2

Function Descriptions

Connectivity Functions

CCODE

NWSMListTSAs - Data Requestor Fn

(STRING scanPattern,
NWSM_NAME_LIST **serviceAgentNameList);

Parameters

scanPattern	(INPUT) Passes a search string that contains all upper case characters (see "Remarks" for a list of legal search patterns).
serviceAgentNameList	(OUTPUT) Passes the address of a pointer and returns a pointer to a list of active TSAs. Do not set this parameter to null. The calling routine must call **NWSMFreeNameList** to free this list.

Completion Codes

0x0	Successful
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFD	NWSMDR_OUT_OF_MEMORY

Remarks

This function returns a list of active TSAs (see Appendix "Data Structure Description" for more information about *serviceAgentNameList*). A comprehensive list of legal search patterns is shown below. smdrName and tsaName contains either a complete or partial name. If a partial name is given, a wild character must precede or succeed the name. smdrName is always the name of the file server the TSA resides on.

Pattern	Result
smdrName.tsaName	Returns an exact match.
smdrName	Returns all TSAs on file server named smdrName.
*smdrName	Returns all TSAs on file server(s) that have names that end with smdrName.
smdrName*	Return all TSAs on file server(s) that have names that begin with smdrName.
smdrName.*	Return all TSAs on file server smdrName.
*.tsaName	Return all TSAs named tsaName.
*smdrName.tsaName	Return all TSAs named tsaName on file servers that have names that end with smdrName.
smdrName.tsaName*	Return all TSAs that begin with tsaName on file server(s)

	smdrName.
*	Return all available TSAs.
.	Return all available TSAs.

Note: Any search that has a wild card in smdrName may take a long time to return.

The returned name(s) have the following format:

smdrName.tsaName

For example, "DJ.NetWare4.0 File System" is a legal name. "DJ" is the smdrName and "NetWare 4.0 File System" is the tsaName. There is no size limit imposed by the TS API on the names.

Note: If the response from this call is too slow, you can call **NWSMListSMDRs** to get a list of available targets. After selecting the target, call **NWSMConnectToTSA** to see if the target has a TSA.

Example

```
/* This example shows method that may speed up the listings of TSAs */
NWSM_NAME_LIST *smdrList, *serviceAgentNameList = NULL;
STRING scanPattern = "*";
CHAR  chosenTarget[120];

NWSMListSMDRs(scanPattern, &smdrList);
/* Have the user select a target.  The SMDR name is the same as the target's
name.  Next pass the target's name to NWSMListTSAs. */
NWSMListTSAs(chosenTarget,   &serviceAgentNameList);
```

See Also

NWSMConnectToTSA
NWSMFreeNameList

CCODE

NWSMConnectToTSA - Data Requestor Fn

(STRING tsaName,
UINT32 *connection);

Parameters

tsaName	(INPUT) Passes an application built string or a name returned by **NWSMListTSAs**.
connection	(OUTPUT) Passes the address of a UINT32 and returns the connection information, which is used for all subsequent DR API and TS API calls.

Completion Codes

0x0	Successful
0xFFFEFFF6	NWSMDR_NO_SUCH_SMDR
0xFFFEFFF7	NWSMDR_TSA_NOT_LOADED
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFD	NWSMDR_OUT_OF_MEMORY
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER

Remarks

NWSMConnectToTSA establishes a connection between the SME and a TSA. This function allows an engine access only to a TSA, information that the TSA can return (e.g., a list of target services or connecting to a target service - see **NWSMTSListTargetServices**), and information about the target (see **NWSMTSGetTargetServiceType**). The function does not allow access to the target's data. To access the target's data, call **NWSMTSConnectToTargetService**.

Example

```
UINT32 connection;
STRING tsaName;

/* Get a list of TSAs and set tsaName to the selected TSA, see NWSMListTSAs. se*/
. . .
NWSMConnectToTSA(tsaName, &connection);
```

See Also

NWSMReleaseTSA
NWSMTSConnectToTargetService
NWSMListTSAs
NWSMFreeNameList
NWSMTSListTargetServices
NWSMTSGetTargetServiceType

CCODE

NWSMTSScanTargetServiceName

```
( UINT32 connection,
  UINT32 *sequence,
  STRING scanPattern,
  STRING serviceName);
```

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
sequence	(INPUT/OUTPUT) Passes a pointer to a sequence number. Initially, set this value to zero. If the function returns an error, **sequence* is set to 0xFFFFFFFF.
scanPattern	(INPUT) Passes a search string. Legal search patterns are: "*" Return all names "*xxxx" Return all names that end with "xxxx" "xxxx*" Return all names that begin with "xxxx" "xxxx" Find name "xxxx" where "xxxx" is one or more characters.
serviceName	(OUTPUT) Passes a NWSM_MAX_TARGET_SRVC_NAME_LEN byte buffer and returns the target's name. A null string is returned if no target is found.

Completion Codes

0x0	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFD2	NWSMTS_NO_MORE_DATA
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFD	NWSMDR_OUT_OF_MEMORY

Prerequisites

The engine must be connected to a TSA.

Remarks

This function returns the name of one target service that is available through the current TSA. In some cases, such as NetWare, a TSA services only one target. In other cases a TSA may service multiple targets (e.g., the DOS TSA). In this case, this function must be called multiple times to retrieve all specified target names.

See Also

NWSMTSGetTargetServiceType
NWSMTSListTargetServices

CCODE

NWSMTSListTargetServices

(UINT32 connection,
STRING scanPattern,
NWSM_NAME_LIST **serviceNameList);

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
scanPattern	(INPUT) Passes the search string. Legal search patterns are: "*" Return all names "*xxxx" Return all names that end with "xxxx" "xxxx*" Return all names that begin with "xxxx" "xxxx" Find name "xxxx" where "xxxx" is one or more characters.
serviceNameList	(OUTPUT) Passes the address of a pointer and returns a block of memory containing a list of available services. The maximum length of *serviceNameList→name*, NWSM_MAX_TARGET_SRVC_NAME_LEN, includes the null termination character. This parameter cannot be null.

Completion Codes

0x0	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFD	NWSMDR_OUT_OF_MEMORY

Prerequisites

The engine must be connected to a TSA.

Remarks

This function returns the names of available target services through the current TSA. *serviceNameList* uses the following data structure:

```
typedef struct _NWSM_NAME_LIST
{
   STRING name;
   _NWSM_NAME_LIST *next;
} NWSM_NAME_LIST;
```

To free the *serviceNameList*, call **NWSMFreeNameList**. To retrieve information about the target call **NWSMTSGetTargetServiceType**.

Note: This function is implemented with

NWSMTSScanTargetServiceName.

Example

```
STRING scanPattern = "*";
NWSM_LIST *serviceNameList = NULL;

/* Connect to a TSA, then make the call */
. . .
NWSMTSListTargetServices(connection, scanPattern, &serviceNameList);
```

See Also

NWSMTSGetTargetServiceType
NWSMFreeNameList

CCODE

NWSMTSGetTargetServiceAddress

(UINT32 connection,
STRING targetServiceName,
UINT32 *addressType,
STRING address);

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
targetServiceName	(INPUT) Passes a name that was returned by **NWSMTSScanTargetServiceName** or **NWSMTSListTargetServices**.
addressType	(OUTPUT) Passes the address of a UINT32 and returns *address*' physical address type. The following types are defined: 0x1 SPX 0x2 TCPIP 0x3 ADSP
address	(OUTPUT) Passes a buffer and returns the target service's physical address in binary form (i.e., a character string is not returned). Ensure that *address*' buffer is long enough to handle all the protocol address lengths listed below: SPX 12 bytes TCPIP 4 bytes ADSP 4 bytes

Completion Codes

0x0	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFD2	NWSMTS_NO_MORE_DATA
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

The engine must be connected to the TSA that has access to the target service.

Remarks

This function returns the physical network address of a target service.

See Also

NWSMTSScanTargetServiceName
NWSMTSListTargetServices

CCODE

NWSMTSGetTargetServiceType

(UINT32 connection,
STRING serviceName,
STRING serviceType,
STRING serviceVersion);

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
serviceName	(INPUT) Passes a target service name that **NWSMTSScanTargetServiceName** or **NWSMTSListTargetServices** returned. The string's buffer size is NWSM_MAX_TARGET_SRVC_NAME_LEN.
serviceType	(OUTPUT) Returns a TSA-defined service type name into a NWSM_MAX_TARGET_SRVC_TYPE_LEN byte buffer (e.g., NetWare, DOS, etc.).
serviceVersion	(OUTPUT) Returns a TSA-defined target service version string into a NWSM_MAX_TARGET_SRVC_VER_LEN long buffer.

Completion Codes

0x0	Successful
0xFFFDFFDD	NWSMTS_INVALID_PARAMETER
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

The engine must be connected to the TSA.

Remarks

This function returns the target's type and version information. The buffer sizes includes a null terminator.

Example

```
char serviceType[NWSM_MAX_TARGET_SRVC_TYPE_LEN],
   serviceVersion[NWSM_MAX_TARGET_SRVC_VER_LEN],
   chosenServiceName[NWSM_MAX_TARGET_SRVC_NAMELEN];

/* select the target service, set chosenServiceName to it, then make the call. th
*/
NWSMTSGetTargetServiceType(connection, (STRING)chosenServiceName,
   (STRING)serviceType, (STRING)serviceVersion);
```

See Also

NWSMTSListTargetServices

CCODE

NWSMTSConnectToTargetService

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

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
targetServiceName	(INPUT) Passes a target-service name that **NWSMTSScanTargetServiceName** or **NWSMTSListTargetServices** returned.
targetUserName	(INPUT) Passes the engine user's name (i.e., the user's name on a file server). The maximum buffer size is NWSM_MAX_TARGET_USER_NAME_LEN bytes.
authentication	(INPUT) Passes the authentication necessary to establish a connection with a target (e.g., a user's password on a file server). It is an unencrypted, length preceded (UINT16) string. If no authentication is passed, set the length to zero. Do not pass a null pointer. **Note:** The SMDR encrypts the password are encrypted if it is passed between SMDRs.

Completion Codes

0x0	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFC9	NWSMTS_OUT_OF_MEMORY
0xFFFDFFD7	NWSMTS_LOGIN_DENIED
0xFFFDFFF9	NWSMTS_CREATE_ERROR
0xFFFEFFF2	NWSMDR_INVALID_PROTOCOL
0xFFFEFFF8	NWSMDR_ENCRYPTION_FAILURE
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

The engine be connected to a TSA that has access to the desired target.

Remarks

This function connects an engine to the specified target. After successful connection, the engine can access the target's data. The NetWare TSAs have one target service per TSA.

targetUserName contains the user name that allows proper access to the target's data. For pre-NetWare 4.0 names, these are the bindery names. For NetWare 4.0 and above, these are the names contained in the directory.

Example

```
char targetServiceName[NWSM_MAX_TARGET_SRVC_NAMELEN ],
   targetUserName[NWSM_MAX_TARGET_USER_NAME_LEN];
char authentication[30];      /* buffer size set to arbitrary length */

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

/* select a target service name, set user's name,  and make the call.  See
   NWSMTSListTargetServices to get a service name. */
NWSMTSConnectToTargetService(&connection, (STRING)targetServiceName,
   (STRING)targetUserName, &authentication);
```

See Also

NWSMTSListTargetServices
NWSMReleaseTSA
NWSMListTSAs
NWSMFreeNameList

TSA Option Functions

CCODE

NWSMTSGetUnsupportedOptions

(UINT32 connectionID,
UINT32 *unsupportedBackupOptions,
UINT32 *unsupportedRestoreOptions);

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
unsupportedBackupOptions	(OUTPUT) Returns a bit map that represents the TSA's unsupported backup options (see "Remarks" for more information).
unsupportedRestoreOptions	(OUTPUT) Returns a bit map that represents the TSA's unsupported restore options (see the Remarks section for more information).

Completion Codes

0x0	Successful
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFEFFFB	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

The engine must be connected to the TSA and target service.

Remarks

This function returns the backup/restore options that are not supported by a TSA. The information is used to modify the user backup and restore option display. The unsupported backup options are:

0x01 NWSM_BACK_ACCESS_DATE_TIME
0x02 NWSM_BACK_CREATE_DATE_TIME
0x04 NWSM_BACK_MODIFIED_DATE_TIME
0x08 NWSM_BACK_ARCHIVE_DATE_TIME
0x10 NWSM_BACK_SKIPPED_DATA_SETS

The unsupported restore options are:

0x01 NWSM_RESTORE_NEW_DATA_SET_NAME
0x02 NWSM_RESTORE_CHILD_UPDATE_MODE
0x04 NWSM_RESTORE_PARENT_UPDATE_MODE
0x08 NWSM_RESTORE_PARENT_HANDLE

See Also

NWSMTSConnectToTSA

CCODE

NWSMTSScanTargetServiceResource

(UINT32 connection,
UINT32 *sequence,
STRING resourceName);

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
sequence	(INPUT/OUTPUT) Used to sequence through the primary resources. The engine must initially set this to zero, but it does not increment the value.
resourceName	(OUTPUT) Returns the name of a primary resource. See "TSA-Specific Resources" in Chapter 1 for more information. The maximum buffer size is NWSM_MAX_RESOURCE_LEN bytes long.

Completion Codes

0x0	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFC5	NWSMTS_RESOURCE_NAME_NOT_FOUND
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE

Prerequisites

The engine must be connected to the TSA and target service.

TSA Developer

The TSA increments *sequence*.

Remarks

This function returns the name of a single primary resource (e.g., volumes). If *sequence* is set to zero (0), *resourceName* returns the resource that represents all the resources on a target (e.g., 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 bindery, volumes, directories, and files are scanned. For more information about resources, see "TSA-Specific Resources" in Chapter 1.

To get all the primary resources, call this function repeatedly until NWSMTS_RESOURCE_NAME_NOT_FOUND is returned.

Before passing a resource name to **NWSMTSScanDataSetBegin**, you must convert it to a NWSM_DATA_SET_NAME_LIST structure. Use the data set name functions described in *Storage Management Services Utilities Library* to help convert this name.

Example

```
UINT32 sequence = 0;
char resourceName[NWSM_MAX_RESOURCE_LEN];

while (NWSMTSScanTargetServiceResource (connection, &sequence,
   (STRING)resourceName) == 0)
{
   /* build resource list */
}
   . . .
/* Have the user to chose the resource(s) to service */
```

See Also

NWSMTSScanTargetServiceResource
NWSMTSGetTargetResourceInfo

CCODE

NWSMTSListTSResources

(UINT32 connection,
NWSM_NAME_LIST **serviceResourceList);

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
serviceResourceList	(OUTPUT) Passes the address of a pointer and returns a pointer to a list of valid primary resources on the target service. The buffer size for *serviceResourceList→name* is NWSM_MAX_RESOURCE_LEN. Do not pass the address of a structure, an allocated buffer, or a null pointer.

Completion Codes

0x0	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFC5	NWSMTS_RESOURCE_NAME_NOT_FOUND
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFD	NWSMDR_OUT_OF_MEMORY
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

The engine must be connected to the TSA and target service.

Remarks

NWSMTSListTSResources returns a list of valid primary resources. The first entry in the list represents all the resources on a target (e.g., 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 bindery, volumes, directories, and files are scanned. For more information about resources, see "TSA-Specific Resources" in Chapter 1.

serviceResourceList uses the following data structure:

```
typedef struct _NWSM_NAME_LIST
{
   _NWSM_NAME_LIST *next;
   STRING name[1];
} NWSM_NAME_LIST;
```

Before passing a resource name to **NWSMTSScanDataSetBegin**, you must convert it to a NWSM_DATA_SET_NAME_LIST structure. Use the data set name functions described in *Storage Management Services Utilities Library* to help convert this name.

This function is implemented with **NWSMTSScanTargetServiceResource**.

Example

```
NWSM_NAME_LIST *serviceResourceList = NULL;

NWSMTSListTSResources(connection, &serviceResourceList);
```

See Also

NWSMFreeNameList
NWSMTSScanTargetServiceResource
NWSMTSGetTargetResourceInfo.

[1] The maximum size of *name* is 48 characters including the null terminator.

CCODE

NWSMTSBuildResourceList

(UINT32 connection);

Parameters

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

Completion Codes

0x0	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
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

The engine must be connected to a TSA and target service.

Remarks

This function builds a list of primary resources that exists under the first primary resource (or root). However, the list does not include the root. Do not call this function if there are any active scans. This function is used, for example in NetWare 3.11, to update the resource list when a volume is either mounted or dismounted.

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	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
resourceName	(INPUT) Passes a resource name that **NWSMTSScanTargetServiceResource** or **NWSMTSListTSResources** returned.
blockSize	(OUTPUT) Returns the resource's block size (e.g., disk block size).
totalBlocks	(OUTPUT) Returns the total number of blocks on the resource.
freeBlocks	(OUTPUT) Returns the number of free blocks on the resource.
resourceIsRemovable	(OUTPUT) If set, the resource is removable (e.g., removable hard disk).
purgeableBlocks	(OUTPUT) Returns the total number of blocks that are set aside as purgeable blocks.
notYetPurgeableBlocks	(OUTPUT) Returns the number of blocks that are not marked to be purged.
migratedSectors	(OUTPUT) Returns the number of migrated sectors.
precompressedSectors	(OUTPUT) Returns the number of sectors used by all data sets before they were compressed.
compressedSectors	(OUTPUT) Returns the number of sectors used by all compressed data sets.

Completion Codes

0x0	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFD0	NWSMTS_NO_MORE_NAMES
0xFFFDFFDD	NWSMTS_INVALID_PARAMETER
0xFFFDFFE9	NWSMTS_GET_SERVER_INFO_ERR
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION

0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER

Prerequisites

The engine must be connected to a TSA and target service.

Remarks

This function retrieves information about the primary resource. The function does not return resource information for all resources listed by **NWSMTSScanTargetServiceResource** or **NWSMTSListTSResources**. For example, in NetWare 3.11, the function returns resource information for volumes, but not for the file server or the bindery. NWSMTS_INVALID_PARAMETER is returned for resources that do not have any information.

Example

```
char resourceName[NWSM_MAX_RESOURCE_LEN];
UINT16 blockSize;
UINT32 totalBlocks, freeBlocks, purgeableBlocks, notYetPurgeableBlocks;
NWBOOLEAN resourceIsRemovable;

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

See Also

NWSMTSScanTargetServiceResource
NWSMTSListTSResources

CCODE

NWSMTSGetTargetScanTypeString

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

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
typeNumber	(INPUT) Passes a scan type number (1 - 31). See "Remarks" for more information.
scanTypeString	(OUTPUT) Returns a string that describes the scan type number. The maximum string length is NWSM_MAX_STRING_LEN.
required	(OUTPUT) Returns a bit map of all scan types bits that must be set (in *scanType* of NWSM_SCAN_CONTROL) if *typeNumber* is used.
disallowed	(OUTPUT) Returns a bit map of all scan types bits that must be cleared (in *scanType* of NWSM_SCAN_CONTROL) if *typeNumber* is used.

Completion Codes

0x0	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFBF	NWSMTS_SCAN_TYPE_NOT_USED
0xFFFDFFDB	NWSMTS_INVALID_SCAN_TYPE
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

The engine must be connected to a TSA and target service.

Remarks

This function returns a string that describes a TSA's scanning option and two bit maps that indicate the other scan type options that must be used or cannot not be used if scan type *typeNumber* is used. If a TSA does not support a scan type, the function returns a null string and a zero completion code (e.g., the list is not contiguous). To get every scan type, call this function repetitively until

NWSMTS_SCAN_TYPE_NOT_USED is returned.

If *typeNumber* is greater than 31, NWSMTS_INVALID_SCAN_TYPE is returned.

To indicate the user's scan type choice(s) to a TSA, set and clear the appropriate bits of *scanType* in the NWSM_SCAN_CONTROL structure when calling **NWSMTSScanDataSetBegin**.

For a list of scan types see "NWSM_SCAN_CONTROL" in Appendix B. For more information about scan types, see "Scan Type Options" in Chapter 1.

Example

```
char scanTypeString[NWSM_MAX_STRING_LEN];
UINT8 typeNumber = 0;
UINT32 required, disallowed;

while(NWSMTSGetTargetScanTypeString( connection, &typeNumber,
   (STRING)scanTypeString, &required, &disallowed) == 0)
{
   /* build scan type list */
   . . .
   typeNumber++;
}

/* Allow the user to select the scan types.  Make sure that corresponding bits
   shown by the required and disallow are set or cleared */
. . .
```

CCODE

NWSMTSGetTargetSelectionTypeStr

(UINT32 connection,
  UINT8 typeNumber,
  STRING selectionTypeString1,
  STRING selectionTypeString2);

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
typeNumber	(INPUT) Passes a selection type number (1 - 31). For more information about selection types, see "Selection Type Options" in Chapter 1.
selectionTypeString1	(OUTPUT) Returns a string (NWSM_MAX_STRING_LEN bytes long) that describes the selection type number if bit 0 of *selectionType* is not set and bit *typeNumber* of *selectionType* is set. Otherwise, a null string is returned. For more information about *selectionType*, see "Selection Type Options" in Chapter 1 and Appendix "Data Structure Description."
selectionTypeString2	(OUTPUT) Returns a string (NWSM_MAX_STRING_LEN bytes long) that describes the selection type number if bit 0 and bit *typeNumber* of *selectionType* are set. Otherwise, a null string is returned. For more information about *selectionType*, see "NWSM_SELECTION_LIST" in Appendix B.

Completion Codes

0x0	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFBE	NWSMTS_SELECTION_TYPE_NOT_USED
0xFFFDFFD9	NWSMTS_INVALID_SELECTION_TYPE
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFB	NWSMDR_UNSUPPORTED FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

The engine must be connected to TSA and target service.

Remarks

This function gets the strings for a selection type, (see "Selection Type Options" in Chapter 1 for more information). If a TSA does not support a predefined selection type, a null string and zero completion code is returned.

To get every selection type, call this function repetitively until NWSMTS_SELECTION_TYPE_NOT_USED is returned. For a list of selection types see "NWSM_SELECTION_LIST" in Appendix B.

If *typeNumber* is equal to 0, or greater than 31, NWSMTS_INVALID_SELECTION_TYPE is returned.

To indicate the user's desired selection type(s), set the corresponding bits of *selectionType* in the NWSM_SELECTION_LIST structure when calling **NWSMTSScanDataSetBegin**.

Example

```
UINT8 typeNumber = 1;
char selectionTypeString1[NWSM_MAX_STRING_LEN],
   selectionTypeString2[NWSM_MAX_STRING_LEN];

while(NWSMTSGetTargetSelectionTypeStr(connection, &typeNumber,
   (STRING)selectionTypeString1, (STRING)selectionTypeString2) == 0)
{
   /* build selection list here */
   /* You may need to keep track of which string represents which bit position.*/
      . . .
   typeNumber++;
}
```

CCODE

NWSMTSScanSupportedNameSpaces

```
( UINT32 connection,
  UINT32 *sequence,
  STRING resourceName,
  UINT32 *nameSpaceType,
  STRING nameSpaceName);
```

Parameters

connection	(INPUT) Passes a connection handle that **NWSMConnectToTSA** returned.
sequence	(INPUT/OUTPUT) Used to sequence through the supported name spaces. The engine must initially set this to 0.
resourceName	(INPUT) Passes a resource name that **NWSMTSListTSResources** or **NWSMTSScanTargetServiceResource** returned.
nameSpaceType	(OUTPUT) Returns a number that represents the name space. See "SMSUTAPI.H" for a list of the name space type constants.
nameSpaceName	(OUTPUT) Returns the name of *nameSpaceType*.

Completion Codes

0x0	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFD0	NWSMTS_NO_MORE_NAMES
0xFFFDFFE4	NWSMTS_INVALID_DATA_SET_NAME
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

The engine must be connected to a TSA and target service.

Remarks

NWSMTSScanSupportedNameSpaces returns the name of one name space.

Example

```
UINT32 sequence = 0, nameSpaceType;
char resourceName[NWSM_MAX_RESOURCE_LEN], nameSpaceName[NWSM_MAX_STRING_LEN];

while(NWSMTSScanSupportedNameSpaces(connection, &sequence, (STRING)resourceName,
&nameSpaceType, (STRING)nameSpaceName) == 0)
{
   /* build name space list */
   . . .
}
```

See Also

NWSMTSListSupportedNameSpaces

CCODE

NWSMTSListSupportedNameSpaces

(UINT32 connection,
STRING resourceName,
NWSM_NAME_LIST **supportedNameSpaces);

Parameters

connection	(INPUT) Passes a connection handle that **NWSMConnectToTSA** returned.
resourceName	(INPUT) Passes a resource name that **NWSMTSListTSResources** or **NWSMTSScanTargetServiceResource** returned.
supportedNameSpaces	(OUTPUT) Returns a list of name spaces. Do not pass a null pointer.

Completion Codes

0x0	Successful (0)
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
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
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFD	NWSMDR_OUT_OF_MEMORY
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

The engine must be connected to a TSA and target service.

Remarks

This function returns a list of name spaces supported by the specified resource. Each entry in the list is a name buffer with the first four bytes (UINT32) set to one of the following constants:

```
NWSM_ALL_NAME_SPACES              0xFFFFFFFFL
NWSM_TSA_DEFINED_RESOURCE_TYPE    0xFFFFFFFEL
NWSM_CREATOR_NAME_SPACE           0xFFFFFFFDL
NWSM_DIRECTORY_NAME_SPACE         0xFFFFFFFCL
DOSNameSpace                      0x0
MACNameSpace                      0x1
NFSNameSpace                      0x2
```

```
FTAMNameSpace                   0x3
OS2NameSpace                    0x4
```

Following the name space number is a null terminated name space name. **NWSMFreeNameList** should be called to deallocate the memory space.

This function is implemented with **NWSMTSScanSupportedNameSpaces**.

Example

```
STRING resourceName,
NWSM_NAME_LIST *supportedNameSpaces = NULL;
char *nameSpaceName;
UINT32 nameSpaceNumber;

NWSMTSListSupportedNameSpaces(connection, resourceName, &supportedNameSpaces);

nameSpaceNumber = *((UINT32 *)supportedNameSpaces->name);
nameSpaceName = &(supportedNameSpace->name[4]);
```

See Also

NWSMTSScanSupportedNameSpaces

CCODE

NWSMTSGetNameSpaceTypeInfo

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

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
nameSpaceType	(INPUT) Passes a data set's name space type that **NWSMTSScanSupportedNameSpaces** or **NWSMTSListSupportedNameSpaces** returned.
reverseOrder	(OUTPUT) If *reverseOrder* is set, the data set name is in reverse order (i.e., subordinates are to the left rather than to the right).
firstSeparator	(OUTPUT) Returns the first separator string.
secondSeparator	(OUTPUT) Returns the second separator string.

Completion Codes

0x0	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFC9	NWSMTS_OUT_OF_MEMORY
0xFFFDFFE0	NWSMTS_INVALID_NAME_SPACE_TYPE
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

The engine must be connected to a TSA and target service.

Remarks

This function returns the name space information. The reverse order and separator information are discussed in Appendix B under "NWSM_SELECTION_LIST" and "NWSM_SCAN_INFORMATION." The name space information is used to build data set names for NWSM_DATA_SET_NAME_LIST and NWSM_SELECTION_LIST (see the data set name functions in Storage Management Services Utilities Library).

The reverse order flag informs the engine if the TSA needs to build the data set names in reverse order. That is, siblings are to the left of a parent rather than to the right.

firstSeparator and *secondSeparator* must point to a valid structure or null. The function allocates memory when passed a null or if the structure does not have enough space. To Free the memory for the separators, call **NWSMFreeString** (see *Storage Management Services Library*).

Example

```
UINT32 nameSpaceType;
NWBOOLEAN reverseOrder;
STRING_BUFFER *firstSeparator = NULL, *secondSeparator = NULL;
/* The string buffer pointers must be set to NULL, if no buffer is passed */

/* Select the name space type and make the call */
. . .

NWSMTSGetNameSpaceTypeInfo(connection, nameSpaceType, &reverseOrder,
   &firstSeparator, &secondSeparator);
```

CCODE

NWSMTSGetOpenModeOptionString

(UINT32 connection,
UINT8 optionNumber,
STRING optionString);

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
optionNumber	(INPUT) Passes an option number from 0 through 23 inclusive.
optionString	(OUTPUT) Returns a string (NWSM_MAX_STRING_LEN bytes long) that describes the option.

Completion Codes

0x0	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFCB	NWSMTS_OPEN_MODE_TYPE_NOT_USED
0xFFFDFFDE	NWSMTS_INVALID_OPEN_MODE_TYPE
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

The engine must be connected to a TSA and target service.

TSA Developer

TSA developers should define their TSA-specific non-numeric open mode options starting from option number 23 (see the following table).

Remarks

This function returns TSA-specific non-numeric open mode options that are common to **NWSMTSOpenDataSetForBackup** and **NWSMTSOpenDataSetForRestore**. These options allow the engine to specify how one (as opposed to every) data set is opened.

optionNumber represents a bit position within the TSA-specific non-numeric open mode option bit map. The table below shows the *optionNumber* values and the constants that represent it in the left hand side of the table. The right

hand side of the table contains *optionNumber*'s corresponding bit mapped values. These bit map values, combined with other open modes, are passed to **NWSMTSOpenDataSetForBackup** and **NWSMTSOpenDataSetForRestore**.

Opt. No.	Open Mode Option Constant	Bit map constant	Value
0	NO_DATA_STREAMS	NWSM_NO_DATA_STREAMS	0x00000100
1	EXCLUDE_EXTENDED_ATTRIBUTE	NWSM_NO_EXTENDED_ATTRIBUTES	0x00000200
2	EXCLUDE_DIRECTORY_TRUSTEES	NWSM_NO_PARENT_TRUSTEES	0x00000400
3	EXCLUDE_FILE_TRUSTEES	NWSM_NO_CHILD_TRUSTEES	0x00000800
4	EXCLUDE_VOLUME_RESTRICTIONS	NWSM_NO_VOLUME_RESTRICTIONS	0x00001000
5	EXCLUDE_SPACE_RESTRICTIONS	NWSM_NO_DISK_SPACE_RESTRICTIONS	0x00002000
.			.
.			.
.			.
23			0x80000000

Valid open mode option numbers are from 0 through 23. If *optionNumber* is greater than 23, NWSMTS_INVALID_OPEN_MODE_MODE_TYPE is returned.

If a TSA does not support a TSA-specific non-numeric open mode option , **NWSMTSGetOpenModeOptionString** returns a null string and a 0 completion code. The function returns NWSMTS_OPEN_MODE_MODE_TYPE_NOT_USED when there are no more options defined beyond and including *optionNumber*. For more information about open mode options, see "Open Mode Options" in Chapter 1.

Example

```
UINT8 optionNumber = 0;
char optionString[(NWSM_MAX_STRING_LEN];

while(NWSMTSGetOpenModeOptionString(connection, optionNumber,
   (STRING)optionString) == 0)
{
   /* build open option mode list */
   . . .
   optionNumber++;
}

/* Allow the user to choose how a data set must be opened.
   NWSMTSOpenDataSetForBackup and NWSMTSOpenDataSetForRestore each have
   particular open modes.  These modes should be shown with the modes returned by
   NWSMTSGetOpenModeOptionString. */
```

See Also

NWSMTSOpenDataSetForBackup
NWSMTSOpenDataSetForRestore

Back Up Functions

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	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
resourceName	(INPUT) Passes the resource's name to be scanned (the starting point for the scan). *resourceName* can be obtained from *dataSetNames* of a previous scan or from these two functions: **NWSMTSScanTargetServiceResource** or **NWSMTSListTSResources**. The "Data Set Name Functions" can be used to construct the resource name (see "TSA-Specific Resources" in Chapter 1and *Storage Management Services Utility Library* for more information). **Note:** The name space type of *resourceName* should be NWSM_TSA_DEFINED_RESOURCE_TYPE
scanControl	(INPUT) Passes a pointer to a scan control structure. If the pointer is null, the TSA scans the target as if the structure's values are 0 (see "NWSM_SCAN_CONTROL" in Appendix "Data Structure Description" for more information). **Note:** *scanControl*'s settings apply to all data sets in a session.
selectionList	(INPUT) Passes a pointer to a selection list structure. To scan for all data sets, pass a null pointer; otherwise it points to a list containing the scanning patterns or explicit data set names to filter on. See "Selection Type Options" in Chapter 1 and "NWSM_SELECTION_LIST" in Appendix "Data Structure Description" for more information.
sequence	(OUTPUT) Passes a pointer to a sequence value. *sequence* is defined by the TSA and does not need to be initialized to any value.

scanInformation	(OUTPUT) Returns a data set's scan information (data set information). This information is normally used for display purposes only. Set this parameter to one of three values: null — Do not return any scan information. *scanInformation = null — The function allocates memory for the structure and returns the scan information. *scanInformation = address of allocated structure — The function returns the scan information. If there is not enough room for the information, a larger memory space is allocated. See Appendix, "Data Structure Description" for more information about this structure.
dataSetNames	(OUTPUT) Returns a list of names for one data set. Each entry has a name space type and the data set's name as it appears under that name space. The first entry in this list, is always the name space that created the data set. See Appendix, "Data Structure Description" for more information about this structure. Set *dataSetNames* to one of two values: *dataSetNames = null — The function allocates memory for the structure and returns the name space information. *dataSetNames = address of allocated structure — The function returns the name space information. If there is not enough room for the information, a larger memory space is allocated. To access the data set name see "Data Set Name Functions" in *Storage Management Services Utilities Library*. For more information, see "NWSM_DATA_SET_NAME_LIST" in Appendix "Data Structure Description."

Completion Codes

0x0	Successful
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
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

The engine must be connected to a TSA and target service.

TSA Developer

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

SME Developer

The SME can put *scanInformation* and *dataSetNames* into an SIDF transfer buffer by using **NWSMSetRecordHeader** and **NWSMUpdateRecordHeader** (see *Storage Management Services Utilities Library*). The scan information and data set names are known as the "data set information" by SIDF.

Remarks

NWSMTSScanDataSetBegin starts the scan for all specified data sets under *resourceName*. The data sets to scan are specified by *scanControl* and *selectionList*. The function returns the first data set found or the parent of a data set (see "Scanning Settings" in Appendix B). For further information about scanning see "Scanning" in Chapter 1.

To build *resourceName* and *selectionList* the "Data Set Name Functions" listed in *Storage Management Services Library* can be used. For NetWare 3.x and 4.0, wild card matching are allowed for parents and children. For NetWare 2.x, 3.x, and 4.0 all children of *resourceName* are scanned according to the information in *scanControl* and *selectionList*.

Special values for *resourceName* are "ERROR LOG" and "SKIPPED DATA SETS". "ERROR LOG" references an ASCII file, which list the errors that occurred during the back-up/restore session.

"SKIPPED DATA SETS" is a binary file, which lists the data sets that could not be backed up during the back up session. Each entry in the skipped data sets file is an NWSM_DATA_SET_NAME_LIST structure, but without the buffer size information. The reserved field of each entry contains an error code that describes the reason for skipping the data set.

To read the skipped data set file or the error log file, perform the following steps:

1. Set *resourceName* to "SKIPPED DATA SETS".
2. Call NWSMTSScanDataSetBegin and receive the name of the file in *dataSetNames*.
3. Open the file with **NWSMTSOpenDataSetForBackup**.
4. Read the file with **NWSMTSReadDataSet**.
5. Close the file with **NWSMTSCloseDataSet**.
6. End the read file session by calling **NWSMTSScanDataSetEnd**.

If *resourceName* is "ERROR LOG" or "SKIPPED DATA SETS", *scanInformation* and *dataSetNames* will not contain any valid information.

Concurrent scans can be initiated by calling **NWSMTSScanDataSetBegin** once for each resource name. If concurrent scans are used, make sure that the data area covered by a resource does not overlap the area covered by another resource. Resources that overlap each other will cause a duplication of effort. Each scan started must be terminated by a scanning error or by calling **NWSMTSScanDataSetEnd**.

Note: Each scan has a unique *sequence*. If the SME allows concurrent scans, it must keep track of each *sequence*.

Besides looking for data sets to back up, **NWSMTSScanDataSetBegin** can also be used to check for the presence of a data set(s). For example, it can be used to scan for data sets to be deleted. Just set the scan control and selection list structures, call the function, and pass *sequence* to **NWSMTSDeleteDataSet**. **NWSMTSRenameDataSet** works in a similar fashion.

Example

```
#define MAX_BUFFER_SIZE 1024
NWSM_DATA_SET_NAME_LIST *resourceName, *dataSetNames = NULL;
NWSM_SCAN_CONTROL *scanControl, *scanInformation = NULL;
NWSM_SELECTION_LIST *selectionList;
UINT32 sequence, dataSetHandle, mode, bytesRead;
CCODE ccode = 0;
char tsaData[MAX_BUFFER_SIZE];
NWSM_RECORD_HEADER_INFO recordHeaderInfo;

/* build resourceName, scanControl, and selectionList */
. . .

/* begin the scan */
NWSMTSScanDataSetBegin(connection, resourceName, scanControl, selectionList,
   &sequence, &scanInformation, &dataSetNames);

/* The open mode can be specified for each data set, but to keep things simple,
   we will set mode to apply to all data sets. */
   mode = 0;

   while(NWSMTSOpenDataSetForBackup(connection, sequence, mode, &dataSetHandle)
      == 0)
   {
      /* Here we assume that NWSMTSReadDataSet retrieves all the data on the
      first call. */
      NWSMTSReadDataSet(connection, dataSetHandle,  MAX_BUFFER_SIZE, &bytesRead,
         (BUFFERPTR)tsaData);

      /* Put the data set information and data set data into a record and then
      into a transfer buffer.  Here, we assume that the transfer buffer is
      defined somewhere else (see the Storage Device API document's
      NWSMSDOpenSessionForWriting)  */
      recordHeaderInfo.isSubRecord = FALSE;
      recordHeaderInfo→dataSetName = dataSetNames;
      recordHeaderinfo→scanInformation = scanInformation;
      recordHeaderinfo.headerSize = sizeof(NWSM_RECORD_HEADER_INFO);
      recordHeaderinfo.recordSize = bytesRead;
      recordHeaderinfo.archiveDateAndTime = todaysDateAndTime;

      /* The function below builds the record and puts it into a transfer buffer
      */
      NWSMSetRecordHeader(transferBuffer, transferBufferLeft, tsaData, TRUE,
         &recordHeaderInfo);

      /* If the transfer buffer is full, reset and update the record header and
      send the transfer buffer to SDI.  SDI will write the transfer buffer to the
      media. */
      NWSMSetRecordHeader(transferBuffer, transferBufferLeft, tsaData, TRUE,
         &recordHeaderInfo);
      NWSMUpdateRecordHeader(&recordHeaderInfo);
      . . .

      /* close the data set and get the next one */
      NWSMTSCloseDataSet(connection, &dataSetHandle);
      NWSMTSScanNextDataSet(connection, &sequence, &scanInformation,
         &dataSetNames);
   }

   /* if the scanning functions did not return and error, end the scan */
   NWSMTSScanDataSetEnd(connection, &sequence, &scanInformation, &dataSetNames);
```

See Also

NWSMTSListTSResources
NWSMTSScanTargetServiceResource
NWSMTSScanNextDataSet
NWSMTSScanDataSetEnd.
NWSMTSDeleteDataSet
NWSMTSRenameDataSet

CCODE

NWSMTSOpenDataSetForBackup

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

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
sequence	(INPUT) Passes the sequence number returned by **NWSMTSScanDataSetBegin** or **NWSMTSScanNextDataSet**.
mode	(INPUT) Passes zero or one numeric open modes and zero or more TSA-specific-non-numeric open modes (see "Remarks" for more information).
dataSetHandle	(OUTPUT) Returns a handle used for subsequent **NWSMTSReadDataSet** or **NWSMTSCloseDataSet** calls.

Completion Codes

0x0	Successful
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
0xFFFDFFE7	NWSMTS_DATA_SET_IN_USE
0xFFFDFFF6	NWSMTS_DATA_SET_EXECUTE_ONLY
0xFFFDFFFB	NWSMTS_CLOSE_BINDERY_ERROR
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

NWSMTSScanDataSetBegin initiated the scan.

TSA Developer

It is the TSA's responsibility to ensure that all attributes (e.g., last access date and time) are not altered on the target service during the back up session.

Remarks

This function opens the target's data set. To close the data set call **NWSMTSCloseDataSet**. *mode* allows the engine to define how the data set is opened and what kind of data is returned for the specified data set. The following lists show the open modes (see Chapter 1, "Open Mode Options" for more information about open modes):

Numeric-open modes for back up - Select zero or one of the following modes. If no mode is selected, the data set is opened "write deny all":

NWSM_USE_LOCK_MODE_IF_DW_FAILS (0x0001)
Attempt to open the data set with "deny write" access rights. If this fails, open the data set with read only and lock access rights.

NWSM_NO_LOCK_NO_PROTECTION (0x0002)
Attempt to open the data set with "deny write," lock, and protection access rights. If the attempt fails, open the data set anyway. The data set's state is not guaranteed if this mode is used.

NWSM_OPEN_READ_ONLY (0x0003)
Open the data set with read only access rights. Do not attempt to open the data set with any lock or protection access rights. The data set's state is not guaranteed if this mode is used.

TSA-specific-non-numeric open modes - Select zero or more of the following open modes. These modes can be ORed with one another and with one numeric-open mode. **NWSMTSGetOpenModeOptionString** indicates which modes are supported by the TSA.

NWSM_NO_DATA_STREAMS
NWSMTSReadDataSet performs a normal back up on the data set, but does not back up data stream(s).

NWSM_NO_EXTENDED_ATTRIBUTES
This mode does not back up the data set's extended attributes.

NWSM_NO_PARENT_TRUSTEES
If the data set is a parent, do not back up its trustee(s) information.

NWSM_NO_CHILD_TRUSTEES
If the data set is a child, do not back up its trustee(s) information.

NWSM_NO_VOLUME_RESTRICTIONS
This mode does not back up any volume restrictions

NWSM_NO_DISK_SPACE_RESTRICTIONS
This mode does not back up the disk space restrictions.

Example

```
See NWSMTSScanDataSetBegin
```

See Also

NWSMTSCloseDataSet
NWSMTSScanDataSetBegin
NWSMTSScanNextDataSet
NWSMTSSetRestoreOptions

CCODE

NWSMTSReadDataSet

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

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
dataSetHandle	(INPUT) Passes the handle returned by **NWSMTSOpenDataSetForBackup**.
bytesToRead	(INPUT) Passes the number of bytes to read.
bytesRead	(OUTPUT) Returns the number of bytes read.
buffer	(OUTPUT) Returns an SIDF data set data.

Completion Codes

0x0	Successful
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
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

NWSMTSScanDataSetBegin initiated the scan and **NWSMTSOpenDataSetForBackup** opened the data set.

TSA Developer

This function formats the data set according to SIDF and puts it into *buffer*. Also, this function must not modify any of the data set's access attributes (e.g., last access).

SME Developer

The data returned in *buffer* is the data set data mentioned in *System Independent Data Format*. 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 the data set data into a record and then into a transfer buffer. **NWSMSetRecordHeader** and **NWSMUpdateRecordHeader** can be used to put the information into a record and transfer buffer (see *Storage Management Services Library*).

Remarks

NWSMTSReadDataSet reads the prepared data into *data*. To get all of the data set, call this function repeatedly until *bytesRead* is less than *bytesToRead*.

Example

```
See NWSMTSScanDataSetBegin
```

See Also

NWSMTSOpenDataSetForBackup

CCODE

NWSMTSScanNextDataSet

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

Parameters

connection	(INPUT) Passes the connection information **NWSMConnectToTSA** returned.
sequence	(INPUT/OUTPUT) Passes the address of *sequence*. **NWSMTSScanDataSetBegin** initialized sequence.
scanInformation	(OUTPUT) Returns a data set's scan information. Set this parameter to one of the following three values: null — Do not return any scan information. *scanInformation = null — The function allocates memory for the structure and returns the scan information. *scanInformation = address of allocated structure — The function returns the scan information. If there is not enough room for the information, a larger memory space is allocated. For more information about this structure see "NWSM_SCAN_INFORMATION" in Appendix B.
dataSetNames	(OUTPUT) Returns a list of names for one data set. Each entry has a name space type and the data set's name as it appears under that name space. The first entry in this list, is always the name space that created the data set. See Appendix, "Data Structure Description" for more information about this structure. Set *dataSetNames* to one of two values: *dataSetNames = null — The function allocates memory for the structure and returns the name space information. *dataSetNames = address of allocated structure — The function returns the name space information. If there is not enough room for the information, a larger memory space is allocated. To access the data set name see "Data Set Name Functions" in *Storage Management Services Utilities Library*. For more information, see "NWSM_DATA_SET_NAME_LIST" in Appendix "Data Structure Description."

Completion Codes

0x0	Successful
0xFFFBFFFB	NWSMUT_OUT_OF_MEMORY
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFC4	NWSMTS_SCAN_ERROR
0xFFFDFFC9	NWSMTS_OUT_OF_MEMORY
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
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

The scan was initiated by **NWSMTSScanDataSetBegin**.

Remarks

This function continues the scan started by **NWSMTSScanDataSetBegin** and retrieves the next data set. *scanInformation* and *dataSetNames* may be reallocated by **NWSMTSScanNextDataSet** if they are not large enough.

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*.

Calling **NWSMTSScanNextDataSet** after it returns NWSMTS_NO_MORE_DATA_SETS is invalid, unless *sequence* from another **NWSMTSScanDataSetBegin** or **NWSMTSScanNextDataSet** call is passed. If **NWSMTSScanNextDataSet** is called after NWSMTS_NO_MORE_DATA_SETS is returned, NWSMTS_INVALID_SEQUENCE_NUMBER is returned.

Example

```
See NWSMTSScanDataSetBegin
```

See Also

NWSMTSScanDataSetBegin

CCODE

NWSMTSScanDataSetEnd

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

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
sequence	(INPUT/OUTPUT) Passes a sequence number that was returned by **NWSMTSScanDataSetBegin** or **NWSMTSScanNextDataSet**.
scanInformation	(INPUT) Passes a pointer to the scan information to be freed.
dataSetNames	(INPUT) Passes a pointer to the data set names to be freed.

Completion Codes

0x0	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFD8	NWSMTS_INVALID_SEQUENCE_NUMBER
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDLE
0xFFFDFFF3	NWSMTS_DATA_SET_IS_OPEN
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

A scan was initiated by **NWSMTSScanDataSetBegin**.

TSA Developers

This function must notify the server/service to free its state information tables.

Remarks

This function frees *scanInformation* and *dataSetNames*, sets the pointers to null, and sets *sequence* to zero. **NWSMTSScanDataSetEnd** must be called to terminate a scan prematurely. However, do not call this function if **NWSMTSScanDataSetBegin** or **NWSMTSScanNextDataSet** returned an error.

Example

```
See NWSMTSScanDataSetBegin
```

See Also

NWSMTSScanDataSetBegin
NWSMTSScanNextDataSet.

CCODE

NWSMTSReturnToParent

(UINT32 connection,
  UINT32 *sequence);

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
sequence	(INPUT) Passes the sequence number that **NWSMTSScanDataSetBegin** or **NWSMTSScanNextDataSet** returned.

Completion Codes

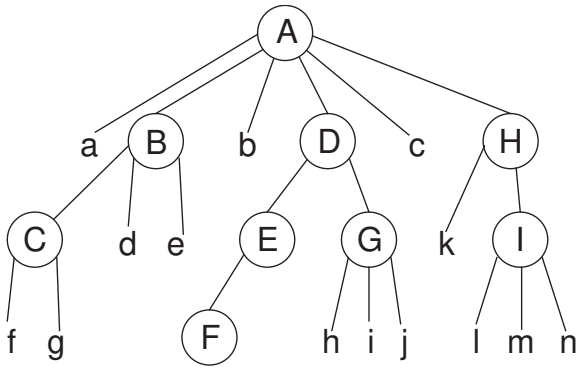
0x0	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFD8	NWSMTS_INVALID_SEQUENCE_NUMBER
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFDFFF3	NWSMTS_DATA_SET_IS_OPEN
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE

Prerequisites

A scan was initiated by **NWSMTSScanDataSetBegin**.

Remarks

This function terminates the current scan, traverses the tree one level up, and continues the scan in the next parent of the current parent. This function is used during a scan, when the engine decides that it does not need to scan the current parent any further. For example, Figure 2-1 shows a file system tree (we are assuming that the scan moves from top to bottom and left to right). If this function is called while the scan is in parent E, the scan is moved to parent G.



Scanning Order: AabcBdeCfgDEFGhijHkIlmn

Figure 2-1. Continuing the Scan

See Also

NWSMTSScanDataSetBegin
NWSMTSScanNextDataSet

CCODE

NWSMTSRenameDataSet

(UINT32 connection,
UINT32 sequence,
UINT32 nameSpaceType,
STRING newDataSetName);

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
sequence	(INPUT) Passes a data set's sequence number. The number is returned by **NWSMTSScanDataSetBegin** or **NWSMTSScanNextDataSet**.
nameSpaceType	(INPUT) Passes *newDataSetName*'s name space type (*nameSpaceType* indicates the path format used by *newDataSetName*). The name space type can be retrieved from the NWSM_DATA_SET_NAME_LIST structure that **NWSMTSScanDataSetBegin** or **NWSMTSScanNextDataSet**.
newDataSetName	(INPUT) Passes the data set's new name. *newDataSetName* must only contain the terminal node name (i.e., the last name node in a path). Do not set this parameter to null.

Completion Codes

0x0	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
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

NWSMTSScanDataSetBegin or **NWSMTSScanNextDataSet** returned a valid sequence number.

Remarks

NWSMTSRenameDataSet provides an engine the ability to rename existing data sets before restoring them. This function cannot relocate a data set to another directory or

logical location. To move a data set to another location, see **NWSMTSOpenDataSetForRestore**.

> **Note:** This function may not apply to all target services, because some services may not have a file system or the ability to rename a data set.

See Also

NWSMTSScanDataSetBegin
NWSMTSScanNextDataSet

CCODE

NWSMTSDeleteDataSet

(UINT32 connection,
UINT32 sequence);

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
sequence	(INPUT) Passes a data set's sequence number. The number was returned by **NWSMTSScanDataSetBegin** or **NWSMTSScanNextDataSet**.

Completion Codes

0x0	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFD8	NWSMTS_INVALID_SEQUENCE_NUMBER
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFDFFF1	NWSMTS_DELETE_ERR
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

One of the scanning function returned a valid sequence number.

Remarks

NWSMTSDeleteDataSet removes the data set specified by *connection* and *sequence*.

> **Note:** The data set specified by connection and sequence cannot contain wild cards.

See Also

NWSMTSScanDataSetBegin
NWSMTSScanNextDataSet

CCODE

NWSMTSSetArchiveStatus

(UINT32 connection,
UINT32 dataSetHandle,
UINT32 setFlag,
UINT32 archivedDateAndTime);

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
dataSetHandle	(INPUT) Passes a handle that **NWSMTSOpenDataSetForBackup** or **NWSMTSOpenDataSetForRestore** returned.
setFlag	(INPUT) Passes zero or more of the following flags (there is no default): NWSM_CLEAR_MODIFY_FLAG: 0x0001 NWSM_SET_ARCHIVE_DATE_AND_TIME: 0x0002 NWSM_SET_ARCHIVER_ID: 0x0004
archivedDateAndTime	(INPUT) Passes a DOS packed date and time value (see "DOS Date and Time Functions" in *Storage Management Services Utilities Library* for more information).

Completion Codes

0x0	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFBD	NWSMTS_SET_FILE_INFO_ERR
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

NWSMTSSetArchiveStatus should be called before closing the data set.

Note: NWSMTSOpenDataSetForBackup or **NWSMTSReadDataSet** does not alter the access date and time.

Restore Functions

CCODE

NWSMTSSetRestoreOptions

(UINT32 connection,
NWBOOLEAN checkCRC,
NWBOOLEAN dontCheckSelectionList,
NWSM_SELECTION_LIST *selectionList);

Parameters

connection	(INPUT) Passes the connection information returned by **NWSMConnectToTSA**.
checkCRC	(INPUT) If set, *checkCRC* causes the TSA to check the data set's CRC. This enhances data integrity, but decreases performance slightly. If no CRC was generated, no checking is done.
dontCheckSelectionList	(INPUT) *dontCheckSelectionList* affects how **NWSMTSWriteDataSet** functions. If set to FALSE, *dontCheckSelectionList* tells **NWSMTSWriteDataSet** to compare the received data set name against *selectionLis*t to see if it is included in or excluded from the restore session. Setting *dontCheckSelectionList* to TRUE, tells **NWSMTSWriteDataSet** to ignore *selectionList* because the engine knows the data set is included in the restore session (the engine previously called **NWSMTSIsDataSetExcluded**). See "Remarks" for more information.
selectionList	(INPUT) Passes a pointer to a selection list structure containing a list of data sets to restore. Passing a null clears the previous selection list. See "Selection Type Options" in Chapter 1 and "NWSM_SELECTION_LIST" in appendix "Data Structure Description" for more information. The "Data Set Name Functions" described in *Storage Management Services Library* can be used to help create this list.

Completion Codes

0x0	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
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE

0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

This function must be called before **NWSMTSOpenDataSetForRestore**, **NWSMTSIsDataSetExcluded**, and **NWSMTSWriteDataSet** is called.

Remarks

This function sets the restore selection list for the session and allows the engine to determine the mechanism used to restore a data set (the description for *dontCheckSelectionList* will discuss this further). *selectionList* is used internally by **NWSMTSIsDataSetExcluded** and **NWSMTSWriteDataSet**.

NWSMTSSetRestoreOptions can be called more than once; however, the previous *selectionList* will be replaced. This function rebuilds the resource list (e.g., volume information under NetWare), that was calculated at the time of connection. The list is rebuilt because a volume(s) may have been mounted or dismounted, or the volume's name space information may have changed. If the engine wants the resource list rebuilt, set *selectionList* to null or call **NWSMTSBuildResourceList**.

NWSMTSIsDataSetExcluded and **NWSMTSWriteDataSet** compare the same data set against *selectionList*. To prevent this double checking, when both functions are used, set *dontCheckSelectionList* to TRUE. In this manner, only **NWSMTSIsDataSetExcluded** compares the data set against *selectionList*. If **NWSMTSIsDataSetExcluded** is not used, set *dontCheckSelectionList* to FALSE.

See Also

NWSMTSIsDataSetExcluded
NWSMTSOpenDataSetForRestore
NWSMTSWriteDataSet

CCODE

NWSMTSIsDataSetExcluded

(UINT32 connection,
NWBOOLEAN isParent,
NWSM_DATA_SET_NAME_LIST **dataSetName);

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
isParent	(INPUT) Passes a flag that indicates whether the data is a parent or a child.
dataSetName	(INPUT) Passes a data set's fully qualified path.

Completion Codes

0x0	FALSE
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFDC	NWSMTS_INVALID_PATH
0xFFFDFFDD	NWSMTS_INVALID_PARAMETER
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFDFFF7	NWSMTS_DATA_SET_EXCLUDED
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

NWSMTSSetRestoreOptions was called and its parameter, *dontCheckSelectionList*, was set to TRUE.

TSA Developer

To build *dataSetName* the "Data Set Name Functions" listed in *Storage Management Services Library* can be used.

Remarks

This function uses the selection list passed in by **NWSMTSSetRestoreOptions** to decide if the data set is excluded. This function is used to help speed up the restore process and does not have to be used. For more information, see **NWSMTSSetRestoreOptions** remarks section.

dataSetName can be can be taken from a data base, from a transfer buffer received from SDI, etc. *dataSetName* must be a fully qualified path.

If the data set name is not already in a NWSMTS_DATA_SET_NAME_LIST structure, the "Data Set Name Functions" shown in *Storage Management Services Utilities Library* can be used.

See Also

NWSMTSSetRestoreOptions

CCODE

NWSMTSOpenDataSetForRestore

```
( UINT32 connection,
  UINT32 parentHandle,
  NWSM_DATA_SET_NAME_LIST *newDataSetName,
  UINT32 mode,
  UINT32 *dataSetHandle);
```

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
parentHandle	(INPUT) Passes a handle (from a previous **NWSMTSOpenDataSetForRestore** call) to the parent data set. If the handle is 0, the data set must contain a full path.
newDataSetName	(INPUT)(Optional) Passes a data set's new name. Set this parameter to null to keep the data set's original name. *newDataSetName* must be set if *parentHandle* is set to zero and the data set does not contain a full path name. To construct the data set name the engine can use the data set name functions described in *Storage Management Services Utilities Library*. For more information about this structure, see "NWSM_DATA_SET_NAME_LIST" in Appendix B.
mode	(INPUT) Passes the open mode(s). See "Remarks" for more information.
dataSetHandle	(OUTPUT) Returns a data set handle that is used by **NWSMTSWriteDataSet**, **NWSMTSCloseDataSet**, and **NWSMTSSetArchiveStatus**.

Completion Codes

0x0	Successful
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
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

SME Developer

To build *newDataSetName* the "Data Set Name Functions" listed in *Storage Management Services Library* can be used.

Remarks

This function initializes a data set handle.

Full Paths And Validation

To restore a data set, a fully qualified data set name[2] is required. A fully qualified path consists of one of the following:

- A parent data set handle and a terminal node name.

  or

- A fully qualified data set name.

[2] Defined in glossary

NWSMTSOpenDataSetForRestore uses one of three options to construct a path for the data set from *parentHandle*, *newDataSetName*, and/or the data set name contained in the data set as follows:

1. If *parentHandle* is specified, **NWSMTSOpenDataSetForRestore** ignores all path information contained in *newDataSetName* and in the data set data. However, the function uses the terminal node name contained in *newDataSetName* or in the data set if *parentHandle* does not have one.

2. If *newDataSetName* is specified, the function overrides the data set's name contained in data set data.

3. If neither *parentHandle* or *newDataSetName* are specified, the data set data must contain a fully qualified data set name.

In some cases **NWSMTSOpenDataSetForRestore** cannot verify if the constructed full path is valid because it does not have access to the data set's real full path. This real full path is in data set data. Since this function does not have access to the data set data, it must defer the constructed path's validation to **NWSMTSWriteDataSet**, which has access to the data set data. If this is the case, **NWSMTSOpenDataSetForRestore** still returns successfully. If the real full path is buried deep within the data set data, the engine may have to call **NWSMTSWriteDataSet** several times before the constructed path can be validated.

Open Modes

The open modes, although set here, are used by **NWSMTSWriteDataSet** to determine how to write the data set. The open modes are defined as follows:

- Numeric open modes for restore (defined by this function). Choose one or more modes from this list:

  NWSM_OVERWRITE_DATA_SET
  NWSMTSWriteDataSet creates the data set if it does not exist or replaces if it does exist.

  NWSM_DO_NOT_OVERWRITE_DATA_SET
  NWSMTSWriteDataSet creates the data set if it does not exist, and returns NWSM_DATA_SET_ALREADY_EXIST, if it does exist. In this mode the engine must call this

function repetitively until it returns a non-zero completion code. See **NWSMTSWriteDataSet** for more information.

Caution:
NWSM_DO_NOT_OVERWRITE_DATA_SET has a 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.

NWSM_CREATE_PARENT_HANDLE
NWSMTSWriteDataSet creates the parent handle, but will not overwrite an existing data set. This mode is similar to NWSM_DO_NOT_OVERWRITE_DATA_SET, except that NWSM_VALID_PARENT_HANDLE is returned if the data set exists.

NWSM_UPDATE_DATA_SET
If the data set on the media is newer than the one on the target, the TSA restore it.

- Non-numeric open modes for restore (defined by this function). Zero or one or more modes can be chosen from this list.

  NWSM_CLEAR_MODIFY_FLAG_RESTORE
  Clear a data set's modify flag after it is restored.

  NWSM_RESTORE_MODIFY_FLAG
  Restore the data set's modify flag to what it was at the time it was backed up.

  Note: If neither option is chosen, the data set's modified flag is set.

- TSA-specific non-numeric open modes. **NWSMTSGetOpenModeOptionString** indicates the supported by a TSA. Zero or more modes can be chosen from this list:

  NWSM_NO_DATA_STREAMS
  NWSMTSWriteDataSet performs a normal restore on the data set, but does not overwrite existing data streams.

  NWSM_NO_EXTENDED_ATTRIBUTES

This mode does not restore the data set's extended attributes.

NWSM_NO_PARENT_TRUSTEES
This mode does not restore the parent's trustee information.

NWSM_NO_CHILD_TRUSTEES
This mode does not restore a child's trustee information.

NWSM_NO_VOLUME_RESTRICTIONS
This mode does not restore any volume restrictions

NWSM_NO_DISK_SPACE_RESTRICTIONS
This mode does not restore the disk space restrictions.

In summary, when selecting the restore open mode options select zero or one of the following modes:

NWSM_OVERWRITE_DATA_SET
NWSM_DO_NOT_OVERWRITE_DATA_SET
NWSM_CREATE_PARENT_HANDLE
NWSM_UPDATE_DATA_SET

plus (ORed with) zero or one of the following modes:

NWSM_CLEAR_MODIFY_FLAG_RESTORE
NWSM_RESTORE_MODIFY_FLAG
NWSM_NO_DATA_STREAMS
NWSM_NO_EXTENDED_ATTRIBUTES
NWSM_NO_PARENT_TRUSTEES
NWSM_NO_CHILD_TRUSTEES
NWSM_NO_VOLUME_RESTRICTIONS
NWSM_NO_DISK_SPACE_RESTRICTIONS

Misplaced Data Sets

Under certain conditions a data set will not be restored to its original parent when using the data set's full path. The problem stems from backing up or restoring a data set of one name space to a parent of another name space and using the data set's full path stored on the media. The following two examples illustrate this problem. In the first example the engine backs up and restores a data set whose creator name space maximum data set name length is shorter than its parent. The following events happen.

1. Backup data set "::Macintosh_Folder:DOSfile" with its full path.

2. Rename Macintosh_Folder to Mac_Folder.

3. Create folder Macintosh_Stuff.

4. Restore the data set by using its full path information (for some reason it is placed into directory "::Macintosh_Stuff").

The data set is misplaced because the data set's full path is used to restore it. When "::Macintosh_Folder:DOSfile" is translated to a DOS path (because the data set's creator name space is DOS), the translated path is:

":\MACINTO1\DOSFILE"

Since "::Macintosh_Stuff" also translates to ":\MACINTO1\", DOSFILE is restored to that directory.

In the second example, the data set is misplaced because its full path is used and the parent was deleted. The following events happen ("t" stands for "time"):

t1. An AFP user creates the following directory structure:

::AFP dir:DOSFILE (AFP name space, creator)
:\AFPDIR\DOSFILE (DOS name space equivalent)

t2. "::AFP dir:DOSFILE" is backed up.

t3. The entire directory structure is deleted.

t4. A DOS user creates the following directory:

:\AFPDIR\ (DOS name space, creator)
::AFPDIR: (AFP name space equivalent)

t5. A DOS file "DOSFILE" is created under ":\AFPDIR\".

t6. Restore the data set using its full path; however it is placed into the directory, "\AFPDIR\", created at t4 instead of "\AFPDIR0\", its real parent, which was created at t1.

The data set is misplaced because the restoration process follows this rule:

"Restore all data sets to the same location within the name space that created them when full paths are used."

This is how the data set is misplaced. To restore the data set,

its full path "::AFP DIR:DOSFILE" is retrieved and compared to the target's directory structure. Since the parent does not exist, it is recreated under the AFP name space as "::AFP DIR:". Notice that its DOS equivalent name is now "\AFPDIR0\" (the ordinal DOS name is taken by "\AFPDIR\" which was created at t4). Next, since the data set is a DOS data set, its full path is translated to "\AFPDIR\DOSFILE". Notice that this path places the data set into the directory created at t4 not its parent '\AFPDIR0\".

To avoid the problems depicted in both examples, we recommend that parent handles be used.

To close the data set, call **NWSMTSCloseDataSet**.

Example

```
/* This example uses the information on the media to restore the data set instead
   of a log or a database that contains the serviced data sets.  If you are not
   using SDI, substitute your media calls for SDI's  */

UINT32 parentHandle, mode = NWSM_OVERWRITE_DATA_SET, dataSetHandle;
NWSM_RECORD_HEADER_INFO recordHederInfo;

/* Connect to SDI and the media, set up the transfer buffer */
. . .

/* initialize record header info, we want the function allocate scanInformation
   for us. */
recordHeaderInfo→scanInformation = NULL;

/* Here we assume that there are not subrecords and that each record occupies one
   transfer buffer. */
while(SDI returns a transfer buffer)
{
   /* Move the transfer buffer pointer to the first record */
   . . .

   /* Now get the record header information from transfer buffer */
   NWSMGetRecordHeader(&transferBuffer, transferBufferSize, recordHeaderInfo);

   /* prepare the data set for restoration */
   NWSMTSOpenDataSetForRestore(connection, parentHandle, NULL, mode,
      &dataSetHandle);

   /* Now we write the data to target.  NWSMGetRecordHeader moved the transfer
      buffer pointer to the data set data, so we just pass that pointer to the
      function below.  The FID functions discussed in the Storage Management
      Services Utilities Library document can be used to look at the fields and
      sections.*/
   NWSMTSWriteDataSet(connection, dataSetHandle, recordHeaderInfo.recordSize,
      transferBuffer);

   /* Close the data set and get the next transfer buffer. */
   NWSMTSCloseDataSet(connection, &dataSetHandle);
}

free(recordHeaderInfo→scanInformation);
```

See Also

NWSMTSOpenDataSetForBackup
NWSMTSCloseDataSet
NWSMTSSetRestoreOptions
NWSMTSWriteDataSet

CCODE

NWSMTSWriteDataSet

(UINT32 connection,
UINT32 dataSetHandle,
UINT32 bytesToWrite,
BUFFERPTR buffer);

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
dataSetHandle	(INPUT) Passes the handle returned by **NWSMTSOpenDataSetForRestore**.
bytesToWrite	(INPUT) Passes the number of bytes to write.
buffer	(INPUT) Passes the buffer to write from. The information contained in *buffer* is the data set data contained in the transfer buffer retrieved from SDI (see *System Independent Data Format* and *Storage Device API* for more information about the data set data and transfer buffers).

Completion Codes

0x0	Successful
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
0xFFFDFFF3	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
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

NWSMTSOpenDataSetForRestore opened the data set.

TSA Developers

This function deformats the data set data and writes it out to the target. The FID functions described in *Storage Management Services Utilities Library* can be used to deformat the data set data.

SME Developers

The FID functions described in *Storage Management Services Utilities Library* can be used to deformat the data set information if needed.

Remarks

This function writes the data to the specified data set and may continue the validation of *dataSetHandle* (see **NWSMTSOpenDataSetForRestore** for more information about validation). Since this function may continue the validation of *dataSetHandle* NWSMTS_VALID_PARENT_HANDLE or NWSMTS_DATA_SET_ALREADY_EXISTS may be returned (see **NWSMTSOpenDataSetForRestore** for more information about validation).

In other words, if the restore mode was set to NWSM_DO_NOT_OVERWRITE_DATA_SET, (set in **NWSMTSOpenDataSetForRestore**) the engine must call **NWSMTSWriteDataSet** repetitively until a non-zero completion code is returned. For more information about restoring data sets see **NWSMTSOpenDataSetForRestore**'s remarks.

NWSMTSSetRestoreOptions sets up the selection list used for the restore session and also indicates whether **NWSMTSWriteDataSet** should look at the selection list.

See Also

NWSMTSOpenDataSetForRestore
NWSMTSSetRestoreOptions

Connection Termination Functions

CCODE

NWSMTSReleaseTargetService

(UINT32 *connection);

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.

Completion Codes

0x0	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFD6	NWSMTS_LOGOUT_ERROR
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER

Remarks

NWSMTSReleaseTargetService releases a connection between the engine and the target service after a session is terminated or completed.

See Also

NWSMTSConnectToTargetService

CCODE

NWSMReleaseTSA

(UINT32 *connection);

Parameters

connection	(INPUT) Passes the address of a UINT32 that contains the connection information that **NWSMConnectToTSA** returned.

Completion Codes

0x0	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER

Prerequisites

The target service must be released before this function is called.

Remarks

NWSMReleaseTSA releases the connection between the engine and the TSA. There is only one engine/TSA pair for each session, therefore 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

NWSMConnectToTSA

Miscellaneous Functions

CCODE

NWSMTSCloseDataSet

(UINT32 connection,
UINT32 *dataSetHandle);

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
dataSetHandle	(INPUT/OUTPUT) Passes the address of the data set handle that **NWSMTSOpenDataSetForBackup** or **NWSMTSOpenDataSetForRestore** returned.

Completion Codes

0x0	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFE5	NWSMTS_INVALID_DATA_SET_HANDLE
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

NWSMTSCloseDataSet closes a data set that was opened by **NWSMTSOpenDataSetForBackup** or **NWSMTSOpenDataSetForRestore** and sets *dataSetHandle* to zero. To set the archive status, call **NWSMTSSetArchiveStatus**.

> **Note:** Between the time **NWSMTSOpenDataSetForBackup** and **NWSMTSCloseDataSet** are called, it is the TSA's responsibility to ensure that the data set's attributes (e.g., last access date and time) are not altered (i.e., the access for back up is transparent).

See Also

NWSMTSOpenDataSetForBackup
NWSMTSOpenDataSetForRestore
NWSMTSSetArchiveStatus

CCODE

NWSMFreeNameList - A Utility Fn

(NWSM_NAME_LIST **nameList);

Parameters

nameList	(INPUT) Passes a pointer to a name list to be freed.

Completion Codes

0x0	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER

Remarks

NWSMFreeNameList frees the memory allocated by **NWSMTSListTargetServices**, **NWSMTSListTSResources**, **NWSMTSListSupportedNameSpaces**, and **NWSMListTSAs**. *nameList* is set to null upon successful completion.

nameList uses the following data structure:

```
typedef struct _NWSM_NAME_LIST
{
    struct _NWSM_NAME_LIST *next;
    STRING name;
} NWSM_NAME_LIST;
```

See Also

NWSMTSListTSResources
NWSMTSListTargetServices
NWSMTSListSupportedNameSpaces
NWSMListTSAs

CCODE

NWSMTSCatDataSetName

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

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
nameSpaceType	(INPUT) Passes the data set's name space type.
dataSetName	(INPUT) Passes the data set name (e.g., path) to append to.
terminalName	(INPUT) Passes the data set name (parent or child) to be appended.
terminalNameIsParent	(INPUT) If set, *terminalName* is a parent.
newDataSetName	(OUTPUT) Returns the concatenated data set name. If *terminalName* is a parent, and if the target service requires it, a separator is placed at the end of *newDataSetName*. Call **NWSMFreeString** to deallocate *newDataSetName* (see *Storage Management Services Library*).

Completion Codes

0x0	Successful
0xFFFDFFB9	NWSMTS_UNSUPPORTED_FUNCTION
0xFFFDFFC9	NWSMTS_OUT_OF_MEMORY
0xFFFDFFE7	NWSMTS_INVALID_CONNECTION_HANDL
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Prerequisites

None

Remarks

This function appends a terminal node name (i.e., a child or a parent) onto an existing data set name (path). *newDataSetName* must point to a valid structure or null. The function allocates memory if a null is passed or if the structure does not have enough space.

CCODE

NWSMTSParseDataSetName

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

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
nameSpaceType	(INPUT) Passes the data set's name space type.
dataSetName	(INPUT) Passes the data set name to be parsed.
count	(OUTPUT) Returns the number of nodes and separators in *dataSetName*.
namePositions	(OUTPUT) Returns an array of indexes containing the beginning of each node in *dataSetName* . Call free to deallocate *namePositions*.
separatorPositions	(OUTPUT) Returns an array of indexes containing the beginning of each separator in *dataSetName* . Call free to deallocate *separatorPositions*.

Completion Codes

0x0	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
0xFFFEFFFB	NWSMDR_UNSUPPORTED_FUNCTION
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 "NWSM_DATA_SET_NAME_LIST" in Appendix B for more information.

This function uses the following structure:

```
typedef struct
{
   UINT16 size;
   UINT16 buffer[1];
} UINT16_BUFFER;
```

size contains the size of the UINT16_BUFFER structure in words not the size of *buffer*.

CCODE

NWSMTSSeparateDataSetName

```
( UINT32 connection,
  UINT32 nameSpaceType,
  STRING dataSetName,
  STRING_BUFFER **parentDataSetName,
  STRING_BUFFER **childDataSetName);
```

Parameters

connection	(INPUT) Passes the connection information that **NWSMConnectToTSA** returned.
nameSpaceType	(INPUT) Passes the data set's name space type.
dataSetName	(INPUT) Passes the data set name to be separated.
parentDataSetName	(OUTPUT) Returns the name of the parent data set less the terminal node name. This parameter can be set to one of three values: parentDataSetName = null — do not return parent name *parentDataSetName = null — allocate memory and return the parent's name *parentDataSetName = address of a buffer — reallocate the buffer if it is too small and return the parent's name Call free to deallocate *parentDataSetName*.
childDataSetName	(OUTPUT) (Optional) Returns the terminal node. This parameter can be set to one of three values: childDataSetName = null — do not return child's name *childDataSetName = null — allocate memory and return child's name *childDataSetName = address of a buffer — reallocate the buffer if it is too small and return the child's name Call free to deallocate *childDataSetName*.

Completion Codes

0x0	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
0xFFFEFFFC	NWSMDR_TRANSPORT_FAILURE
0xFFFEFFFE	NWSMDR_INVALID_PARAMETER
0xFFFEFFFF	NWSMDR_INVALID_CONNECTION

Remarks

This function separates the data set name into a parent and child. *parentDataSetName* and *childDataSetName* must point to a valid structure or null. The function allocates memory if a null is passed or if the structure does not have enough space.