remote server and try again
0xFFFEFFF4	NWSMDR_NO_MORE_DATA	No more data to process
0xFFFEFFF3	NWSMDR_NO_SOCKETS	No more sockets available
0xFFFEFFF2	NWSMDR_INVALID_PROTOCOL	The specified protocol does not exist
0xFFFEFFF1	NWSMDR_NO_MORE_CONNECTIONS	Unable to create another connection handle. Maximum number of connections is reached. Try to close some existing connections
0xFFFEFFF0	NWSMDR_NO_SUCH_TSA	Requested TSA does not exist
0xFFFEFFEF	NWSMDR_INVALID_MESSAGE_NUMBER	Invalid message number
0xFFFEFFEE	NWSMDR_OUT_OF_SHMEM	SMDR out of shared memory
0xFFFEFFED	NWSMDR_INVALID_PATHNAME	Invalid path name
0xFFFEFFEC	NWSMDR_INVALID_BUFFER_SIZE	Invalid buffer size
0xFFFEFFEB	NWSMDR_INVALID_ADDRESS	Invalid address
0xFFFEFFEA	NWSMDR_INVALID_HANDLE	Invalid Handle passed to SMDR
0xFFFEFFE9	NWSMDR_INVALID_CONTEXT	The specified context is invalid
0xFFFEFFE8	NWSMDR_INVALID_NAME	The specified name does not exist or is invalid
0xFFFEFFE7	NWSMDR_INVALID_INSTANCE	Invalid SMDR instance
0xFFFEFFE6	NWSMDR_INVALID_TARGET	Invalid target
0xFFFEFFE5	NWSMDR_INVALID_SERVICE	Requested RPC call is invalid
0xFFFEFFE4	NWSMDR_INVALID_STREAM	Invalid transport stream
0xFFFEFFE3	NWSMDR_INVALID_TYPE	Invalid Registry type
0xFFFEFFE2	NWSMDR_INVALID_SYNTAX	Invalid syntax
0xFFFEFFE1	NWSMDR_NAME_TOO_LONG	Specified name is too long
0xFFFEFFE0	NWSMDR_CORRUPTED_STATE	The TSA information is corrupted
0xFFFEFFDF	NWSMDR_UNKNOWN_ADDRESS	The specified address does not exist
0xFFFEFFDE	NWSMDR_UNKNOWN_SERVICE_TYPE	Unknown RPC Service Type

Value	Error String	Description
0xFFFEFFDD	NWSMDR_UNKNOWN_ERROR	Unknown error occurred in SMDR
0xFFFEFFDC	NWSMDR_UNKNOWN_COMMAND	Unknown command for listener
0xFFFEFFDB	NWSMDR_UNKNOWN_ENCRYPTION	Unknown encryption mechanism
0xFFFEFFDA	NWSMDR_REGISTRY_INVALID	Invalid registry
0xFFFEFFD9	NWSMDR_REGISTRY_EMPTY	Registry is empty
0xFFFEFFD8	NWSMDR_REGISTRY_FULL	Registry is full
0xFFFEFFD7	NWSMDR_RESOURCE_LOCKED	Resource is locked by another thread
0xFFFEFFD6	NWSMDR_INSTANCE_BUSY	Service instance is busy
0xFFFEFFD5	NWSMDR_SHUTDOWN_FAILURE	SMDR shutdown failed
0xFFFEFFD4	NWSMDR_TIMEOUT	SMDR timed out
0xFFFEFFD3	NWSMDR_NO_PROTOCOLS	No protocols specified
0xFFFEFFD2	NWSMDR_NO_SERVICES	No services available
0xFFFEFFD1	NWSMDR_NO_DSAPI	DS API interface not initialized
0xFFFEFFD0	NWSMDR_FILE_NOT_FOUND	The specified file was not found
0xFFFEFFCF	NWSMDR_CREATE_FAILURE	Failed to create Object Instance
0xFFFEFFCE	NWSMDR_OPEN_FAILURE	Unable to open the configuration file for reading
0xFFFEFFCD	NWSMDR_READ_FAILURE	Unable to read from source
0xFFFEFFCC	NWSMDR_WRITE_FAILURE	Unable to write to the destination
0xFFFEFFCB	NWSMDR_SEEK_FAILURE	Unable to seek the position
0xFFFEFFCA	NWSMDR_CLOSE_FAILURE	Unable to close the resource
0xFFFEFFC9	NWSMDR_ACCESS_DENIED	Access to resource denied
0xFFFEFFC8	NWSMDR_DELETE_FAILURE	Unable to delete
0xFFFEFFC7	NWSMDR_BAD_CONFIGURATION	Invalid SMDR configuration
0xFFFEFFC6	NWSMDR_BAD_FILE_HANDLE	Invalid file handle passed
0xFFFEFFC5	NWSMDR_BAD_COMMAND_LINE	Invalid command line
0xFFFEFFC4	NWSMDR_BAD_OPTION	Invalid option passed
0xFFFEFFC3	NWSMDR_BAD_OPTION_VALUE	Invalid option value passed
0xFFFEFFC2	NWSMDR_NO_MECHANISM	No mechanism passed
0xFFFEFFC1	NWSMDR_HELP_OPTION	No Help option
0xFFFEFFC0	NWSMDR_BUFFER_OVERFLOW	Buffer overflow occurred
0xFFFEFFBF	NWSMDR_BUFFER_UNDERFLOW	Buffer underflow occurred
0xFFFEFFBE	NWSMDR_BUFFER_LOCKED	Buffer is locked

Value	Error String	Description
0xFFFEFFBD	NWSMDR_ENTRY_NOT_FOUND	The specified entry was not found
0xFFFEFFBC	NWSMDR_ENTRY_EXISTS	Duplicate entry exists
0xFFFEFFBB	NWSMDR_REMOVING_ENTRY	Removing the entry
0xFFFEFFBA	NWSMDR_TABLE_OVERFLOW	Table overflow occurred
0xFFFEFFB9	NWSMDR_INVALID_INDEX	Invalid index
0xFFFEFFB8	NWSMDR_PROTOCOL_NOT_FOUND	The specified protocol was not found
0xFFFEFFB7	NWSMDR_SVCTYPE_NOT_FOUND	Unknown service type
0xFFFEFFB6	NWSMDR_SERVICE_NOT_FOUND	Unknown service
0xFFFEFFB5	NWSMDR_NOT_AUTHENTICATED	Authentication necessary to access resource
0xFFFEFFB4	NWSMDR_INSUFFICIENT_RIGHTS	Insufficient rights to access resource
0xFFFEFFB3	NWSMDR_UNKNOWN_HOST	Unknown host specified
0xFFFEFFB2	NWSMDR_CONNECT_FAILURE	SMDR connection failed
0xFFFEFFB1	NWSMDR_DISCONNECTED	Resource disconnected
0xFFFEFFB0	NWSMDR_DISCONNECT_FAILURE	Failed to disconnect resource
0xFFFEFFAF	NWSMDR_LISTENER_FAILURE	Socket Listener failed
0xFFFEFFAE	NWSMDR_ACCEPT_FAILURE	Socket could not accept connection
0xFFFEFFAD	NWSMDR_TSP_BIND_FAILURE	Cannot bind to the socket
0xFFFEFFAC	NWSMDR_POLL_FAILURE	Unable to poll
0xFFFEFFAB	NWSMDR_LOGIN_FAILURE	Unable to login
0xFFFEFFAA	NWSMDR_AUTHENTICATION_FAILURE	Unable to authenticate
0xFFFEFFA8	NWSMDR_SPAWN_FAILURE	Unable to spawn child thread
0xFFFEFFA7	NWSMDR_MODULE_LOAD_FAILURE	Dynamic loading of module failed
0xFFFEFFA6	NWSMDR_MODULE_UNLOAD_FAILURE	Dynamic unloading of module failed
0xFFFEFFA5	NWSMDR_DYNAMIC_BIND_FAILURE	Dynamic function binding failed
0xFFFEFFA4	NWSMDR_DYNAMIC_UNBIND_FAILURE	Dynamic function unbinding failed
0xFFFEFFA3	NWSMDR_INVALID_DYNAMIC_SYMBOL	Dynamic function is invalid
0xFFFEFFA2	NWSMDR_INVALID_CONSTRUCTOR	Invalid constructor
0xFFFEFFA1	NWSMDR_NO_DATA_FOUND	Unable to find data
0xFFFEFFA0	NWSMDR_TARGET_NOT_REGISTERED	Specified target not registered
0xFFFEFF9F	NWSMDR_SAP_FAILED	SAP mechanism failed

9.3.2 TSANDS Return Values

The TSANDS return values directly map to the NDS return values.

The following table lists the TSANDS return values.

TSANDS Hexadecimal Number	Constant	Maps to...
FFFDFEFF	NWSMTS_INSUFFICIENT_MEMORY	0xFFFF FF6A (http://developer.novell.com/ndk/doc/ndslib/dsov_enu/data/hcvwzt90.html) > NDS Return Values from the Operating System
FFFDFEFE	NWSMTS_REQUEST_UNKNOWN	0xFFFF FF05 (http://developer.novell.com/ndk/doc/ndslib/dsov_enu/data/hcvwzt90.html) > NDS Return Values from the Operating System
FFFDFEFD	NWSMTS_OF_SOME_SORT	0xFFFF FF01 (http://developer.novell.com/ndk/doc/ndslib/dsov_enu/data/hcvwzt90.html) > NDS Return Values from the Operating System

TSANDS return values from FFFDFEFC to FFFDFEC8 map to NDS return values: NDS Return Values (http://developer.novell.com/ndk/doc/ndslib/dsov_enu/data/hcvwzt90.html) > NDS Client Return Values - 0xFFFF FED3 through 0xFFFF FE9E.

TSANDS return values from FFFDFEC7 to FFFDFE1B map to NDS return values: NDS Return Values (http://developer.novell.com/ndk/doc/ndslib/dsov_enu/data/hcvwzt90.html) > NDS Agent return values 0xFFFF FDA7 through 0xFFFF FCFA.

9.4 Utility Library Values

This section lists the values associated with Utility Library.

- "nameSpaceType Values" on page 347
- "selectionType Values" on page 348
- "Time Zone Values" on page 349
- "Wildcard Values" on page 349

9.4.1 nameSpaceType Values

nameSpaceType can have the following values:

Value	Description
0x000	DOSNameSpace: name contains a DOS path in MBCS format.

Value	Description
0x001	MACNameSpace: name contains a Macintosh path in MBCS format.
0x002	NFSNameSpace: name contains an NFS path in MBCS format.
0x003	FTAMNameSpace: name contains an FTAM path in MBCS format.
0x004	OS2NameSpace: name contains an OS/2 path in MBCS format.
0x005	NT(Deprecated)
0x006	DOS Unicode (Deprecated)
0x007	MAC Unicode (Deprecated)
0x008	NFS Unicode (Deprecated)
0x100	DOSNameSpaceUtf8Type: **name** contains a DOS path in UTF-8 format.
0x101	MacNameSpaceUtf8Type: **name** contains a Macintosh path in UTF-8 format.
0x102	NFSNameSpaceUtf8Type: **name** contains an NFS path in UTF-8 format.
0x104	LONGNameSpaceUtf8Type: **name** contains a DOS path in UTF-8 format.

9.4.2 selectionType Values

The values that be set for selectionType is listed in the following table.

Value	Description
0x00	No selection type: If a data set name list is being built, selectionType must be "No selection type." If a selection list is being built, selectionType cannot be set to "No selection type."
0x02	NWSM_TSA_DEFINED_RESOURCE_EXC: Exclude name and all its subordinates from the scan.
0x03	NWSM_TSA_DEFINED_RESOURCE_INC: Include name and all its subordinates in the scan.
0x04	NWSM_PARENT_TO_BE_EXCLUDED: Exclude parent name and all its subordinates from the scan.
0x05	NWSM_PARENT_TO_BE_INCLUDED: Include parent name and all its subordinates in the scan.
0x08	NWSM_CHILD_TO_BE_EXCLUDED: Exclude child name from the scan.
0x09	NWSM_CHILD_TO_BE_INCLUDED: Include child name in the scan.
0x10	NWSM_EXCLUDE_CHILD_BY_FULL_NAME: Exclude child name from the scan. The full path must be contained in name
0x11	NWSM_INCLUDE_CHILD_BY_FULL_NAME: Include child name in the scan. The full path must be contained in name

All selection types, except the last two, can be applied globally to the target's data. Including or excluding a child by its full name (full path) affects one location in the target. For example, to

include all .c files in a specific directory, choose NWSM_INCLUDE_CHILD_BY_FULL_NAME
and specify the fully qualified path below:

```
SYS:SYSTEM\*.c (for NetWare)
/home/user/dir1/*.c (for Linux)
```

To include all .c files on the target, choose NWSM_CHILD_TO_BE_INCLUDED and set name to:

```
*.c
```

To find out if a selection type is supported, call NWSMTSGetTargetSelectionTypeStr.

9.4.3 Time Zone Values

The most significant four bits represents the type of time zone, while the lower twelve bits represent the time zone value.

The following table shows the specification for the upper 4 bits:

Value	Description
0	ECMA_Type_CUT: Coordinated Universal Time. Ignore the lower 12 bits.
1	ECMA_Type_Local: Local Time. The lower 12 bits contain the local time value.
2	ECMA_Type_Determined: The interpretation of the date and time (the lower 12 bits) is subject to agreement between the originator and recipient of the medium.
	Under SMS, ECMA_Type_Determined is treated as ECMA_Type_CUT
3-15	Reserved.

The following table shows the specification for the lower 12 bits:

Value	Description
-1140 through 1440	Coordinated Universal Time: Offset from CUT in 1 minute increment.
-2047	No time zone: No offset.

9.4.4 Wildcard Values

The following table contains the wildcard options.

Value	Description
0x2A	ASTERISK: Regular asterisk.
0x3F	QUESTION: Regular question mark
0xAE	SPERIOD: Special Period with the most significant bit set.
0xAA	SASTERISK: Special Asterisk with the most significant bit set.

Value	Description
0xBF	SQUESTION: Special Question with the most significant bit set.

9.5 ExtensionTag Values

extensionTag contains the following values.

Value	Macro	Description
0x00000001	NWSM_SCAN_INFO_EXTN_NFS_TAG	NWSM_SCAN_INFORMATION extension that holds NFS meta data information.
0x00000002	NWSM_RESOURCE_INFO_EXTN_NETWARE_TAG	NWSMTSGetTargetResourceInfoEx extension that holds NetWare file system resource information.
0x00000003	NWSM_RESOURCE_INFO_EXTN_UNIX_TAG	NWSMTSGetTargetResourceInfoEx extension that holds Linux file system resource information.
0x00000004	NWSM_RESOURCE_INFO_EXTN_UNSUPPORTED_TAG	NWSMTSGetTargetResourceInfoEx extension that holds information regarding unsupported TSA options by a resource.

9.6 TagVersion Values

TagVersion contains the following values.

Value	Macro
0x00000001	NWSM_EXTENSION_VERSION_1

Performance and the File System TSA

10

This chapter describes the performance model with TSAFS

10.1 Introduction

Backup has traditionally been serial in nature. While this approach was acceptable in the past, it is certainly not effective as today's backup involves several terabytes of data and tape devices are able to service several megabytes of data per second. In addition to this, file systems are generally optimized for servicing multiple client requests simultaneously. The serial nature of backup thus limits the optimal usage of the file system characteristics.

The target service agent (TSA) functions as a library that provides any service the backup engine demands. The TSA available on NetWare 6.0 or lower reads files based on engine requests and blocks till the file system services the read request. Due to this, the backup engine is unable to keep data streaming to the tape device that would allow the device to perform optimally. While the TSA library model does not limit or prevent simultaneous access to multiple files, it has traditionally been used in a serial manner that has effectively reduced the throughput of the backup system.

The Target Service Agent for NetWare 6.5 and Linux (TSAFS) have been designed to deliver performance within the parameters of the current model of usage.

10.2 Performance Model with TSAFS

The nature of backup enables predictable access of files across the file system. TSAFS has been re-designed to takes advantage of this property and incorporates read-ahead caching while maintaining backward compatibility with the current serial model of usage.

The TSA library model has been modified to de-couple the serial usage of the interface from the file system access. In this model, the TSA takes advantage of the predictable nature of requests and caches data ahead of time, so that engine requests can be serviced from the memory instead of the disk. The TSA achieves this using a multi-threading model.

The four primary tasks that constitute a backup operation and use a co-operative pre-fetching mechanism are:

- **Scan:** The Scan (NWSMTSScanDataSetBegin) defines the scope of the backup operation. The Scan thread uses this as a hint to predict and build the meta-data cache and the list of data sets to be opened. Further scan requests (NWSMTSScanNextDataSet) are serviced from this cache.

- **Open:** The Open thread works on the data set list built by the scan thread to open data sets in parallel with other tasks. Open requests (NWSMTSOpenDataSetForBackup) are serviced from this cache.

- **Read:** The Read task is implemented by multiple threads that issue simultaneous read requests to the file system and builds the data cache. This results in multiple pending I/O requests at the disk hardware which enables the subsystem to minimize the seek and rotational latencies. The read requests also enable building up a cache of data blocks ahead of engine requests (NWSMTSReadDataSet) thereby reducing the service time to the engine.

- **Close:** The Close thread implements a lazy close mechanism wherein data sets are closed asynchronously with respect to the engine close requests (NWSMTSCloseDataSet).

In the TSAFS model, each of these tasks is executed in parallel.

From this new model, it maybe evident that appropriate usage of the API would help backup engines exploit the performance benefits delivered by TSAFS.

10.3 Performance Enablers and Inhibitors

The basis for achieving performance is the ability of TSAFS to predict the backup job, and read-ahead thereby reducing the average read service time.

- To achieve higher performance, provide TSAFS with as large a set of data items as possible. This can be in the form of specifying multiple resources or a container resource that is likely to contain a large number of data sets.

- The TSAs are capable of traversing the file system tree. In this new model, this is used effectively to build the cache. It is recommended that backup engines provide the highest level resource to be backed up and do not start a job at each intermediate level in the file system tree

- Implementing selection lists in the backup engine reduces the effectiveness of the read-ahead as a subset of the data sets read by TSAFS will be discarded by the engine. These unnecessary read-aheads impact the average read service time for data sets that are actually backed up. This can be mitigated by either providing the TSA with a filtered list of data sets to be backed up or using the TSA selection list.

- It is advisable not to change the open mode during the course of the backup, if possible. The open mode provided to TSAFS is used to build a cache of open files. If the open mode is changed, all open and read-ahead operations have to be stopped, the caches destroyed and rebuilt. This will affect the average service time.

- In the new model, the open operation triggers a chain of events such as open and read-aheads. Given this, it is recommended that open is used only during the backup and not during a scan for any reason.

- It is recommended that NWSMTSScanDataSetEnd can be used to terminate the job prematurely. This assumes more importance in the current model. If this API is not used, scan, open and read-ahead continue impacting the overall system performance.

- In this model, NWSMTSReturnToParent is likely to degrade system performance as all open and read-ahead operations have to be stopped, and the caches should be destroyed and rebuilt. While this is the case, the API is useful in certain circumstances. Hence, the benefits should be understood before its usage.

- It is recommended that Close is called immediately on all data sets that were successfully read. This allows TSAFS to effectively use its resources and also prevent resource starvation in extreme cases.

10.4 Sample

The following flowchart illustrates a sample usage of the SMS API

```
                          ┌─────────┐
                          │  Start  │
                          └────┬────┘
                               ▼
                    ┌──────────────────────┐
                    │  NWSMTSScanDataSetBegin │
                    └──────────┬───────────┘
                               ▼
                          ╱─────────╲        No
                         ╱  Return    ╲──────────────┐
                         ╲  Zero?     ╱              │
                          ╲─────────╱                ▼
                               │ Yes           ╱──────────╲   Yes
                               ▼              ╱  No more    ╲──────┐
                    ┌──────────────────────┐ ╲  data sets? ╱       │
                    │ NWSMTSOpenDataSetForBackup ╲─────────╱         ▼
                    └──────────┬───────────┘                   ┌────────┐
                               ▼                               │  Done  │
                          ╱─────────╲                          └────────┘
                         ╱  Return    ╲
                         ╲  Zero?     ╱
                          ╲─────────╱
                               │ Yes
                               ▼
                    ┌──────────────────────┐
                    │   NWSMTSReadDataSet   │◄───┐
                    └──────────┬───────────┘    │
                               ▼                │
                          ╱─────────╲            │
                         ╱  Return    ╲          │
                         ╲  Zero?     ╱          │
                          ╲─────────╱            │
                               │ Yes             │
                               ▼         No      │
                          ╱─────────╲───────────┘
                         ╱  Read all  ╲
                         ╲  Data?     ╱
                          ╲─────────╱
                               │ Yes
                               ▼
                    ┌──────────────────────┐
                    │   NWSMTSCloseDataSet  │
                    └──────────┬───────────┘
                               ▼
                    ┌──────────────────────┐
                    │  NWMSTSScanNextDataSet │
                    └──────────────────────┘
```
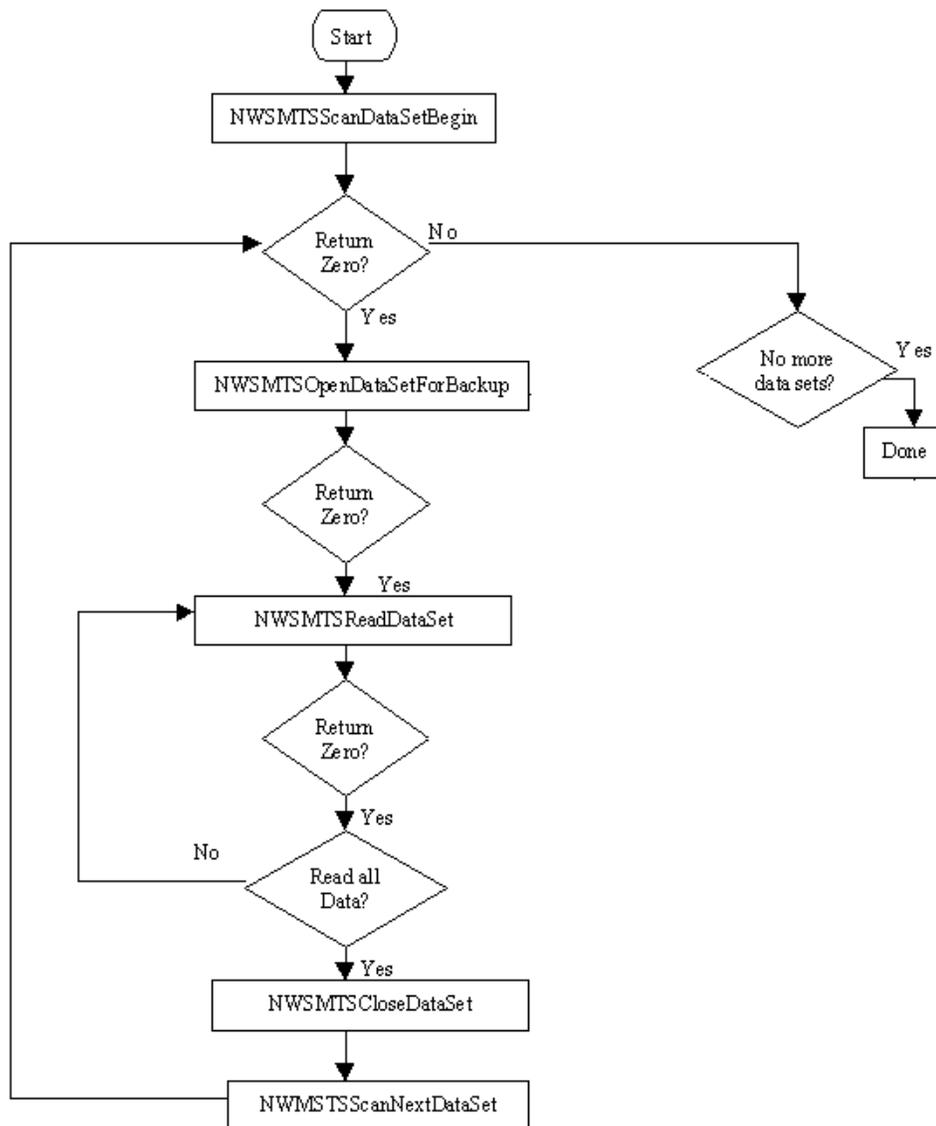
The Demonstration Engine is a sample backup engine available in source and binary form and backs up data to the disk that uses the above sample calling sequence. This can be used to understand the purpose and usage of the SMS API set.

TSATEST is a sample utility that emulates a virtual tape with infinite bandwidth that helps in analyzing the TSA performance independent of other bottlenecks. In effect, TSATEST reads the data to be backed up from SMS API and discards the data. This utility is also available in source and binary form. This can be used to understand the calling sequence and can be used to compare performance of actual backup engine implementation.

10.5 Conclusion

The TSAFS has been re-designed to exploit file system characteristics to deliver better performance. This has been done within the constraints of maintaining backward compatibility with the current models of usage. However, to take advantage of these enhancements, it is important that the backup engine appropriately use the API set as described in this chapter. It is also important to understand that the TSA and the backup engine are only two of the many parameters that influence the throughput of the backup system, disk and tape hardware being some of the other notable ones.

Obsolete Functions

<div style="text-align: right">A</div>

This section lists the obsolete functions and describes their purpose, syntax, parameters, and return values.

- "NWSMTSReadDataSets (Obsolete)" on page 356
- "NWSMTSEndReadDataSets (Obsolete)" on page 362
- "NWSMFixDirectoryPath (Obsolete)" on page 363
- "NWSMFixGenericDirectoryPath (Obsolete)" on page 365
- "NWSMTSGetTargetServiceAddress (Obsolete)" on page 367

NWSMTSReadDataSets (Obsolete)

Reads data sets on the Target Service, formats the data according to SIDF, and returns it in a buffer.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSReadDataSets (
   UINT32      connection,
   UINT32     scanSequence,
   UINT32      openMode,
   UINT32     bytesToRead,
   BUFFERPTR  buffer,
   UINT32     *bytesRead,
   UINT32     *done,
   UINT32     *readHandle);
```

Parameters

connection

   (IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServiceEx.

scanSequence

   (IN) Specifies the scanning sequence value returned by NWSMTSScanDataSetBegin.

openMode

   (IN) Specifies the generic and TSA-specific open mode to apply to all the data sets that can fit the buffer (see Open Modes for possible values).

bytesToRead

   (IN) Specifies the amount of free space in the buffer.

buffer

   (OUT) Points to the buffer to contain the data (must be at least bytesToRead bytes).

bytesRead

   (OUT) Points to the number of bytes read into buffer.

done

   (OUT) Points to a boolean value indicating if NWSMTSReadDataSets should be called again:

   TRUE Finished; no need to call NWSMTSReadDataSets again
   FALSE Call NWSMTSReadDataSets again

readHandle

   (IN/OUT) Points to the iteration handle (should be zero the first time NWSMTSReadDataSets is called).

Return Values

See for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER

Engine Developer

ReadDataSets is an enhanced version of ReadDataSet which helps in backing up small files faster. It is used during backup to collect all requested information about possibly more than one data set. Eventually, the OUT buffer will contain the data in System Independent Data Format (SIDF).

In the cases when the data set data is larger than the size of the buffer, it behaves similar to the ReadDataSet wherein it has to be called repeatedly as it returns portions of the data set at a time.

For instances when the data for more than one data set is able to fit within the buffer, it shall be appended one after another. It formats the File Headers (or Record Headers) and File Continuation Headers (or Sub-Record Headers) in the buffer. This facilitates reading more than one data set on a single call to ReadDataSets.

This call takes care of open/reading/closing of data sets (OpenDataSetForBackup, ReadDataSet and CloseDataSet as well as calling ScanNextDataSet).

The data returned in the buffer does not include unused bytes at the end. The unused value will be bytesToRead - *bytesRead. The unused part of the buffer will not become zero by this call. NWSMPadBlankSpace can be used to fill the unused space in buffer with zeroes.

Remarks

Before NWSMTSReadDataSets is called, NWSMTSScanDataSetBegin must be called to initiate the scan.

done will be freed on the last successful read.

NWSMTSReadDataSets reads the data for a set of data sets into buffer (if SIDF is used). If the buffer cannot contain all of the data set's data, NWSMTSReadDataSets must be called repeatedly to retrieve all data.

If buffer can accommodate data for more than one data set, all data for those data sets will be returned. done indicates if data for all data sets have been read. Normally when done is FALSE, bytesRead will be equal to bytesToRead except when some intermediate data sets were not read.

IMPORTANT: This API is not supported in NetWare 6.5.

See Also

Example

```
#include <smsutapi.h>
#include <smstsapi.h>


NWSM_DATA_SET_NAME_LIST *resourceName;
NWSM_DATA_SET_NAME name;
NWSM_SCAN_CONTROL scanControl;
NWSM_SCAN_INFORMATION *scanInformation=NULL;
NWSM_SELECTION_LIST *selectionList;
UINT32 sequence, openModes, bytesRead, transferBufferSpaceLeft,
      bytesWritten, transferBufferOffset, maxTransferBufferSize;
NWSM_RECORD_HEADER_INFO recordHeaderInfo = {0};
BUFFERPTR transferBuffer;

/* Setup resource name - see NWSMListTSResources and
NWSMTSScanTargetServiceResource. */
/* Gather the TSA options and get the user's input*/
/* Setup scanControl */
/* Setup selectionList - see NWSMTSGetTargetSelectionTypeStr */
/* Begin the backup session. */

NWSMTSScanDataSetBegin(connection, resourceName, &scanControl,
    selectionList, &sequence, &scanInformation, &dataSetName);


/*Set the open modes from the user's selection. The modes were
       received when the TSA options were gathered. */
openModes = user selected modes;


/* Create Transfer Buffer. The larger the TB, the faster it backs up */
transferBuffer = (BUFFERPTR) calloc(1, maxTransferBufferSize);
/* Initialize the current transferBuffer space to maximum.*/
currentTBSpace = initialTBSpace =
session.transferBufferInfo.maxTransferBufferSize -
session.transferBufferInfo.transferBufferDataOffset;
/* It will be nice to start using NWSMTSReadDataSets now. */
do
  {
  /* Check for user intervention. Hopefully he won't. */
  if ((ccode = CheckForOperatorAbort()) != 0)
    goto Return;
  /* Read as many data sets as possible in one shot. Depends on
         buffer size though. */
    NWSMTSReadDataSets (connection, sequence, openModes,
                       currentTBSpace, transferBufferPtr, &bytesRead,
&done,
                       &readHandle) ;
```

```
   /* Display number of bytes written to TB. */
          StatusTotalWritten(bytesRead);
  myHeaderSize = transferBufferData ;
  /* Update the TB data */
  transferBufferData += bytesRead ;
  myBufferSize = transferBufferData - myHeaderSize ;
  myBufferPtr = transferBufferPtr ;
  /* We are not sure which data sets got backed up with the call to
     NWSMTSReadDataSets. So get and display all data set info for all
     data sets contained in TB. User will again be happy to know. */
 while (myBufferSize)
    {
    /* While there is data in the TB, get the record/subrecord header
          from TB */
    ccode = NWSMGetRecordHeaderOnly (&myBufferPtr, &myBufferSize,
&recordHeaderInfo) ;

    if (!ccode)
      {
      /* If we have a record/subrecord, get the data from the
                  transfer buffer. */
              if ((recordHeaderInfo.dataSetInfoRetrieved ==
                                   DATA_SET_INFO_NOT_STARTED)||
                  (recordHeaderInfo.dataSetInfoRetrieved ==
DATA_SET_INFO_SPANNED))
 NWSMGetDataSetInfo (&myBufferPtr, &myBufferSize,
                      &recordHeaderInfo) ;
      /* If all data retrieved from current transfer buffer, break
                  and get the next transfer buffer. */
      if (!myBufferSize)
break;
              if (recordHeaderInfo.dataSetInfoRetrieved ==
DATA_SET_INFO_COMPLETE)
{
/* Data in the recordHeaderInfo.dataSetName and
                  recordHeaderInfo.scanInformation can now be used. */
if ((_ccode = NWSMGetOneName ((recordHeaderInfo.dataSetName, &name))
!= 0)
  {
  if (NWSMConvertError (connection, _ccode, errorMessage))
    sprintf (errorMessage, GetMessage (EM_UNDEFINED_ERROR), _ccode);
  StatusDisplayError (TRUE, GET_ONE_NAME_ERR, _ccode);
  }
else
  {
  /* Display the data set name to the user. */
  if (recordHeaderInfo.scanInformation->parentFlag)
    {
    /* Display name.name as a parent (e.g., directory) */
    StatusDirectoryName (name.nameSpaceType, name.name);
    recordHeaderInfo.pathIsFullyQualified = TRUE;
    }
  else
    {
```

```
      /* Display name.name as a child (e.g., file) */
      StatusFileName (name.name);
    if (withParentHandle)
recordHeaderInfo.pathIsFullyQualified = FALSE;
      else
recordHeaderInfo.pathIsFullyQualified = TRUE;
      }
    }
}
      /* Check again if all data retrieved from current transfer buffer,
break and get the next transfer buffer. */
        if (!myBufferSize)
break;
      /* Update TB information. */
      myBufferPtr += recordHeaderInfo.recordSize ;
      myBufferSize -= recordHeaderInfo.recordSize ;
      }
    else
      break;
      }
  transferBufferPtr  += bytesRead ;
  if (done)
    {
    NWSMPadBlankSpace ( transferBufferPtr, currentTBSpace-bytesRead));
    ccode = WriteTransferBufferToMedia ( &bufferIndex, isFirstBuffer,
&transferBufferData, FALSE, &transferBufferPtr);
    }
    else
    {
    ccode = WriteTransferBufferToMedia ( &bufferIndex, isFirstBuffer,
&transferBufferData, TRUE, &transferBufferPtr);
    }
  if (ccode != 0)
    {
    #if defined (DEBUG_CODE)
DEBUG_BEGIN
  printf ("BkTgt: WTBToMedia = 0x%X (%s:%u)\n",
ccode, __FILE__, __LINE__);
DEBUG_END
    #endif
    goto Return;
    }
  break;
  }
if (isFirstBuffer)
  isFirstBuffer = FALSE;
      currentTBSpace = initialTBSpace;
}
      }while (!done) ;
      /* We've reached the end and that was real fast. It will be nice
to
      clean up. */
      /* Cleaning up is normally a pain but its so easy here*/
```

Returns NWSMTSEndReadDataSets (connection, &readHandle);

NWSMTSEndReadDataSets (Obsolete)

Terminates the read session started by NWSMTSReadDataSets when all the data sets have not been completed read.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSEndReadDataSets (
   UINT32   connection,
   UINT32  *readHandle);
```

Parameters

connection

(IN) Specifies the connection information returned by NWSMTSConnectToTargetService or NWSMTSConnectToTargetServiceEx.

readHandle

(IN/OUT) Points to the handle freed when NWSMTSEndReadDataSets returns.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

Before NWSMTSEndReadDataSets is called, NWSMTSScanDataSetBegin must be called to initiate the scan and NWSMTSReadDataSets must be called to initiate the read.

NWSMTSEndReadDataSets frees all the memory allocated within readHandle. The readHandle parameter internally maintains the sequence structure, the set handle, and the read state.

If NWSMTSReadDataSets is called until all data sets have been read, all memory allocated within readHandle will be freed the last time NWSMTSReadDataSets is called so calling NWSMTSEndReadDataSets is not required.

IMPORTANT: This API is not supported in NetWare 6.5.

See Also

NWSMTSReadDataSet (page 51), NWSMTSReadDataSets (Obsolete) (page 356), NWSMTSScanDataSetEnd (page 58), NWSMTSScanNextDataSet (page 53)

NWSMFixDirectoryPath (Obsolete)

Formats a fully qualified parent directory path according to the specified NetWare name space specifications, but is obsolete. Call NWSMTSFixDataSetName (page 85) instead.

Syntax

```
#include <smsutapi.h>

STRING NWSMFixDirectoryPath (
   STRING          *path,
   UINT32           nameSpaceType,
   STRING_BUFFER **newPath,
   NWBOOLEAN        wildAllowedOnTerminal);
```

Parameters

path

(IN/OUT) Points to the NetWare path to be fixed on input. Points to the formatted string on output.

nameSpaceType

(IN) Specifies the name space type of path (see "nameSpaceType Values" on page 347).

newPath

(OUT) Points to the buffer to contain the fixed up directory path (optional).

wildAllowedOnTerminal

(IN) Specifies if the path contains wildcard in the last path node:

TRUE Last node contains wildcards
FALSE Last node does not contain wildcards

Return Values

The following table lists the return values associated with the function.

NULL	An error occurred
Nonzero	A pointer to the formatted path or newPath.string.

Remarks

NWSMFixDirectoryPath ensures that path has a primary separator and the proper case. If path does not contain a primary separator (: or ::), it returns NULL.

Volume names on NetWare and DOS paths are always converted to upper case.

If wildAllowedOnTerminal is set and the path nodes (other than the terminal node) contain wildcards, NWSMFixDirectoryPath returns NULL.

NWSMFixDirectoryPath adds an end separator to the terminal path node if the terminal path node is a parent. If an ending separator is added to the string, newPath will contain the formatted string.

The following statement determines where the formatted path will be returned:
```
if (newPath && newpath->string)
   /* newPath contains the fixed up path. */
else
   /* path contains the fixed up path. */
```

Usually, path and nameSpaceType are determined from NWSM_DATA_SET_NAME_LIST.

The following fixes are applied to path for the specified name space (except Macintosh):

- All backslashes (\ ) are converted to forward slashes (/).
- For all colon and slash combinations (:\ or :/), the slash is removed and all characters after the slash are moved one space toward the colon.

If newPath is NULL, NWSMFixDirectoryPath allocates the needed buffer space.

Call NWSMFreeString to free the buffer memory.

If the name space requires it, a separator must follow a parent terminal node.

IMPORTANT: This API is not supported in NetWare 6.5.

NWSMFixDirectoryPath Example

```
unsigned char *p = "Vol1:\system\temp", *ptr;
STRING path = p;
UINT32 nameSpaceType = DOSNameSpace;
STRING_BUFFER *newPath = NULL;

ptr = NWSMFixDirectoryPath(path, nameSpaceType, &newPath, FALSE);

if (newPath && newPath->string)
{
   /* newPath contains the fixed up path. */
}

else
   /* path contains the fixed up path. */
```

NWSMFixGenericDirectoryPath (Obsolete)

Formats a fully qualified parent directory path according to the specified NetWare name space specifications in both Byte and Unicode formats. Call NWSMTSFixDataSetName (page 85) instead.

Syntax

```
#include <smsutapi.h>

STRING NWSMFixGenericDirectoryPath (
   void               *path,
   UINT32              nameSpaceType,
   GENERIC_BUFFER **newPath,
   NWBOOLEAN           wildAllowedOnTerminal);
```

Parameters

path

(IN/OUT) Points to the NetWare path to be formatted in Byte or Unicode format.

nameSpaceType

(IN) Specifies the name space type of name (see "nameSpaceType Values" on page 347).

newPath

(OUT) Points to the buffer to contain the formatted directory path.

wildAllowedOnTerminal

(IN) Specifies if the path contains wildcard in the last path node:

TRUE Last node contains wildcards
FALSE Last node does not contain wildcards

Return Values

The following table lists the return values associated with the function.

NULL	Error
Nonzero	A pointer to the fixed up path in path or newPath.string.

Remarks

NWSMFixGenericDirectoryPath ensures that path has a primary separator and the proper case. If path does not contain a primary separator (: or ::), it returns NULL.

Volume names on NetWare and DOS paths are always converted to upper case.

If wildAllowedOnTerminal is set and the path nodes (other than the terminal node) contain wildcards, NWSMFixGenericDirectoryPath returns NULL.

NWSMFixGenericDirectoryPath adds an end separator to the terminal path node if the terminal path node is a parent. If an ending separator is added to the string, newPath will contain the formatted string.

The following statement determines where the formatted path will be returned:

```
if (newPath && newpath->string)
   /* newPath contains the fixed up path. */
else
   /* path contains the fixed up path. */
```

Usually, path and nameSpaceType are determined from NWSM_DATA_SET_NAME_LIST.

The following fixes are applied to path for the specified name space (except Macintosh):

- ◆ All backslashes (\ ) are converted to forward slashes (/).
- ◆ For all colon and slash combinations (:\ or :/), the slash is removed and all characters after the slash are moved one space toward the colon.

If newPath is NULL, NWSMFixGenericDirectoryPath allocates the needed buffer space.

Call NWSMFreeString to free the buffer memory.

If the name space requires it, a separator must follow a parent terminal node.

NWSMFixGenericDirectoryPath Example

```
unsigned char *p = "Vol1:\system\temp", *ptr;
STRING path = p;
UINT32 nameSpaceType = DOSNameSpace;
STRING_BUFFER *newPath = NULL;

ptr = NWSMFixDirectoryPath(path, nameSpaceType, &newPath, FALSE);

if (newPath && newPath->string)
{
   /* newPath contains the fixed up path. */
}

else
   /* path contains the fixed up path. */
```

NWSMTSGetTargetServiceAddress (Obsolete)

Returns the Target Service physical address.

Syntax

```
#include <smstsapi.h>

CCODE NWSMTSGetTargetServiceAddress (
   UINT32   connection,
   STRING   targetServiceName,
   UINT32  *addressType,
   STRING   address);
```

Parameters

connection

(IN) Specifies the connection information returned by NWSMConnectToTSA.

targetServiceName

(IN) Specifies the Target Service name returned by NWSMTSScanTargetServiceName or NWSMTSListTargetServices.

addressType

(OUT) Points to the physical address type of address.

address

(OUT) Returns the Target Service's physical address in binary form.

Return Values

See Section 9.3, "Target Service Return Values," on page 339 for more information.

The following table lists the return values associated with the function.

0x00000000	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFD2	NWSMTS_NO_MORE_DATA
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER

Remarks

Before NWSMTSGetTargetServiceAddress is called, the engine must be connected to the TSA that has access to the specified Target Service.

The values that can be returned by addressType is listed in the following table

Value	Byte Information
SPX	0x1 (12 bytes)
TCPIP	0x2 (4 bytes)
ADSP	0x3 (4 bytes)

The engine must ensure that address is long enough to handle the protocol address.

See Also

Example

```
char buffer[NWSM_MAX_TARGET_SRVC_NAME_LEN], addrBuffer[12];
STRING targetServiceName = buffer, address = addrBuffer;
UINT32 addressType;

/* Get target service name and copy it to targetServiceName. See the
example code of NWSMTSScanTargetServiceName or
NWSMTSListTargetServices. */

NWSMTSGetTargetServiceAddress(connection, targetServiceName,
&addressType, address);
```

Revision History

The following table outlines the changes made to the Storage Management Services documentation from July 2000 onwards.

February 2008	◆ Updated the definition of Section 6.2, "Data Set Name Functions," on page 198 with the maximum limit of 255 names and paths. ◆ Added new error code NWSMUT_BUFFER_OVERFLOW to the NWSMPut functions in Section 6.2, "Data Set Name Functions," on page 198.
August 2007	Corrected the following values in the Remarks section of the NWSM_SCAN_CONTROL (page 162) function: ◆ 0xFFFFFFFB with 0xFFFFFFFAL ◆ 0xFFFFFFFA with 0xFFFFFFF9L ◆ 0xFFFFFFF9 with 0xFFFFFFF8L

June 2007	◆ Updated the Remarks sections of the following APIs:
	
	    ◆ Added new flag NWSM_AUTH_CASA_Token to NWSMTSConnectToTargetServiceEx (page 68)
	
	    ◆ Added new scan types NWSM_OR_DATE_TIME_FILTER, NWSM_EXCLUDE_SECONDARY_DATA_STREAMS, and NWSM_INCLUDE_SOFTLINK_CHILD to NWSM_SCAN_CONTROL (page 162)
	
	◆ Replaced the following datatypes in Section 6.2, "Data Set Name Functions," on page 198 and Section 6.3, "Extension Functions," on page 218:
	
	    ◆ UINT32 with SMS_HANDLE
	
	    ◆ HUGE with SM_HUGE
	
	◆ Updated the description of the "0xFFFDFFF2" on page 343 value.
	
	◆ Deleted error strings prefixed with SMERR in the Section 9.3.1, "TSAPI and SMDR Return Values," on page 339 section.

April 2006	◆ Updated the Remarks sections of the following APIs: NWSMTSGetTargetServiceAPIVersion (page 47), NWSMTSScanDataSetContinue (page 55), NWSMTSGetTargetResourceInfoEx (page 114), NWSMTSConfigureTargetService (page 99) and NWSMTSGetSupportedNameTypes (page 108). ◆ Replaced the error string NWSMTS_INVALID_CONNECTION_HANDLE with NWSMTS_INVALID_CONNECTION_HANDL ◆ Updated Sample code to resolve formatting errors. ◆ Fixed broken links

March 2006	◆ Deleted NWSMTSGetTargetServiceAddress function from Section 2.9, "Function Access Scope," on page 32.
	◆ Deleted return value 0xFFFDFFB0 in NWSMTSScanDataSetContinue (page 55) function.
	◆ Added new definition for "0xFFFDFFB0" on page 339 value.
	◆ Replaced otherInfo to other_info in the NWSM_NAME_LIST (page 161).
	◆ Changed NWSMFreeString to NWSMFreeNameList in the Remarks section of NWSMListSMDRs (page 301).
	◆ Deleted error code NWSMSD in the Remarks section of NWSMConvertError (page 297).
	◆ Replaced references of scanPattern to pattern wherever applicable.
June 2005	◆ NWSMTSGetTargetServiceAddress (Obsolete) (page 367) function is not supported since NetWare 6.0. It is marked as obsolete in the documentation.
October 2004	◆ Added information about the new Extension Functions, that are NWSMCloseExtension (page 219), NWSMGetExtension (page 220), NWSMGetFirstExtension (page 222), and NWSMGetNextExtension (page 224).
	◆ Added information on the extensions and their fields in Section 5.7, "Extensions," on page 182.
	◆ Added information about the newly added Utility Library structures; these are NWSM_EXTENSION_INFORMATION (page 317), NWSM_RESOURCE_INFO_EXTN_NETWARE_DATA_1 (page 328), NWSM_RESOURCE_INFO_EXTN_UNIX_DATA_1 (page 330), NWSM_RESOURCE_INFO_EXTN_UNSUPPORTED_DATA _1 (page 331), and NWSM_SCAN_INFO_EXTN_NFS_DATA_1 (page 332).
	◆ Added new Utility Library values. See ExtensionTag Values (page 350) and TagVersion Values (page 350).
	◆ Updates in the Remarks sections of Option Function NWSMTSGetUnsupportedOptions (page 127) and Miscellaneous Function NWSMTSSetArchiveStatus (page 95).
	◆ Added information about the new API NWSMTSGetTargetResourceInfoEx (page 114) .
	◆ Updated open modes with additional option in the Remarks section of NWSMTSGetOpenModeOptionString (page 104), NWSMTSGetTargetScanTypeString (page 118), and NWSM_SCAN_CONTROL (page 162). Updates are also in Target Services Generic Open Mode Values (page 337).

July 2004	◆ Included information about the new API NWSMTSGetSupportedNameTypes (page 108).
	◆ Added new action flag, NWSM_CACHING_MODE_TYPE to NWSMTSConfigureTargetService (page 99) function.
	◆ Added new return values for nameSpaceType in Data Set Name List (page 158) function.
	◆ Added new defnitions for returnNameSpaceType in NWSM_SCAN_CONTROL (page 162) function.
	◆ Added new return value for NWSMTSConnectToTargetServiceEx (page 68) and NWSMTSConfigureTargetService (page 99) functions.
	◆ Added new error code to the TSAPI and SMDR Return Values (page 339).
June 2004	◆ Added new error code to NWSMTSConnectToTargetService (page 65) function.
	◆ Added new error code to TSAPI and SMDR Return Values (page 339) .
March 2004	◆ Added new error codes to NWSMTSScanDataSetContinue (page 55) function.
	◆ Added new error codes to NWSMTSGetTargetServiceAPIVersion (page 47) function
	◆ Added new return values and error codes to NWSMTSConfigureTargetService (page 99) function
	◆ Added new return values and error codes to NWSMTSConnectToTargetServiceEx (page 68) function
	◆ Added details related to NWSM_AUTH_RAW_DATA option in NWSMTSConnectToTargetServiceEx (page 68) function and removed references to NWSM_AUTH_LOCAL_DATA option.
	◆ Modified the description in NWSMMatchName (page 253)
October 2003	Included information about the following new APIs: .
	◆ NWSMTSGetTargetServiceAPIVersion (page 47).
	◆ NWSMTSConnectToTargetServiceEx (page 68).
	Modified and enhanced "Performance and the File System TSA" on page 351.
August 2003	Included information about the new API NWSMTSConfigureTargetService (page 99).
	Included information about support for larger files (file size more than 4 GB) in NWSM_SCAN_INFORMATION (page 168).
	Updated return value for NWSM_SCAN_CONTROL (page 162).
	Included a section on "Performance and the File System TSA" on page 351.

March 2003	NWSMTSReadDataSets (Obsolete) (page 356) and NWSMTSEndReadDataSets (Obsolete) (page 362) functions are no longer supported. They are marked as obsolete in the documentation.
	Added a new TSA return value, NWSMTS_CLUSTER_TARGET_DOES_NOT_EXIST (0xFFFDFFAF)
	Rearranged the SMDR return values in "TSAPI and SMDR Return Values" on page 339 section.
	Updated the "Recovering Backup Session on Cluster Failover or Failback" on page 29 section.
	Updated the remarks section of NWSMTSConnectToTargetService (page 65) function.
	Updated the "Backing Up Data" on page 15 section.
May 2002	Added Section 2.5, "Backup and Restore of Cluster Resources," on page 29 section
	Added new function, NWSMTSScanDataSetContinue (page 55)
	Added two new error codes to NWSMConnectToTSA (page 60) function
	Removed OS/2, Macintosh, and NetWare 3.x references wherever applicable
	Updated sample deployment illustrations in Section 3.2, "Connection Functions," on page 59 section
	Updated the Section 1.1.4, "Backing Up and Restoring Data," on page 15 and Section 2.8, "Resources," on page 31 sections
	Updated NWSMListSMDRs (page 301), NWSMListTSAs (page 62), NWSMTSListTSResources (page 134), and NWSMTSScanTargetServiceResource (page 141) with cluster support details
	Updated the sample code for NWSMTSListTargetServices (page 73)
	Modified the *buffer* parameter description for NWSMTSWriteDataSet (page 154)
	Following SMDR and TSA error codes have updated information. See "TSAPI and SMDR Return Values" on page 339.
	0xFFFEFFF1 0xFFFEFFCE 0xFFFDFFCD 0xFFFDFFC0
	Corrected the name of the NWSMTSReleaseTSA function to NWSMReleaseTSA (page 76)
July 2000	Updated the "TSAPI and SMDR Return Values" on page 339 section with additional completion codes.