Tivoli Storage Manager

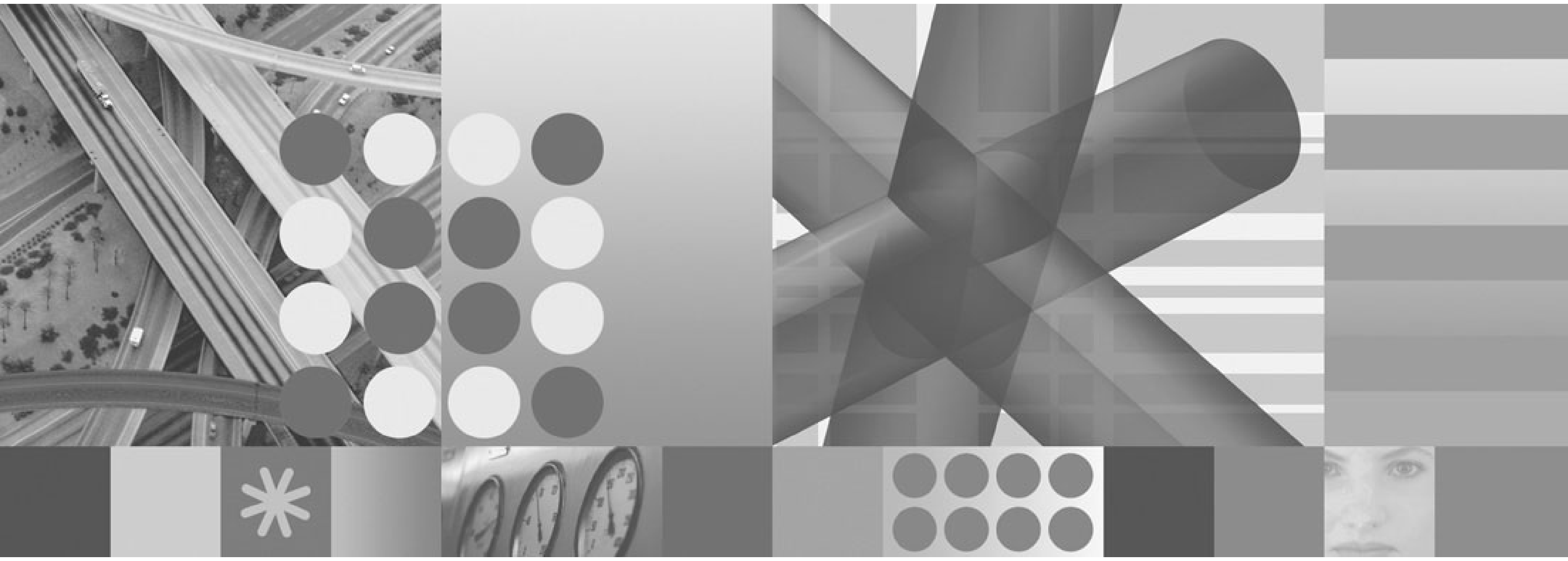
Version 6.1

Using the Application Programming Interface

SC23-9793-00

Tivoli Storage Manager

Version 6.1



Using the Application Programming Interface

SC23-9793-00

Note

Before using this information and the product it supports, read the information in “Notices” on page 211.

This edition applies to Version 6.1 of IBM Tivoli Storage Manager and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters.

Contents

Preface

This publication provides information to help you perform the following tasks on your workstation:

- Add IBM® Tivoli® Storage Manager application program interface calls to an existing application
- Write programs with general-use program interfaces that obtain the services of Tivoli Storage Manager.

The terms *hierarchical storage management* and *space management* mean the same throughout this information.

In addition to the application programming interface (API), the following programs are included on several operating systems:

- A backup-archive client program that backs up and archives files from your workstation or file server to storage, and restores and retrieves backup versions and archived copies of files to your local file systems.
- A hierarchical storage management program that automatically migrates eligible files to storage to maintain specific levels of free space on local file systems. It automatically recalls migrated files when they are accessed, and permits users to migrate and recall specific files.
- A Web backup-archive client that an authorized administrator, support person, or end user can use to perform backup, restore, archive, and retrieve services using a Web browser on a remote machine.
- An administrative client program that you can access from a Web browser or from the command line. An administrator controls and monitors server activities, defines storage management policies for backup, archive, and space management services, and sets up schedules to perform these services at regular intervals.

Who should read this publication

This publication includes instructions for a user to add API calls to an existing application. You should be familiar with C programming language and Tivoli Storage Manager functions.

Publications

Tivoli Storage Manager publications and other related publications are available online.

You can search all publications in the Tivoli Storage Manager Information Center: http://publib.boulder.ibm.com/infocenter/tsminfo/v6.

You can download PDF versions of publications from the Tivoli Storage Manager Information Center or from the IBM Publications Center at http://www.ibm.com/shop/publications/order/.

You can also order some related publications from the IBM Publications Center Web site. The Web site provides information for ordering publications from countries other than the United States. In the United States, you can order publications by calling 800-879-2755.

Tivoli Storage Manager publications

Publications are available for the server, storage agent, client, and Data Protection.

Table 1. Tivoli Storage Manager server publications

Publication title	Order number
IBM Tivoli Storage Manager Messages	GC23-9787
IBM Tivoli Storage Manager Performance Tuning Guide	GC23-9788
IBM Tivoli Storage Manager Problem Determination Guide	GC23-9789
IBM Tivoli Storage Manager for AIX Installation Guide	GC23-9781
IBM Tivoli Storage Manager for AIX Administrator's Guide	SC23-9769
IBM Tivoli Storage Manager for AIX Administrator's Reference	SC23-9775
IBM Tivoli Storage Manager for HP-UX Installation Guide	GC23-9782
IBM Tivoli Storage Manager for HP-UX Administrator's Guide	SC23-9770
IBM Tivoli Storage Manager for HP-UX Administrator's Reference	SC23-9776
IBM Tivoli Storage Manager for Linux Installation Guide	GC23-9783
IBM Tivoli Storage Manager for Linux Administrator's Guide	SC23-9771
IBM Tivoli Storage Manager for Linux Administrator's Reference	SC23-9777
IBM Tivoli Storage Manager for Sun Solaris Installation Guide	GC23-9784
IBM Tivoli Storage Manager for Sun Solaris Administrator's Guide	SC23-9772
IBM Tivoli Storage Manager for Sun Solaris Administrator's Reference	SC23-9778
IBM Tivoli Storage Manager for Windows Installation Guide	GC23-9785
IBM Tivoli Storage Manager for Windows Administrator's Guide	SC23-9773
IBM Tivoli Storage Manager for Windows Administrator's Reference	SC23-9779
IBM Tivoli Storage Manager Server Upgrade Guide	SC23-9554
IBM Tivoli Storage Manager for System Backup and Recovery Installation and User's Guide	SC32-6543

Table 2. Tivoli Storage Manager storage agent publications

Publication title	Order number
IBM Tivoli Storage Manager for SAN for AIX Storage Agent User's Guide	SC23-9797
IBM Tivoli Storage Manager for SAN for HP-UX Storage Agent User's Guide	SC23-9798
IBM Tivoli Storage Manager for SAN for Linux Storage Agent User's Guide	SC23-9799
IBM Tivoli Storage Manager for SAN for Sun Solaris Storage Agent User's Guide	SC23-9800
IBM Tivoli Storage Manager for SAN for Windows Storage Agent User's Guide	SC23-9553

Table 3. Tivoli Storage Manager client publications

Publication title	Order number
IBM Tivoli Storage Manager for UNIX and Linux: Backup-Archive Clients Installation and User's Guide	SC23-9791
IBM Tivoli Storage Manager for Windows: Backup-Archive Clients Installation and User's Guide	SC23-9792
IBM Tivoli Storage Manager for Space Management for UNIX and Linux: User's Guide	SC23-9794
IBM Tivoli Storage Manager for HSM for Windows Administration Guide	SC23-9795
IBM Tivoli Storage Manager Using the Application Program Interface	SC23-9793
Program Directory for IBM Tivoli Storage Manager z/OS Edition Backup-Archive Client	GI11-8912
Program Directory for IBM Tivoli Storage Manager z/OS Edition Application Program Interface	GI11-8911

Table 4. Tivoli Storage Manager Data Protection publications

Publication title	Order number
IBM Tivoli Storage Manager for Advanced Copy Services: Data Protection for Snapshot Devices Installation and User's Guide	SC33-8331
IBM Tivoli Storage Manager for Databases: Data Protection for Microsoft SQL Server Installation and User's Guide	SC32-9059
IBM Tivoli Storage Manager for Databases: Data Protection for Oracle for UNIX and Linux Installation and User's Guide	SC32-9064
IBM Tivoli Storage Manager for Databases: Data Protection for Oracle for Windows Installation and User's Guide	SC32-9065
IBM Tivoli Storage Manager for Enterprise Resource Planning: Data Protection for SAP Installation and User's Guide for DB2	SC33-6341
IBM Tivoli Storage Manager for Enterprise Resource Planning: Data Protection for SAP Installation and User's Guide for Oracle	SC33-6340
IBM Tivoli Storage Manager for Mail: Data Protection for Lotus Domino® for UNIX, Linux, and OS/400 Installation and User's Guide	SC32-9056
IBM Tivoli Storage Manager for Mail: Data Protection for Lotus Domino for Windows Installation and User's Guide	SC32-9057
IBM Tivoli Storage Manager for Mail: Data Protection for Microsoft Exchange Server Installation and User's Guide	SC23-9796
Program Directory for IBM Tivoli Storage Manager for Mail (Data Protection for Lotus Domino)	GI11-8909

Support information

You can find support information for IBM products from a variety of sources.

Getting technical training

Information about Tivoli technical training courses is available online.

Go to http://www.ibm.com/software/tivoli/education/.

Searching knowledge bases

If you have a problem with Tivoli Storage Manager, there are several knowledge bases that you can search.

You can begin with the Tivoli Storage Manager Information Center at http://publib.boulder.ibm.com/infocenter/tsminfo/v6. From this Web site, you can search all Tivoli Storage Manager publications.

Searching the Internet

If you cannot find an answer to your question in the Tivoli Storage Manager information center, search the Internet for the latest, most complete information that might help you resolve your problem.

To search multiple Internet resources, go to the support Web site for Tivoli Storage Manager at http://www.ibm.com/software/sysmgmt/products/support/IBMTivoliStorageManager.html. From there, you can search a variety of resources including:

- IBM technotes
- IBM downloads
- IBM Redbooks®

If you still cannot find the solution to the problem, you can search forums and newsgroups on the Internet for the latest information that might help you resolve your problem. To share your experiences and learn from others in the user community, go to the Tivoli Storage Manager wiki at http://www.ibm.com/developerworks/wikis/display/tivolistoragemanager/Home.

Using IBM Support Assistant

At no additional cost, you can install on any workstation the IBM Support Assistant, a stand-alone application. You can then enhance the application by installing product-specific plug-in modules for the IBM products that you use.

The IBM Support Assistant helps you gather support information when you need to open a problem management record (PMR), which you can then use to track the problem. The product-specific plug-in modules provide you with the following resources:

- Support links
- Education links
- Ability to submit problem management reports

For more information, see the IBM Support Assistant Web site at http://www.ibm.com/software/support/isa/.

Finding product fixes

A product fix to resolve your problem might be available from the IBM Software Support Web site.

You can determine what fixes are available by checking the Web site:

1. Go to the IBM Software Support Web site at http://www.ibm.com/software/tivoli/products/storage-mgr/product-links.html.
2. Click the **Support Pages** link for your Tivoli Storage Manager product.
3. Click **Download**, and then click **Fixes by version**.

Getting e-mail notification of product fixes

You can get notifications about fixes and other news about IBM products.

To receive weekly e-mail notifications about fixes and other news about IBM products, follow these steps:

1. From the support page for any IBM product, click **My support** in the upper-right corner of the page.
2. If you have already registered, skip to the next step. If you have not registered, click **Register** in the upper-right corner of the support page to establish your user ID and password.
3. Sign in to **My support**.
4. On the My support page, click **Edit profiles** in the left navigation pane, and scroll to **Select Mail Preferences**. Select a product family and check the appropriate boxes for the type of information you want.
5. Click **Submit**.
6. For e-mail notification for other products, repeat steps 4 and 5.

Contacting IBM Software Support

You can contact IBM Software Support if you have an active IBM software maintenance contract and if you are authorized to submit problems to IBM.

Before you contact IBM Software Support, follow these steps:

1. Set up a software maintenance contract.
2. Determine the business impact of your problem.
3. Describe your problem and gather background information.

Then see “Submit the problem to IBM Software Support” on page xi for information on contacting IBM Software Support.

Setting up a software maintenance contract

Set up a software maintenance contract. The type of contract that you need depends on the type of product you have.

- For IBM distributed software products (including, but not limited to, Tivoli, Lotus®, and Rational® products, as well as IBM DB2® and IBM WebSphere® products that run on Microsoft® Windows® or UNIX® operating systems), enroll in IBM Passport Advantage® in one of the following ways:
  - **Online:** Go to the Passport Advantage Web page at http://www.ibm.com/software/lotus/passportadvantage/, click **How to enroll**, and follow the instructions.
  - **By Phone:** For the phone number to call in your country, go to the IBM Software Support Handbook Web page at http://www14.software.ibm.com/webapp/set2/sas/f/handbook/home.html and click **Contacts**.

- For server software products, you can purchase a software maintenance agreement by working directly with an IBM sales representative or an IBM Business Partner. For more information about support for server software products, go to the IBM Technical support advantage Web page at http://www.ibm.com/servers/.

If you are not sure what type of software maintenance contract you need, call 1-800-IBMSERV (1-800-426-7378) in the United States. For a list of telephone numbers of people who provide support for your location, go to the Software Support Handbook page at http://www14.software.ibm.com/webapp/set2/sas/f/handbook/home.html.

Determine the business impact

When you report a problem to IBM, you are asked to supply a severity level. Therefore, you need to understand and assess the business impact of the problem you are reporting.

Severity 1	**Critical** business impact: You are unable to use the program, resulting in a critical impact on operations. This condition requires an immediate solution.
Severity 2	**Significant** business impact: The program is usable but is severely limited.
Severity 3	**Some** business impact: The program is usable with less significant features (not critical to operations) unavailable.
Severity 4	**Minimal** business impact: The problem causes little impact on operations, or a reasonable circumvention to the problem has been implemented.

Describe the problem and gather background information

When explaining a problem to IBM, it is helpful to be as specific as possible. Include all relevant background information so that IBM Software Support specialists can help you solve the problem efficiently.

To save time, know the answers to these questions:

- What software versions were you running when the problem occurred?
- Do you have logs, traces, and messages that are related to the problem symptoms? IBM Software Support is likely to ask for this information.
- Can the problem be recreated? If so, what steps led to the failure?
- Have any changes been made to the system? For example, hardware, operating system, networking software, and so on.
- Are you currently using a workaround for this problem? If so, be prepared to explain it when you report the problem.

Submit the problem to IBM Software Support

You can submit the problem to IBM Software Support online or by phone.

Online

Go to the IBM Software Support Web site at http://www.ibm.com/software/support/probsub.html. Enter your information into the appropriate problem submission tool.

By phone

For the phone number to call in your country, go to the contacts page of the IBM Software Support Handbook at http://www14.software.ibm.com/webapp/set2/sas/f/handbook/home.html.

If the problem that you submit is for a software defect or for missing or inaccurate documentation, IBM Software Support creates an Authorized Program Analysis Report (APAR). The APAR describes the problem in detail. If a workaround is possible, IBM Software Support provides one for you to implement until the APAR is resolved and a fix is delivered. IBM publishes resolved APARs on the Tivoli Storage Manager product support Web site at http://www.ibm.com/software/sysmgmt/products/support/IBMTivoliStorageManager.html, so that users who experience the same problem can benefit from the same resolutions.

Conventions used in this manual

This manual uses the following typographical conventions:

Example	Description
autoexec.ncf hsmgui.exe	A series of lowercase letters with an extension indicates program file names.
DSMI_DIR	A series of uppercase letters indicates return codes and other variables or values.
dsmQuerySessInfo	Boldface type indicates a command that you type on a command line, the name of a function call, the name of a structure, a field within a structure, or a parameter.
timeformat	Boldface italic type indicates a Tivoli Storage Manager option. The bold type is used to introduce the option, or used in an example. Occasionally, file names are entered in boldface italic for emphasis.
dateformat	Italic type indicates an option, the value of an option, a new term, a placeholder for information you provide, or for special emphasis in the text.
`maxcmdretries`	Monospace type indicates fragments of a program or information as it might appear on a display screen, such a command example.
plus sign (+)	A plus sign between two keys indicates that you press both keys at the same time.

New for IBM Tivoli Storage Manager Version 6.1

Several features in IBM Tivoli Storage Manager Version 6.1 are new for previous Tivoli Storage Manager users.

The following features are new for Tivoli Storage Manager in Version 6.1:

- The **dsmUpdateObjEx** function call updates the meta information that is associated with backup or archive objects on the server. This provides the ability to select from several archive objects with the same name.
- Mac OS X support was added in Release 5.5.1.
- Novell Netware is not supported in this release.

Chapter 1. API overview

The Tivoli Storage Manager application program interface (API) enables an application client to use storage management functions.

The API includes function calls that you can use in an application to perform the following operations:
- Start or end a session
- Assign management classes to objects before they are stored on a server
- Back up or archive objects to a server
- Restore or retrieve objects from a server
- Query the server for information about stored objects
- Manage file spaces
- Send retention events

When you, as an application developer, install the API, you receive the files that an end user of an application needs:
- The API shared library
- The messages file
- The sample client options files
- The source code for the API header files that your application needs.
- The source code for a sample application, and the makefile to build it.
- The dsmtca file (UNIX and Linux® only)

Note: References to OS/400® include both OS/400 and i5/OS®.

For 64-bit applications, all compiles should be performed using compiler options that enable 64-bit support. For example, '-q64' should be used when building API applications on AIX®, and '-m64' should be used on Linux. See the sample make files for more information.

The only communication method supported under 64-bit is TCP/IP. There is no 64-bit version of the server and shared memory is not supported. Clio/s and APPC are not supported, since there are currently no 64-bit libraries.

Note: When you install the API, ensure that all files are at the same level.

For information about installing the API, see the installation procedures in the *Tivoli Storage Manager Installing and Using the Backup-Archive Client* for your operating system.

References to UNIX and Linux include Mac OS X, OS/400, HP-UX, Solaris, and z/OS®.

Understanding configuration and options files

Configuration and options files set the conditions and boundaries under which your session runs.

You, an administrator, or an end user can set option values to:
- Set up the connection to a server
- Control which objects are sent to the server and the management class to which they are associated

You define options in one or two files when you install the API on your workstation.

On UNIX and Linux operating systems, the options reside in two options files:
- dsm.opt - the client options file
- dsm.sys - the client system options file

On other operating systems, the client options file (dsm.opt) contains all of the options.

Note: The API does not support these backup-archive client options:
- autofsrename
- changingretries
- domain
- eventlogging
- groups
- subdir
- users
- virtualmountpoint

You also can specify options on the **dsmInitEx** function call. Use the option string parameter or the API configuration file parameter.

The same option can derive from more than one configuration source. When this happens, the source with the highest priority takes precedence. See Table 5 for the priority sequence. For more information about available options and communication methods that the API supports, see the *Tivoli Storage Manager Installing and Using the Backup-Archive Client* for your operating system.

Table 5. Configuration sources in order of decreasing priority

UNIX and Linux	Windows	Description
1. dsm.sys file (client system options)	1. None.	This file contains options that a system administrator sets for UNIX and Linux only.

Table 5. Configuration sources in order of decreasing priority (continued)

UNIX and Linux	Windows	Description
2. Option string (client options)	2. Option string (all options)	One of these options takes effect when it is passed as a parameter to a dsmInitEx call. The list can contain client options such as compressalways, servername (UNIX and Linux only), or tcpserveraddr (non-UNIX). With the API option string, an application client can make changes to the option values in the API configuration file and the client options file. For example, your application might query the end user if compression is required. Depending on the user responses, you can construct an API option string with this option and pass it into the call to dsmInitEx. For information about the API option string format, see "dsmInitEx" on page 99. You also can set this parameter to NULL. This indicates that there is no API option string for this session.
3. API configuration file (client options)	3. API configuration file (all options)	The values that you set in the API configuration file override the values that you set in the Tivoli Storage Manager client options file. Set up the options in the API configuration file with values that you are appropriate in the Tivoli Storage Manager session for the end user. The values take effect when the API configuration file name is passed as a parameter in the dsmInitEx call. You also can set this parameter to NULL. This indicates that there is no API configuration file for this session.
4. dsm.opt file (client options)	4. dsm.opt file (all options)	On UNIX and Linux operating systems the dsm.opt file contains the user options only. On other operating systems, the dsm.opt file contains all options. To override the options in these files, follow the methods that are described in this table.

On OS/400, the UNIX style option processing with both dsm.sys and dsm.opt files is used. However, for compatibility reasons, the previous option file method using one QSYS file is also supported.

The type of options file processing to use is determined at run time using the following information:

1. If no other option specification is used and a file member named QOPTADSM(APIOPT) is in a library in the caller's library list, then the QOPTADSM(APIOPT) file is used and should contain all options.
2. If a QSYS file name in the format LIBRARY/FILE(MEMBER) is passed as a parameter in the **dsmInitEx** call, then the input file is used. The QOPTADSM(APIOPT) file must also exist in the caller's library list and its options are used, but options from the input parameter file have precedence.
3. Any other way of specifying the option file, such as environment variables or with a path name starting with '/', uses the UNIX style processing of dsm.sys and dsm.opt files. The two files must be in the "root" (/) file system.

Setting up the API environment

The API uses unique environment variables to locate files. You can use different files for API applications from those that the backup-archive client uses. Applications can use the **dsmSetup** function call to override the values that the environment variables set.

Tip: On Windows, the default installation directory is: %SystemDrive%\Program Files\Common Files\Tivoli\TSM\api

Table 6 lists the API environment variables by operating system.

Table 6. API environment variables

Variables	UNIX and Linux	Windows
DSMI_CONFIG	The fully-qualified name for the client options file (dsm.opt).	The fully-qualified name for the client options file (dsm.opt).
DSMI_DIR	Points to the path that contains the dsm.sys, dsmtca, en_US subdirectory, and any other national language support (NLS) language. The en_US subdirectory must contain dsmclientV3.cat.	Points to the path that contains dscenu.txt and any NLS message file.
DSMI_LOG	Points to the path for the dsierror.log file.	Points to the path for the dsierror.log file.

Chapter 2. Building and running the sample API application

The API package includes sample applications that demonstrate the API function calls in context. Install a sample application and review the source code to understand how you can use the function calls.

Select one of the following sample API application packages:
- The interactive, single-threaded application package (dapi*)
- The multithreaded application package (callmt*)
- The logical object grouping test application (dsmgrp*)
- The event-based retention policy sample application (callevnt)
- The deletion hold sample application (callhold)
- The data retention protection sample application (callret)
- The Tivoli Storage Manager buffer sample program (callbuff)

To help you get started, review the procedure to build the sample dapismp sample application by your platform:
- OS/400 (see “OS/400 operating system sample application source files”)
- UNIX or Linux (see “UNIX or Linux sample application source files” on page 8)
- Windows (see “Windows 32-bit sample application” on page 10, or “Windows 64-bit sample application” on page 11)

The dapismp sample application creates its own data streams when backing up or archiving objects. It does not read or write objects to the local disk file system. The object name does not correspond to any file on your workstation. The “seed string” that you issue generates a pattern that can be verified when the object is restored or retrieved. Once you compile the sample application and run **dapismp** to start it, follow the instructions that display on your screen.

OS/400 operating system sample application source files

To build and run a sample OS/400 application you need install the API and ensure you have certain source files. You build the sample application using the QShell or CL program.

Note:
- References to OS/400 include both OS/400 and i5/OS.
- The installation instructions for the OS/400 Tivoli Storage Manager API are included for your convenience in “Installing the API on OS/400 or i5/OS” on page 6. Installation instructions for other operating systems, such as Windows, are included in the *Tivoli Storage Manager Installing and Using the Backup-Archive Client* for your operating system.

The files that are listed in Table 7 include the source files and other files that you need to build the sample application that is included with the API package.

Table 7. Files that you need to build the OS/400 API sample application

File names	Description
README_api_enu	README file

Table 7. Files that you need to build the OS/400 API sample application (continued)

File names	Description
crtapismp.clp makesmp.os400.sh	A CL program file to build dapismp for your application.
dsmrc.h dsmapitd.h dsmapips.h dsmapifp.h release.h	Return codes header file Common type definitions header file Operating system-specific type definitions header file Function prototype header file Release values header file
dapibkup.c dapidata.h dapiinit.c dapint64.h dapint64.c dapipref.c dapiproc.c dapiproc.h dapipw.c dapiqry.c dapirc.c dapismp.c dapitype.h dapiutil.h dapiutil.c	Source code files for a sample application that demonstrates the use of the primary API functions
callmt1.c callmt2.c callevnt.c callhold.c callret.c callbuff.c dpsthread.c	Multi-threaded sample files Event-based retention policy sample source code Deletion hold sample source code Data retention protection sample source code Tivoli Storage Manager buffer sample source code

Installing the API on OS/400 or i5/OS

You can run a command to install the API on OS/400 or i5/OS.

Note:

- The RSTLICPGM command includes language specific arguments. By default, the current user's language based on the primary OS/400 language is loaded. As languages are requested, additional languages are added as secondary languages in OS/400.
- The OS/400 command to uninstall the API product is:

  ```
  DLTLICPGM LICPGM(5733197)
  ```

Depending on your installation source, use one of the following to commands to install the API:

- If you are installing the Tivoli Storage Manager API for OS/400 from the CD, place the CD in the CD-ROM drive and issue the following OS/400 command where *OPT1* is the device name for the CD-ROM drive:

  ```
  RSTLICPGM LICPGM(5733197) DEV(OPT1)
  ```

- If you are installing the Tivoli Storage Manager API for OS/400 from a "save file", issue the following OS/400 command where *MYLIB/MYSAVEFILE* is the name of your save file:

  ```
  RSTLICPGM LICPGM(5733197) DEV(*SAVF) SAVF(MYLIB/MYSAVEFILE) LNG(2924)
  ```

Building the OS/400 sample application from the QShell

Once you install the API and have the required files for the OS/400 sample application, you can compile the dapismp sample application with the OS/400 QShell environment.

In the QShell environment, follow these steps to compile the sample applications and run the dapismp application:

1. To start the QShell environment, issue the command **QSH** from the OS/400 command line.
2. To change your directory to the API sample installation directory issue the following command:
   ```
   cd /usr/tivoli/tsm/client/api/bin/sample
   ```
3. Run the makesmp.os400.sh shell script. This shell script compiles the modules, creates the sample application, and binds it to the API service program. The shell script takes one optional parameter of the library in which you would like the modules and program placed. If you do not enter a library name, the QANSAPI library is used. For example, issue the following the command to compile the sample API modules and create the program into library MYLIB:
   ```
   makesmp.os400.sh mylib
   ```
4. After you build the sample, set up your environment variables, your DSMI_DIR file, and your options files. For more information, see "Understanding configuration and options files" on page 2 and *Tivoli Storage Manager Installing and Using the Backup-Archive Client* for your operating system. You can use the Work with Object Links (WRKLNK) and Edit File (EDTF) commands to edit the options files.
5. To run the sample application issue the following command:
   ```
   dapismp
   ```

   The makesmp.os400.sh script creates a symbolic link from the directory to the dapismp program.
6. Issue the **dapismp** command to run that same application.
7. Choose from the list of options displayed on your screen, making sure to run the sign-on action before you run other actions.

   Note: Always prefix the file space, high-level, and low-level names with the correct path delimiter (/) when you enter them, for example: /myfilespace. This is true even if you specify the asterisk (*) wildcard character.

Building the OS/400 sample application from the command line

Once you install the API and have the required files for the OS/400 sample application, you can compile the dapismp sample application from the OS/400 command line.

Follow these steps to compile and run the dapismp sample application from the OS/400 command-line:

1. Copy the crtapismp.clp CL program source into a source file. For example, if your source file is in library MYLIB, you would issue the following command:
   ```
   CPYFRMSTMF FROMSTMF('/usr/tivoli/tsm/client/api/bin/sample/crtapismp.clp') \
   TOMBR('/qsys.lib/mylib.lib/qclsrc.file/crtapismp.mbr')
   ```
2. To compile the sample application from the command line, MYLIB/QCLSRC(CRTAPISMP), issue the command:

```
CRTCLPGM PGM(MYLIB/CRTAPISMP) SRCFILE(MYLIB/QCLSRC)
```

3. To run the program that compiles the sample application modules, creates the sample application, and binds it to the API service program, issue the command:

   ```
   CALL MYLIB/CRTAPISMP PARM(MYLIB)
   ```

   Note: The library is the only required parameter in which you want to place the modules and program.
4. After you build the sample, set up your environment variables, your DSMI_DIR file, and your options files. For more information, see "Understanding configuration and options files" on page 2 and *Tivoli Storage Manager Installing and Using the Backup-Archive Client* for your operating system. You can use the Work with Object Links (**WRKLNK**) and Edit File (**EDTF**) commands to edit the options files.
5. Use the **ADDENVVAR** command to set the environment variables.
6. To run the dapismp sample application, issue the command:

   ```
   CALL MYLIB/DAPISMP
   ```
7. Choose from the list of options displayed on your screen, making sure to run the sign-on action before you run other actions.

   Note: Always prefix the file space, high-level, and low-level names with the correct path delimiter (/) when you enter them, for example:/myfilespace. This is true even if you specify the asterisk (*) wildcard character.

UNIX or Linux sample application source files

To build and run the sample UNIX or Linux sample application you need to ensure you have certain source files. Once you build the sample application you can compile and run it.

The files that are listed in Table 8 include the source files and other files that you need to build the sample application that is included with the API package.

Table 8. Files that you need to build the UNIX or Linux API sample application

File names		Description
README_api_enu		README file
dsmrc.h dsmapitd.h dsmapips.h dsmapifp.h release.h		Return codes header file Common type definitions header file Operating system-specific type definitions header file Function prototype header file Release values header file
dapibkup.c dapidata.h dapiinit.c dapint64.h dapint64.c dapipref.c dapiproc.c dapiproc.h	dapipw.c dapiqry.c dapirc.c dapismp.c dapitype.h dapiutil.h dapiutil.c	Modules for the command line-driven sample application
makesmp[64].*xxx*		Makefile to build dapismp for your operating system. The *xxx* indicates the operating system.
callmt1.c callmt2.c		Multi-threaded sample files

Table 8. Files that you need to build the UNIX or Linux API sample application (continued)

File names	Description
callmtu1.c callmtu2.c	Multi-threaded Unicode sample files
libApiDS.xx libApiDS64.xx, or libApiTSM64.xx	Shared library (the suffix is platform-dependent)
dsmgrp.c	Grouping sample files
callevnt.c	Event-based retention policy sample source code
callhold.c	Deletion hold sample source code
callret.c	Data retention protection sample source code
callbuff.c	
dpsthread.c	

Building the UNIX or Linux sample application

You build the dapismp sample API application using a compiler for your operating system.

You need the following compilers to build the UNIX or Linux API sample application:

- AIX - IBM Visual Age compiler Version 6 or higher
- HP - aCC compiler A.03.30 or higher
- HP-IA64 - aCC compiler A.05.50 or higher
- Linux - GCC compiler Version 3.3.3 or higher
- Mac OS X - GCC compiler Version 4.0 or higher
- Solaris - Sun Studio C++ compiler Version 11 or higher
- USS - OS/390® C/C++ compiler V2R8 or higher

1. To build the API samples, issue the following command (where *xxx* indicates your operating system):
   ```
   gmake -f makesmp[64].xxx
   ```
2. After you build the samples, set up your environment variables, including the DSMI_DIR, and your options files. For more information, see “Understanding configuration and options files” on page 2 and *Tivoli Storage Manager Installing and Using the Backup-Archive Client* for your operating system.
3. Log on as root the first time for password registration.

   Note: Setting the ***compressalways*** option to ***no*** might not resend an object uncompressed. This behavior depends on the application functionality.

   When specifying the Shared Memory communications method on AIX, the Tivoli Storage Manager API client user must be logged in as root or have the same uid as the process running the Tivoli Storage Manager server. This restriction does not apply if the passwordaccess option is set to generate in the client systems option file (dsm.sys) and the TCA is being used or if you alter your application program file permissions using the following commands:
   ```
   chown root.system your_api_program
   chown u+s your_api_program
   ```

   Check the application program documentation for any recommendations about this.
4. Issue the **dapismp** command to run that same application.

5. Choose from the list of options displayed on your screen, making sure to run the sign-on action before you run other actions.

   Note: Always prefix the file space, high-level, and low-level names with the correct path delimiter (/) when you enter them, for example: /myfilespace. This is true even if you specify the asterisk (*) wildcard character.

Windows 32-bit sample application

To build and run the sample Windows 32–bit application you need install the API and ensure you have certain source files.

Note:

- For Windows applications that were built with V3.1 of the API, replace adsmv3.dll with the new adsmv3.dll and add in tsmapi.dll. For new applications, build with, and use the tsmapi.dll. These DLLs are 32 bit DLLs.
- We recommend that you use dynamic loading. See the implementation in the sample code in dynaload.c.
- The api\obj directory contains the API sample program object files.
- Use the Microsoft C/C++ Compiler Version 15 and the makefile, makesmp.mak, to compile the API sample application, dapismp. It might be necessary to adjust the makefiles to your environment (specifically, the library or the include directories).
- After compiling, run the sample application by issuing the command **dapismp** from the api\samprun directory. The dapismp sample program contains the execution directory.
- Choose from the list of options displayed on your screen, making sure to run the sign-on action before you run other actions.
- Always prefix the file space, high-level, and low-level names with the correct path delimiter (\) when you enter them, for example:\myfilespace. This is true even if you specify the asterisk (*) wildcard character.

For Windows operating systems, the files that are listed in Table 9 include the source files that you need to build the sample application. This sample application is included in the API package. For convenience, a precompiled executable (dapismp.exe) is also included.

Table 9. Files that you need to build the Windows 32–bit API sample application

File names	Description
api.txt	README file
tsmapi.dll adsmv3.dll	API DLLs
dsmrc.h dsmapitd.h dsmapips.h dsmapifp.h dsmapidl.h release.h	Return codes header file Common type definitions header file Operating system-specific type definitions header file Function prototype header file Dynamically loaded function prototype header file Release values header file
dapidata.h dapint64.h dapitype.h dapiutil.h	Source code header files
tsmapi.lib	Implicit library

Table 9. Files that you need to build the Windows 32–bit API sample application (continued)

File names		Description
dapibkup.c dapiinit.c dapint64.c dapipref.c dapiproc.c dapiproc.h	dapipw.c dapiqry.c dapirc.c dapismp.c dapiutil.c dynaload.c	Source code files for dapismp.exe
makesmp.mak		Make file for building sample applications
callmt1.c callmt2.c		Multi-threaded sample file
callmtu1.c callmtu2.c		Multi-threaded Unicode sample files
dsmgrp.c		Grouping sample files.
callevnt.c		Makefile to build dsmgrp for your operating system.
callhold.c		Event-based retention policy sample source code
callret.c		Deletion hold sample source code
callbuff.c		Data retention protection sample source code
dpsthread.c		

Windows 64-bit sample application

To build and run the sample Windows 64-bit application you need install the API and ensure you have certain source files.

Note:

- We recommend that you use dynamic loading. See dynaload.c and implementation in the sample code.
- The API64\OBJ directory contains the API sample program object files. The API64\SAMPRUN directory contains the sample program. The dapismp sample program contains the execution directory.
- The DLL tsmapi64.dll is a 64-bit DLL.
- Use the Microsoft C/C++ Compiler Version 15 and the makefile, makesmp64.mak, to compile the API sample application, dapismp. It might be necessary to adjust the makefiles (specifically, the library or the Include directories) to fit your environment.
- After compiling, run the sample application by issuing the command **dapismp** from the api64\samprun directory.
- Choose from the list of options displayed on your screen, making sure to run the sign-on action before you run other actions.
- Always prefix the file space, high-level, and low-level names with the correct path delimiter (\) when you enter them, for example: \myfilespace. This is true even if you specify the asterisk (*) wildcard character.

For Windows operating systems, the files that are listed in Table 10 include the source files that you need to build the sample application. This sample application is included in the API package. For your convenience, a precompiled executable (dapismp.exe) is also included.

Table 10. Files that you need to build the Windows 64-bit API sample application

File names	Description
api.txt	README file

Table 10. Files that you need to build the Windows 64-bit API sample application (continued)

File names	Description
tsmapi64.dll	API DLLs
dsmrc.h	Return codes header file
dsmapitd.h	Common type definitions header file
dsmapips.h	Operating system-specific type definitions header file
dsmapifp.h	Function prototype header file
dsmapidl.h	Dynamically loaded function prototype header file
release.h	Release values header file
dapidata.h dapint64.h dapitype.h dapiutil.h	Source code header files
tsmapi64.lib	Implicit library
dapibkup.c dapiinit.c dapint64.c dapipref.c dapiproc.c dapiproc.h dapipw.c dapiqry.c dapirc.c dapismp64.c dapiutil.c dynaload.c	Source code files for dapismp.exe
makesmpx64.mak (Windows x64) makesmp64.mak (Windows IA64)	Makefiles to build sample applications
callmt1.c callmt2.c callmtu164.c callmtu264.c	Multithreaded sample files
dpsthread.c	Sample file source code
dsmgrp.c	Grouping sample files.
callevnt.c	Makefile to build dsmgrp for your operating system.
callhold.c	Event-based retention policy sample source code
callret.c	Deletion hold sample source code
callbuff.c	Data retention protection sample source code

Chapter 3. API design recommendations and considerations

Before beginning the design of an application, you need to have a broad understanding of many aspects of the API.

You need to familiarize yourself with the following topics:

- "Determining size limits" on page 16
- "Maintaining API version control" on page 16
- "Using multithreading" on page 18
- "Using signals" on page 18
- "Starting or ending a session" on page 19
- "Object names and IDs" on page 24
- "Using passwordaccess generate without TCA" on page 23
- "Accessing objects as session owner" on page 26
- "Accessing objects across nodes and owners" on page 27
- "Managing file spaces" on page 28
- "Associating objects with management classes" on page 30
- "Expiration/deletion hold and release" on page 32
- "Querying the Tivoli Storage Manager system" on page 35
- "Sending data to a server" on page 37
- "Example flow diagrams for backup and archive" on page 47
- "File grouping" on page 51
- "Putting it all together - a summary diagram" on page 63

When you design your application, consider carefully the items in Table 11 on page 14. Start structures with **memset**. Fields might change with future releases. The *stVersion* value increments with future product enhancements.

Table 11. Design recommendations

Design item	Considerations

Session control	Understand and follow these conditions of session control:
	• The node name should be unique for an application.
	• The owner name should be consistent across a backup and restore procedure.
	• The **passwordaccess** option affects the use of the TCA child process (UNIX and Linux only), node name, session owner name, and password management.
	• Sessions for data movement should end as soon as the task completes so that devices on the server are freed for use by other sessions.
	• To permit LAN-free data transfer, use the **dsmSetup** function call with the ***multithread*** flag set to on.
	• On AIX, when using multithreaded applications or LAN-free, especially when running on machines with multiple CPUs, we strongly recommend setting the environment variable AIXTHREAD_SCOPE to S in the environment before starting the application, for better performance and more solid scheduling. For example:
	`EXPORT AIXTHREAD_SCOPE=S`
	Setting AIXTHREAD_SCOPE to S means that user threads created with default attributes are placed into system-wide contention scope. If a user thread is created with system-wide contention scope, it is bound to a kernel thread and it is scheduled by the kernel. The underlying kernel thread is not shared with any other user thread. See "Using multithreading" on page 18 for more information about this environment variable.
	• Applications that use multiple threads with the same session handle need to synchronize the TSM API calls. No two threads in the same session should call any API function at the same time. For example, use a **mutex** to synchronize API calls:
	getTSMMutex()
	issue TSM API call
	releaseTSMMutex()
	This is only needed when the threads are sharing the same handle, you can use parallel calls to API functions if they are using different session handles.
	• For best performance, it is recommended that your application implements a threaded consumer/producer model for data movement, since the API calls are synchronous and the calls for **dsmGetData function** and **dsmSendData function** block until they are finished. By using a consumer/producer model, the application could be reading the next buffer while waiting for the network. Also this decoupling of data read/write and network increases performance when there is a network bottleneck or disk/application delays. In general:
	`Data thread <----> shared queue of buffers <-----> communication thread (issue calls to the TSM API)`

Table 11. Design recommendations (continued)

Design item	Considerations
Operation sequence	The Tivoli Storage Manager server locks file space database entries during some operations. The following rules apply when designing Tivoli Storage Manager API applications: • Queries lock the file space during the entire transaction. • The query lock can be shared with other query operations, so multiple query operations on the same file space can execute concurrently. • Operations that make Tivoli Storage Manager server database changes (**DB Chg**) are send, get, rename, update, and delete. • Completion of a **DB Chg** operation requires a file space lock during the database change at the end of the transaction. • Multiple **DB Chg** operations on the same file space can execute concurrently. There might be a delay while waiting for the lock at the end transaction. • The query lock cannot be shared with **DB Chg** operations. A **DB Chg** operation delays the beginning of a query on the same file space, so applications should separate and serialize queries from **DB Chg** operations on the same file space.
Object naming convention	When naming objects, consider the following: • A file space is a grouping category for the server. Limit the number of unique file spaces to help performance because several file space queries are performed. • The high-level and low-level object names are the specific object names. If a unique identifier, such as a date stamp, is included in the name, then backup objects are always active. They expire only when they are intentionally marked inactive by the **dsmDeleteObj** function call. • Consider how your application restores objects. This determines how to format the name for easy queries. If you plan to use a partial object restore (POR), you cannot use compression. To suppress compression, use **dsmSendObj objAttr objCompressed=bTrue**.
Object handling	Do not store objectID values to use for future restores. They are not guaranteed to be persistent during the life of the object. During restore, pay special attention to the restore order. After the query, sort on this value before doing the restore. If you are using multiple types of serial media then access them in separate sessions. See "Selecting and sorting objects by restore order" on page 56 for more information.
Management class	Consider how much control the application needs to have over the management class that is associated with its objects. You can define include statements, or you can specify a name on the **dsmSendObj** function call.
Object size	Tivoli Storage Manager needs to know a size estimate for each object. Consider how your application does this. It is better to overestimate the size than to underestimate.

Determining size limits

Certain data structures or fields in the API have size limits. These structures are often names or other text fields that cannot exceed a predetermined length.

Examples of fields with such limits include:
- Application type
- Archive description
- Copy group destination
- Copy group name
- File space information
- Management class name
- Object owner name
- Password

These limits are defined as constants within the header file, dsmapitd.h. Any storage allocation should be based on these constants rather than numbers you enter. Refer to Appendix C, “API type definitions source files,” on page 143 for further information and a list of the current constants.

Maintaining API version control

All APIs have some form of version control, and Tivoli Storage Manager is no exception. The API version that you use in your application must be compatible with the version of the API library that is installed on the end user workstation.

The **dsmQueryApiVersionEx** should be the first API call that you enter when you use the API. This call:
- Confirms that the API library is installed and available on the end user’s system
- Returns the version level of the API library that the application accesses

The API is designed to be upwardly compatible. Applications that are written to older versions or releases of the API library operate correctly when you run a newer version.

Determining the release of the API library is very important because some releases might have different memory requirements and data structure definitions. Downward compatibility is unlikely. See Table 12 for information about your platform.

Table 12. Platform compatibility information

Platform	Description
Windows	The message files must be at the same level as the library (DLL). The Trusted Communication Agent module (dsmtca) is not used.
OS/400	The API service program (QANSAPI/QANSAPI), the Trusted Communication Agent program (QANSAPI/DSMTCA), and the message files must be at the same level.
UNIX or Linux	The API library, the Trusted Communication Agent module (dsmtca), and the message files must be at the same level.

The **dsmQueryApiVersionEx** call returns the version of the API library that is installed on the end user workstation. You can then compare the returned value with the version of the API that the application client is using.

The API version number of the application client is entered in the compiled object code as a set of four constants defined in dsmapitd.h:

```
DSM_API_VERSION
DSM_API_RELEASE
DSM_API_LEVEL
DSM_API_SUB_LEVEL
```

See Appendix C, "API type definitions source files," on page 143.

The API version of the application client should be less than, or equal to, the API library that is installed on the user's system. Be careful about any other condition. You can enter the **dsmQueryApiVersionEx** call at any time, whether the API session has been started or not.

Data structures that the API uses also have version control information in them. Structures have version information as the first field. As enhancements are made to structures, the version number is increased. When initializing the version field, use the defined structure Version value in dsmapitd.h.

Figure 1 demonstrates the type definition of the structure, dsmApiVersionEx from the header file, dsmapitd.h. The example then defines a global variable that is named ***apiLibVer***. It also demonstrates how you can use it in a call to **dsmQueryApiVersionEx** to return the version of the end user's API library. Finally, the returned value is compared to the API version number of the application client.

```
typedef struct
{
        dsUint16_t stVersion;     /* Structure version                */
        dsUint16_t version;       /* API version                      */
        dsUint16_t release;       /* API release                      */
        dsUint16_t level;         /* API level                        */
        dsUint16_t subLevel;      /* API sub level                    */
} dsmApiVersionEx;

dsmApiVersionEx apiLibVer;

memset(&apiLibVer,0x00,sizeof(dsmApiVersionEx));
dsmQueryApiVersionEx(&apiLibVer);

/* check for compatibility problems */
dsInt16_t appVersion= 0, libVersion = 0;
  appVersion=(DSM_API_VERSION * 10000)+(DSM_API_RELEASE * 1000) +
               (DSM_API_LEVEL * 100) + (DSM_API_SUBLEVEL);
  libVersion = (apiLibVer.version * 10000) + (apiLibVer.release * 1000) +
                  (apiLibVer.level * 100) + (apiLibVer.subLevel);
   if (libVersion < appVersion)
   {
      printf("\n***********************************************************\n");
      printf("The TSM API library is lower than the application version\n");
      printf("Install the current library version.\n");
      printf("*************************************************************\n");
      return 0;
   }

printf("* API Library Version = %d.%d.%d.%d  *\n",
     apiLibVer.version,
     apiLibVer.release,
     apiLibVer.level,
     apiLibVer.subLevel);
```

Figure 1. An example of obtaining the version level of the API

Using multithreading

The multithreaded API permits applications to create multiple sessions with the Tivoli Storage Manager server within the same process. The API can be entered again. Any calls can run in parallel from within different threads.

Note: When you run applications that assume a multithreaded API, use the **dsmQueryAPIVersionEx** call.

To run the API in multithreaded mode, set the *mtflag* value to DSM_MULTITHREAD on the **dsmSetUp** call. The **dsmSetUp** call must be the first call after the **dsmQueryAPIVersionEx** call. This call must return before any thread calls the **dsmInitEx** call. When all threads complete processing, enter a call to **dsmCleanUp**. The primary process should not end before all the threads complete processing. See callmt1.c in the sample application.

Restriction: The default for the API is single-thread mode. If an application does not call **dsmSetUp** with the *mtflag* value set to DSM_MULTITHREAD, the API permits only one session for each process.

For UNIX or Linux for versions 3.1.6 through version 4.1.2, you cannot use the Trusted Communication Agent in multithread mode. If you want to set the ***passwordaccess*** option to ***generate***, you must be an -Authorized user. For version 4.2 and beyond, this is no longer true.

Once **dsmSetUp** successfully completes, the application can begin multiple threads and enter multiple **dsmInitEx** calls. Each **dsmInitEx** call returns a handle for that session. Any subsequent calls on that thread for that session must use that handle value. Certain values are process-wide, environmental variables (values that are set on **dsmSetUp**). Each **dsmInitEx** call parses options again. Each thread can run with different options by specifying an overwrite file or an options string on the **dsmInitEx** call. This enables different threads to go to different servers, or use different node names.

Recommendation: On HP, set the thread stack to 64K or greater. The default value of the thread stack (32K) might not be sufficient

To permit application users to have a LAN-free session, use **dsmSetUp** *mtFlag DSM_MULTITHREAD* in your application. This is necessary even if the application is single threaded. This flag activates the threading necessary for the Tivoli Storage Manager LAN-free interface.

Using signals

The application handles signals from the user or the operating system. If the user enters a **CTRL+C**, the application should catch the signal and send **dsmTerminate** calls for each of the active threads. Then, call **dsmCleanUp** to exit. Failure to do this might result in unexpected results on the server if sessions are not closed properly.

Recommendation: The application should install signal handlers, such as SIGPIPE and SIGUSR1, for signals that cause the application to end. The application then receives the return code from the API. For example, to ignore SIGPIPE add: signal(SIGPIPE, SIG_IGN). After adding this information, instead of the application exiting on a broken pipe, the proper return code is returned.

You can use the child process, Trusted Communication Agent (TCA) if the *passwordaccess* option is set to *generate*. When the TCA is used, Tivoli Storage Manager uses the SIGCLD signal. If your application uses the SIGCLD signal, be aware of potential interference from the Tivoli Storage Manager program and how it uses SIGCLD. See "Session security" on page 20 for more information about using the TCA.

Starting or ending a session

The Tivoli Storage Manager program is a session-based program, and all activities must be performed within a Tivoli Storage Manager session. To start a session, the application starts the **dsmInitEx** call. This call must be performed before any other API call other than **dsmQueryApiVersionEx**, **dsmQueryCliOptions**, or **dsmSetUp**.

The **dsmQueryCliOptions** function can only be called before the **dsmInitEx** call. It returns the values of important options, such as option files, compression settings, and communication parameters. The **dsmInitEx** call sets up a session with the server as indicated in the parameters that are passed in the call or defined in the options files.

The client node name, the owner name, and the password parameters are passed to the **dsmInitEx** call. The owner name is case-sensitive, but the node name and password are not. The application client nodes must be registered with the server before starting a session.

Each time an API application client starts a session with the server, the client application type is registered with the server. We recommend that the application type value contain an operating system abbreviation because this value is entered in the "platform" field on the server. The maximum string length is DSM_MAX_PLATFORM_LENGTH.

The **dsmInitEx** function call establishes the Tivoli Storage Manager session with the API configuration file and option list of the application client. The application client can use the API configuration file and option list to set a number of Tivoli Storage Manager options. These values override the values that are set in the user's configuration files at installation time. They cannot change the options that the Tivoli Storage Manager administrator defines. If the application client does not have its own configuration file and option list, you can set both of these parameters to NULL. For more information about configuration files, see "Understanding configuration and options files" on page 2.

The **dsmInitEx** function call establishes the Tivoli Storage Manager session, using additional parameters that permit extended verification.

Check the **dsmInitEx** function call and the **dsmInitExOut** information return code. The Tivoli Storage Manager administrator cancelled the last session if the return code is okay (RC=ok) and the information return code (infoRC) is DSM_RC_REJECT_LASTSESS_CANCELED. To end the current session immediately, call **dsmTerminate**.

The **dsmQuerySessOptions** call returns the same fields as the **dsmQueryCliOptions** call. It can be sent only within a session. The values reflect the client options that are valid during that session, from option files, and from any overrides from the **dsmInitEx** call.

Once a session starts, the application can send a call to **dsmQuerySessInfo** to determine the server parameters that are set for this session. Items such as the policy domain and transaction limits are returned to the application with this call.

End sessions with a **dsmTerminate** call. This closes any connection with the server and frees all resources that are associated with this session.

The example in Figure 2 on page 22 defines a number of global and local variables and then uses them in calls to **dsmInitEx** and **dsmTerminate**. The **dsmInitEx** call takes a pointer to dsmHandle for one of its parameters, while the **dsmTerminate** call takes the dsmHandle itself. The example in Figure 3 on page 22 displays the details of rcApiOut. The function, **rcApiOut**, calls the API function **dsmRCMsg**, that translates a return code into a message. The rcApiOut call then prints the message for the user. A version of **rcApiOut** is included in the API sample application. The **dsmApiVersion** function is a type definition that is found in the header file, dsmapitd.h.

Session security

Tivoli Storage Manager, a session-based system, has security components that permit applications to start sessions in a secure manner. These security measures prohibit unauthorized access to the server and help to insure system integrity.

Every session that is started with the server must complete a sign-on process, requires a password. When the password is coupled with the node name of the client, it insures proper authorization when connecting to the server. The application client provides this password to the API to start the session.

Two methods of password processing are available: *passwordaccess=prompt* or *passwordaccess=generate*. If you use the *passwordaccess=prompt* option, you must include the password value on each **dsmInitEx** call. Or, you can supply the node name and owner name on the **dsmInitEx** call.

Passwords have expiration times associated with them. If a **dsmInitEx** call fails with a password-expired return code (DSM_RC_REJECT_VERIFIER_EXPIRED), the application client must enter the dsmChangePW call using the handle that is returned by **dsmInitEx**. This updates the password before the session can be established successfully. The example in Figure 4 on page 23 demonstrates the procedure to change a password by using **dsmChangePW**. The login owner must be root or Tivoli Storage Manager-Authorized to change the password.

The second method, *passwordaccess=generate*, encrypts and stores the password value in a file. The node name and owner name cannot be supplied on the **dsmInitEx** call, and the system default values are used. This protects the security of the password file. When the password expires, the generate parameter creates a new one and updates the password file automatically.

Note:

1. If two different physical machines have the same Tivoli Storage Manager node name or multiple paths are defined on one node using several server stanzas, *passwordaccess=generate* might only work for the stanza which is used first after password expiration. During the first client-server contact, the user is prompted for the same password for each server stanza separately, and for each stanza, a copy of the password is stored separately. When the password expires, a new password is generated for the stanza which connects the first client-server contact. All subsequent attempts to connect via other server stanzas fail,

because there is no logical link between their respective copies of the old password, and the updated copy generated by the stanza used first after password expiration. In this case, you must update the passwords prior to expiration or after expiration as a recovery from the situation, as follows:

   a. Run **dsmadmc** and update the password on the server.
   b. Run **dsmc -servername=stanza1** and use the new password to generate a proper entry.
   c. Run **dsmc -servername=stanza2** and use the new password to generate a proper entry.

2. For UNIX or Linux: Only the root user or the Tivoli Storage Manager-Authorized user can change the password when using *passwordaccess=prompt*. Only the root user or the Tivoli Storage Manager-Authorized user can start the password file when using *passwordaccess=generate*. You can use the Trusted Communication Agent (TCA) child process for password processing. The application should be aware of this because a child process and the SIGCLD signal are used. The TCA is not used in these situations:
   - The *passwordaccess* option is set to *prompt*.
   - The login user is root.
   - The caller of the function must be a Tivoli Storage Manager-Authorized user.

   Note: The options users and groups are not recognized.

An application can restrict user access by other means, such as setting access filters.

Applications that use multiple IP connections to a single Tivoli Storage Manager server should use the same nodename and Tivoli Storage Manager client password for each session. Follow these steps to enable this support:

1. Define one Tivoli Storage Manager server stanza in the dsm.sys file.
2. For the connections not using the default IP address, specify the option values for *TCPserver* address and *TCPport* on the **dsmInitEx** call.

These values override the IP connection information, but the session still uses the same dsm.sys stanza node and password information.

Note: Nodes in a cluster share a single password.

```
dsmApiVersionEx * apiApplVer;
char           *node;
char           *owner;
char           *pw;
char           *confFile = NULL;
char           *options = NULL;
dsInt16_t      rc = 0;
dsUint32_t     dsmHandle;
dsmInitExIn_t  initIn;
dsmInitExOut_t initOut;
char           *userName;
char           *userNamePswd;

memset(&initIn, 0x00, sizeof(dsmInitExIn_t));
memset(&initOut, 0x00, sizeof(dsmInitExOut_t));
memset(&apiApplVer,0x00,sizeof(dsmapiVersionEx));
apiApplVer.version = DSM_API_VERSION; /* Set the applications compile */
apiApplVer.release = DSM_API_RELEASE;  /* time version.         */
apiApplVer.level   = DSM_API_LEVEL;
apiApplVer.subLevel= DSM_API_SUBLEVEL;

printf("Doing signon for node %s, owner %s, with password %s\n", node,owner,pw);

initIn.stVersion = dsmInitExInVersion;
initIn.dsmApiVersionP = &apiApplVer
initIn.clientNodeNameP = node;
initIn.clientOwnerNameP = owner ;
initIn.clientPasswordP = pw;
initIn.applicationTypeP = "Sample-API AIX";
initIn.configfile = confFile;
initIn.options = options;
initIn.userNameP = userName;
initIn.userPasswordP = userNamePswd;
rc = dsmInitEx(&dsmHandle, &initIn, &initOut);

if (rc == DSM_RC_REJECT_VERIFIER_EXPIRED)
{
   printf("*** Password expired. Select Change Password.\n");
   return(rc);
}
else if (rc)
{
   printf("*** Init failed: ");
   rcApiOut(dsmHandle, rc);   /* Call function to print error message */
   dsmTerminate(dsmHandle);    /* clean up memory blocks */
   return(rc);
}
```

Figure 2. An example of starting and ending a session

```
void rcApiOut (dsUint32_t handle, dsInt16_t rc)
{
     char *msgBuf ;

     if ((msgBuf = (char *)malloc(DSM_MAX_RC_MSG_LENGTH+1)) == NULL)
     {
          printf("Abort:  Not enough memory.\n") ;
          exit(1) ;
      }

      dsmRCMsg(handle, rc, msgBuf);
      printf("
      free(msgBuf) ;
      return;
}
```

Figure 3. Details of rcApiOut

```
printf("Enter your current password:");
gets(current_pw);
printf("Enter your new password:");
gets(new_pw1);
printf("Enter your new password again:");
gets(new_pw2);
/* If new password entries don't match, try again or exit. */
/* If they do match, call dsmChangePW.                      */

rc = dsmChangePW(dsmHandle,current_pw,new_pw1);
if (rc)
{
   printf("*** Password change failed.  Rc =
}
else
{
   printf("*** Your new password has been accepted and updated.\n");
}
return 0;
```

Figure 4. An example of changing a password

Using passwordaccess generate without TCA

The Trusted Communication Agent (TCA) is a child process that normally controls access to the protected password file. It is possible to have the *passwordaccess* option set to *generate* work without starting the TCA by using the TSM-Authorized User (for UNIX and Linux).

Note: For version 3.1.6 through version 4.1.2, when you are running in a multithreaded mode and the ***passwordaccess*** is set to ***generate***, only the root, or TSM-Authorized user, is permitted access. The TCA child process does not start.

Follow these guidelines when setting the *passwordaccess* to *generate* without the TCA:

1. Write the application with a call to **dsmSetUp** which passes *argv[0]*. The *argv[0]* contains the name of the application that calls the API. We permit the application to run as TSM-Authorized; however, the Tivoli Storage Manager administrator should decide on the login name for the TSM-Authorized user.
2. Set the S bit (set the effective user ID) to On for the application executable. The owner of that application executable can then become a TSM-Authorized user and can create a password file, update passwords, and run applications. The owner of the application executable must be the same as the user ID that runs the program. In the following example, *User* is *user1*, the name of the application executable is *applA*, and *user1* has read-write permissions on the /home/user1 directory. The permissions on *applA* are:

   ```
   -rwsr-xr-x user1    group1   applA
   ```

   On OS/400, since there is no S bit, set the application program to run under owner authority so the application owner can become a TSM-Authorized user. To set this, use the USRPRF(*OWNER) option of the CRTPGM (create program) or the CHGPGM (change program) commands.
3. Instruct the users of the application to use the TSM-Authorized name to log in. Tivoli Storage Manager verifies that the login ID matches the application executable owner before it permits access to the protected password file.
4. Set the passworddir option in the dsm.sys file to point to a directory where this user has read-write access. For example, under the server stanza in dsm.sys, you would enter:

   ```
   passworddir /home/user1
   ```

5. Create the password file and ensure that the TSM-Authorized user owns the file.
6. Run *applA* logged on as *user1*.
7. Call dsmSetUp and pass in *argv*.

Creating an administrative user with client owner authority

An administrative user with client owner authority can set parameters on the **dsmInitEx** function call to start sessions. This user can function as an "administrative user" with backup and restore authority for the defined nodes.

To receive client owner authority, issue the following commands:

1. Define the administrative user:
   ```
   REGister Admin admin_name password
   ```
   Where *admin_name* is the administrative user name and *password* is the admin password.
2. Define the authority level. Users with system or policy authority also have client owner authority.
   ```
   Grant Authority admin_name classes authority node
   ```
   Where *admin_name* is the administrative user, *classes* is the node, *authority* is the owner (full backup and restore authority for the node), node (single node) or domain (group of nodes).
3. Define access to a single node.
   ```
   Register Node   node_name  password  userid
   ```
   Where *node_name* is the client user node, *password* is the client user node password, and *userid* is the administrative user name.

When the application uses the administrative user, it calls the **dsmInitEx** function call with the *userName* and *userNamePswd* parameters.

```
dsmInitEx
      clientNodeName = NULL
      clientOwnerName = NULL
      clientPassword = NULL
      userName = 'administrative user' name
      userNamePswd = 'administrative user' password
```

You can set the *passwordaccess* option to *generate* or *prompt*. With either parameter, the *userNamePswd* value starts the session. Once the session starts, any backup or restore process can occur for that node.

Object names and IDs

The Tivoli Storage Manager server is an object storage server whose primary function is to efficiently store and retrieve named objects. The object ID is unique for each object and remains with the object for the life of the object *except* when you use export or import.

To meet this requirement Tivoli Storage Manager has two main storage areas, database and data storage.

- The database contains all metadata, such as the name or attributes associated with objects.

- The data storage contains the object data. The data storage is actually a storage hierarchy that the system administrator defines. Data are efficiently stored and managed on either online or offline media, depending on cost and access needs.

Each object that is stored on the server has a name associated with it. The client controls the following key components of that name:

- File space name
- High-level name
- Low-level name
- Object type

When making decisions about naming objects for an application, you might need to use an external name for the full object names to the end user. Specifically, the end user might need to specify the object in an Include or Exclude statement when the application is run. The exact syntax of the object name in these statements is platform-dependent. On the Windows operating system, the drive letter associated with the file space rather than the file space name itself is used in the Include or Exclude statement. On the OS/400 operating system, the first character of the low-level name must be a forward slash (/).

The object ID value that was assigned when you created the object might not be the same as when you perform a restore process. Applications should save the object name and then query to obtain the current object ID before doing a restore.

File space name

The file space name is one of the most important storage components. It can be the name of a file system, disk drive, or any other high-level qualifier that groups related data together.

Tivoli Storage Manager uses the file space to identify the file system or disk drive on which the data are located. In this way, actions can be performed on all entities within a file space, such as querying all objects within a specified file space. Because the file space is such an important component of the Tivoli Storage Manager naming convention, Tivoli Storage Manager has special calls to register, update, query, and delete file spaces.

The server also has administrative commands to query the file spaces on any node in Tivoli Storage Manager storage, and delete them if necessary. All data stored by the application client must have a file space name associated with it. Select the name carefully to group similar data together in the system.

To avoid possible interference, an application client should select different file space names from those that a backup-archive client would use. The application client should publish its file space names so that end users can identify the objects for include-exclude statements, if necessary.

Note: On Windows platforms, a drive letter is associated with a file space. When you register or update a file space, you must supply the drive letter. Because the include-exclude list refers to the drive letter, you must keep track of each letter and its associated file space. In the sample program dapismp, the drive letter is set to ″G″ by default.

See Chapter 2, “Building and running the sample API application,” on page 5 for more information on the sample programs.

High-level and low-level names

Two other components of the object name are the high-level name qualifier and the low-level name qualifier. The high-level name qualifier is the directory path in which the object belongs, and the low-level name qualifier is the actual name of the object in that directory path.

When the file space name, high-level name, and low-level name are concatenated, they must form a syntactically correct name on the operating system on which the client runs. It is not necessary for the name to exist as an object on the system or resemble the actual data on the local file system. However, the name must meet the standard naming rules to be properly processed by the **dsmBindMC** calls. See "Understanding backup and archive objects" on page 40 for naming considerations that are related to policy management.

Object type

The object type identifies the object as either a file or a directory. A file is an object that contains both attributes and binary data, and a directory is an object that contains only attributes.

Table 13 shows what the application client would code is for object names by platform.

Table 13. Application object name examples by platform

Platform	**Client code for object name**
UNIX or Linux	/myfs/highlev/lowlev
Windows	"myvol\\highlev\\lowlev" **Note:** On a Windows platform, a double backslash translates into a single backslash, because a backslash is the escape character. File space names start with a slash on the UNIX or Linux platform, but do not start with a slash on the Windows platform.
OS/400	myfs/highlev/lowlev

Accessing objects as session owner

Each object has an owner name associated with it. The rules determining what objects are accessed depend on what owner name is used when a session is started. Use this session owner value to control access to the object.

The session owner is set during the call to **dsmInitEx** in the *clientOwnerNameP* parameter. If you start a session with **dsmInitEx** owner name of *NULL* and you use *passwordaccess=prompt*, that session owner is handled with session (root or TSM-Authorized) authority. This is also true if you log in with a root ID or TSM authorized ID and you use *passwordaccess= generate*. This session can perform any action on any object that is owned by this node regardless of the actual owner of that object.

If a session is started with a specific owner name, the session can only perform actions on objects that have that object owner name associated with them. Backups or archives into the system all must have this owner name associated with them. Any queries performed return only the values that have this owner name associated with them. The object owner value is set during the **dsmSendObj** call in the **Owner** field of the **ObjAttr** structure. An owner name is case-sensitive. Table 14 on page 27 summarizes the conditions under which a user has access to an object.

Table 14. Summary of user access to objects

Session owner	Object owner	User access
NULL (root, system owner)	“ ” (empty string)	Yes
NULL	Specific name	Yes
Specific name	“ ” (empty string)	No
Specific name	Same name	Yes
Specific name	Different name	No

Accessing objects across nodes and owners

Three function calls support cross-node, cross-owner access on the same platform: **dsmSetAccess**, **dsmDeleteAccess**, and **dsmQueryAccess**. These functions, along with the *-fromnode* and *-fromowner* string options that are passed on **dsmInitEx**, permit a complete cross-node query, restore and retrieve process through the API.

For example, User A on node A uses the **dsmSetAccess** function call to give access to its backups under the /db file space to User B from Node B. The access rule is displayed as:

ID	Type	Node	User	Path
1	Backup	Node B	User B	`/db/*/*`

When User B logs on at Node B, the option string to **dsmInitEx** is:

```
-fromnode=nodeA -fromowner=userA
```

These options are set for this session. Any queries access the file spaces, and files of Node A. Backups and archives are not permitted. Only query, restore, and retrieve processes are permitted from the file spaces for which User B has access. If the application tries to execute any operation using a **dsmBeginTxn** (for examples, backup or update) while signed in with a *-fromnode* or *-fromowner* option set, then the dsmBeginTxn fails with the return code DSM_RC_ABORT_NODE_NOT_AUTHORIZED. See the individual function calls and “dsmInitEx” on page 99 for more information.

Note: On UNIX and Linux you can specify *–fromowner=root* in the option string that is passed on the **dsmInitEx** function call. This permits non-root users access to files that the root owns if a set access was performed.

Use the *asnodename* option on the **dsmInitEx** option string with the appropriate function to back up, archive, restore, retrieve, query or delete data under the target node name on the Tivoli Storage Manager server. See “Backing up multiple nodes with client node proxy support” on page 67 for information on enabling this option.

Managing file spaces

Because the file space is so important to the operation of the system, a separate set of calls is used to register, update, and delete file space identifiers. Before you can store any objects that are associated with a file space on the system, you must first register the file space with Tivoli Storage Manager.

Use the **dsmRegisterFS** call to accomplish this task. See "Object names and IDs" on page 24 for more information.

The file space identifier is the top-level qualifier in a three-part name hierarchy. Grouping related data together within a file space makes management of that data much easier. For example, either the application client or the Tivoli Storage Manager server administrator can delete a file space and all the objects within that file space.

File spaces also permit the application client to provide information about the file space to the server that the Tivoli Storage Manager administrator can then query. This information is returned on the query in the **qryRespFSData** structure and includes:

Type	Definition
fstype	The file space type. This field is a character string that the application client sets.
fsAttr[platform].fsInfo	A client information field used for client-specific data.
capacity	The total amount of space in the file space.
occupancy	The amount of space currently occupied in the file space.
backStartDate	The time stamp when the latest backup started (set by sending a **dsmUpdateFS** call).
backCompleteDate	The time stamp when the latest backup completed (set by sending a **dsmUpdateFS** call).

Using capacity and occupancy depends on the application client. Some applications might not need information about the size of the file space, in which case these fields can default to *0*. See "Querying the Tivoli Storage Manager system" on page 35 for more information about querying file spaces.

After a file space is registered with the system, you can back up or archive objects at any time. We recommend that you call **dsmUpdateFS** to update the occupancy and the capacity fields of the file space after a backup or archive operation. This ensures that the values for the occupancy and capacity of the file system are current. You can also update the **fsinfo**, **backupstart**, and **backupcomplete** fields.

If you want to monitor your last backup dates, enter a **dsmUpdateFS** call before starting your backup. Set the update action to DSM_FSUPD_BACKSTARTDATE. This forces the server to set the **backStartDate** field of the file space to the current time. After the backup is complete for that file space, enter a **dsmUpdateFS** call with the update action that is set to DSM_FSUPD_BACKCOMPLETEDATE. This creates a time stamp on the end of the backup.

If a file space is no longer needed, you can delete it with the **dsmDeleteFS** command. On the UNIX or Linux platform, only the root user or TSM-Authorized user can delete file spaces.

The examples in Figure 5 demonstrate how to use the three file space calls for UNIX or Linux. For an example of how to use the three file space calls for Windows, see the sample program code installed on your system.

```
/*  Register the file space if it has not already been done.  */

dsInt16        rc;
regFSData    fsData;
char         fsName[DSM_MAX_FSNAME_LENGTH];
char         smpAPI[] = "Sample-API";

strcpy(fsName,"/home/tallan/text");
memset(&fsData,0x00,sizeof(fsData));
fsData.stVersion = regFSDataVersion;
fsData.fsName = fsName;
fsData.fsType = smpAPI;
strcpy(fsData.fsAttr.unixFSAttr.fsInfo,"Sample API FS Info");
fsData.fsAttr.unixFSAttr.fsInfoLength =
       strlen(fsData.fsAttr.unixFSAttr.fsInfo) + 1;
fsData.occupancy.hi=0;
fsData.occupancy.lo=100;
fsData.capacity.hi=0;
fsData.capacity.lo=300;

rc = dsmRegisterFS(dsmHandle,fsData);
if (rc == DSM_RC_FS_ALREADY_REGED) rc = DSM_RC_OK;  /* already done */
if (rc)
{
   printf("Filespace registration failed: ");
   rcApiOut(dsmHandle, rc);
   free(bkup_buff);
   return (RC_SESSION_FAILED);
}
```

Figure 5. An example of working with file spaces, Part 1

```
/* Update the file space. */

dsmFSUpd     updFilespace;         /* for update FS */

updFilespace.stVersion = dsmFSUpdVersion;
updFilespace.fsType = 0;             /* no change */
updFilespace.occupancy.hi = 0;
updFilespace.occupancy.lo = 50;
updFilespace.capacity.hi = 0;
updFilespace.capacity.lo = 200;
strcpy(updFilespace.fsAttr.unixFSAttr.fsInfo,
       "My update for filespace") ;
updFilespace.fsAttr.unixFSAttr.fsInfoLength =
       strlen(updFilespace.fsAttr.unixFSAttr.fsInfo);

updAction = DSM_FSUPD_FSINFO |
            DSM_FSUPD_OCCUPANCY |
            DSM_FSUPD_CAPACITY;

rc = dsmUpdateFS (handle,fsName,&updFilespace,updAction);
printf("dsmUpdateFS rc=%d\n", rc);
```

Figure 6. An example of working with file spaces, Part 2

```
/*  Delete the file space.  */

printf("\nDeleting file space
rc = dsmDeleteFS (dsmHandle,fsName,DSM_REPOS_ALL);
if (rc)
{
   printf("  FAILED!!! ");
   rcApiOut(dsmHandle, rc);
}
else printf("  OK!\n");
```

Figure 7. An example of working with file spaces, Part 3

Associating objects with management classes

A primary feature of Tivoli Storage Manager is the use of policies (management classes) to define how objects are stored and managed in Tivoli Storage Manager storage. An object is associated with a management class when the object is backed up or archived.

This management class determines:
- How many versions of the object are kept if backed up
- How long to keep archive copies
- Where to insert the object in the storage hierarchy on the server

Management classes consist of both backup copy groups and archive copy groups. A copy group is a set of attributes that define the management policies for an object that is being backed up or archived. If a backup operation is being performed, the attributes in the backup copy group apply. If an archive operation is being performed, the attributes in the archive copy group apply.

The backup or archive copy group in a particular management class can be empty or NULL. If an object is bound to the NULL backup copy group, that object cannot be backed up. If an object is bound to the NULL archive copy group, the object cannot be archived.

Because the use of policy is a very important component of Tivoli Storage Manager, the API requires that all objects sent to the server are first assigned a management class by using the **dsmBindMC** call. The Tivoli Storage Manager product supports using an include-exclude list to affect management class binding. The **dsmBindMC** call uses the current Include-Exclude list to perform management class binding.

Include statements can associate a specific management class with a backup or archive object. Exclude statements can prevent objects from being backed up but not from being archived. For more information, see *Tivoli Storage Manager Installation and Using Guide* for your operating system.

The API requires that **dsmBindMC** is called before you back up or archive an object. The **dsmBindMC** call returns a mcBindKey structure that contains information on management class and copy groups that are associated with the object. Check the copy group destination before proceeding with a send. When you send multiple objects in a single transaction, they must have the same copy group destination. The **dsmBindMC** function call returns the following information:

Table 15. Information returned on the dsmBindMC call

Information	Description
Management Class	The name of the management class that was bound to the object. The application client can send the **dsmBeginQuery** call to determine all attributes of this management class.
Backup Copy Group	Informs you if a backup copy group exists for this management class. If a backup operation is being performed and a backup copy group does not exist, this object cannot be sent to Tivoli Storage Manager storage. You receive an error code if you attempted to send it using the **dsmSendObj** call.
Backup Copy Destination	This field identifies the Tivoli Storage Manager storage pool to which the data is sent. If you are performing a multiple object backup transaction, all copy destinations within that transaction must be the same. If an object has a different copy destination than previous objects in the transaction, end the current transaction and begin a new transaction before you can send the object. You receive an error code if you attempt to send objects to different copy destinations within the same transaction.
Archive Copy Group	Informs you if an archive copy group exists for this management class. If an archive operation is being performed and an archive copy group does not exist, this object cannot be sent to Tivoli Storage Manager storage. You receive an error code if you attempted to send it using the **dsmSendObj** call.
Archive Copy Destination	This field identifies the Tivoli Storage Manager storage pool to which the data are sent. If you are performing a multiple object archive transaction, all copy destinations within that transaction must be the same. If an object has a different copy destination than previous objects in the transaction, end the current transaction and begin a new transaction before you send the object. You receive an error code if you attempt to send objects to different copy destinations within the same transaction.

Backup copies of an object can be rebound to a different management class if a subsequent back up with the same object name is done that uses a management class different than the original. For example, if you back up ObjectA and bind it to Mgmtclass1, and later you back up ObjectA and bind it to Mgmtclass2, the most current backup rebinds any inactive copies to Mgmtclass2. The parameters defined in Mgmtclass2 would now control all copies. However the data does not move if the destination is different.

You can also rebind backup copies to a different management class using the **dsmUpdateObj** or **dsmUpdateObjEx** call with the DSM_BACKUPD_MC action.

Query management classes

Applications can query management classes to determine what management classes are possible for a given node and to determine what the attributes are within the management class.

You can only bind objects to management classes by using the **dsmBindMC** call. You might want your applications to query the management class attributes and display them to end users. See “Querying the Tivoli Storage Manager system” on page 35 for more information.

In the example in Figure 8 on page 32, a switch statement is used to distinguish between backup and archive operations when calling **dsmBindMC**. The information returned from this call is stored in the **MCBindKey** structure.

```
dsUint16_t     send_type;
dsUint32_t     dsmHandle;
dsmObjName   objName;     /* structure containing the object name  */
mcBindKey    MCBindKey;   /* management class information          */
char         *dest;       /* save destination value                */

switch (send_type)
{
   case (Backup_Send) :
      rc = dsmBindMC(dsmHandle,&objName,stBackup,&MCBindKey);
      dest = MCBindKey.backup_copy_dest;
      break;
   case (Archive_Send) :
      rc = dsmBindMC(dsmHandle,&objName,stArchive,&MCBindKey);
      dest = MCBindKey.archive_copy_dest;
      break;
   default : ;
}

if (rc)
{
   printf("*** dsmBindMC failed: ");
   rcApiOut(dsmHandle, rc);
   rc = (RC_SESSION_FAILED);
   return;
}
```

Figure 8. An example of associating a management class with an object

Expiration/deletion hold and release

You can suspend (hold) deletion and expiration of specific archive objects in response to a pending or ongoing action that requires that particular data be held. In the event an action is initiated that might require access to data, that data must be available until the action is concluded and access to the data is no longer required as part of that process. Upon determination the suspension is no longer required (released), normal deletion and expiration timing resumes per the original retention period.

Prerequisites:

Verify the server license by issuing a test **dsmRetentionEvent** call:
1. Query for one object you want to hold and get the ID.
2. Issue the **dsmBeginTxn**, **dsmRetentionEvent** with *Hold*, and **dsmEndTxn**.
3. If the server is not licensed, you receive a vote of abort with reason code DSM_RC_ABORT_LICENSE_VIOLATION.

Note:
1. You cannot issue more than one **dsmRetentionEvent** call in a single transaction.
2. You cannot issue a hold on an object that is already under hold.

To hold objects follow these steps:
1. Query the server for all the objects that you want to place under hold. Get the object ID for each object.
2. Issue a **dsmBeginTxn** call, then issue a **dsmRetentionEvent** call with the list of objects, followed by a **dsmEventType**: *eventHoldObj* call. If the number of objects exceeds the value of *maxObjPerTxn*, use multiple transactions.
3. Use the qryRespArchiveData response on the **dsmGetNextQObj** function call to confirm if the objects were put under hold (look at the value of objHeld in qryRespArchiveData).

To release objects from hold:

1. Query the server for all the objects that you want to release from hold. Get the object ID for each object.
2. Issue a **dsmBeginTxn** call, then issue a **dsmRetentionEvent** call with the list of objects, followed by a **dsmEventType**: *eventReleaseObj* call. If the number of objects exceeds the value of maxObjPerTxn, use multiple transactions.
3. Use the **qryRespArchiveData** response on the *dsmGetNextQObj* function call to confirm if the objects were released from hold (look at the value of *objHeld* in **qryRespArchiveData**).

Archive data retention protection

Tivoli Storage Manager currently prevents the modification of data under Tivoli Storage Manager control and the deletion of archive objects by unauthorized agents (individual or program). This protection extends to preventing the deletion of data by any agent prior to the expiration of the retention period.

This helps assure that no individual or program can maliciously or accidentally delete data under Tivoli Storage Manager control. An archive object that is sent to an archive retention protection server is protected from accidental deletes and has its retention period enforced.

To set up archive data retention protection, perform the following steps:

1. On a new server installation (no previous data) issue the **SET ARCHIVERETENTIONPROTECTION ON** command.
2. In the API option string on the **dsmInit** or **dsmInitEx** function calls, enter:

   ```
   -ENABLEARCHIVERETENTIONPROTECTION=yes
   ```

   You can also set the *enablearchiveretentionprotection* option in your dsm.opt file on non-UNIX systems or in your dsm.sys file on UNIX systems:

   ```
   SERVERNAME srvr1.ret
      TCPPORT                          1500
      TCPSERVERADDRESS                 node.domain.company.com
      COMMMETHOD                       TCPIP
      ENABLEARCHIVERETENTIONPROTECTION YES
   ```

   See “The enablearchiveretentionprotection option” on page 34 for more information about this option.
3. Issue a query to the server to confirm if the Tivoli Storage Manager server is enabled for archive retention protection. Check the value of the archiveRetentionProtection field in the dsmQuerySessInfo structure.

Note:

- Only archive operations are allowed on a retention protection server.
- Any object that is not bound explicitly to a management class through a value in the **dsmBindMc** function call or through include-exclude statements is bound to the explicit name of the default management class. For example, if the default management class in the node’s policy is MC1, after a bind that would have resulted in a bind to DEFAULT, the object is instead bound explicitly to MC1. On a query response, the object displays as bound to MC1.
- After you enable archive data retention protection, any attempt to delete an object before it is due to expire returns the abort code DSM_RC_ABORT_DELETE_NOT_ALLOWED on the end transaction.

- See the appropriate Tivoli Storage Manager server Administrator's Reference for setting retention protection for an archive object.

The enablearchiveretentionprotection option

The enablearchiveretentionprotection option specifies whether to enable data retention protection for archive objects on a Tivoli Storage Manager server dedicated for this purpose. Your Tivoli Storage Manager server administrator must activate data retention protection on a new Tivoli Storage Manager server that does not already have stored objects (backup, archive, or space-managed). If the API application attempts to store a backup version or space-managed object on the server, an error message is issued.

Note: This option is valid for API applications only. The backup-archive client does not support data retention protection for archive objects to a Tivoli Storage Manager retention protection server.

The note in Chapter 3, "API design recommendations and considerations," on page 13 that states ″Do not store objectID values to use for future restores. They are not guaranteed to be persistent during the life of the object.″ can be relaxed for Archive manager applications since the archive-manager server does not support export or import. Archive-manager applications can save and use the objectID to improve the performance during object restore.

If the Tivoli Storage Manager server issues the SET ARCHIVERETENTIONPROTECTION ON command, you cannot delete an archived object from the server using the **delete filespace** command, until the policy parameters of the archive copy group are satisfied. See the appropriate Tivoli Storage Manager server Administrator's Reference for information on how to set up a management class.

Event-based retention policy

Event-based retention policy provides a mechanism whereby retention time for an archive object can be initiated by the occurrence of a business event, such as the closure of a bank account. This more closely aligns Tivoli Storage Manager's data retention policy with business requirements for data. When the event occurs it is expected that the application sends an **eventRetentionActivate** event on that object to the server to initiate the retention.

To use an event-based retention policy, perform the following steps:

1. On the server, create a management class with an archive **copygroup** of type EVENT (See the appropriate Tivoli Storage Manager server Administrator's Reference).
2. Query the management class to confirm if it is event-based (the **retainInit** field in the **archDetailCG** structure should be ARCH_RETINIT_EVENT).
3. Bind the objects to the event-based management class using include, **archmc**, or explicitly through the **mcNameP** attribute in the **ObjAttr** structure on the **dsmSendObj** function call.
4. At the point that you want to start the retention for the object(s) (i.e., the event has occurred), query the server for all the objects affected, check to see if they are in a PENDING state, and get the object IDs. The **qryRespArchiveData** structure **retentionInitiated** field should indicate DSM_ARCH_RETINIT_PENDING
5. Issue a **dsmBeginTxn** call, then issue a **dsmRetentionEvent** call with the list of objects, followed by a **dsmEventType**:*eventRetentionActivate* call. If the number of objects exceeds the value of *maxObjPerTxn*, use multiple transactions.

6. Query the objects to confirm if the retention has been activated. The **qryRespArchiveData** structure, the **retentionInitiated** field should have the value of DSM_ARCH_RETINIT_STARTED.

Note: Only one **dsmRetentionEvent** call is allowed per transaction.

Querying the Tivoli Storage Manager system

The API has several queries, such as management class query, that applications can use.

All queries that use **dsmBeginQuery** call follow these steps:

1. Send the **dsmBeginQuery** call with the appropriate query type:
   - Backup
   - Archive
   - Active backed-up objects
   - File space
   - Management class

   The **dsmBeginQuery** call informs the API of the data format being returned from the server. The appropriate fields can be placed in the data structures that are passed by the **dsmGetNextQObj** calls. The begin query call also permits the application client to set the scope of the query by properly specifying the parameters on the begin query call.

   Note: On the UNIX or Linux platform, only the root user can query active backed-up objects (also known as ″fast path″).
2. Enter the **dsmGetNextQObj** call to obtain each record from the query. This call passes a buffer that is large enough to hold the data that is returned from the query. Each query type has a corresponding data structure for the data returned. For example, a backup query type has an associated **qryRespBackupData** structure that is filled in when the **dsmGetNextQObj** call is sent.
3. The **dsmGetNextQObj** call usually returns one of the following codes:
   - DSM_RC_MORE_DATA. Send the **dsmGetNextQObj** call again.
   - DSM_RC_FINISHED. There is no more data. Send the **dsmEndQuery** call.
4. Send the **dsmEndQuery** call. When all query data are retrieved or additional query data are not needed, enter the **dsmEndQuery** call to end the query process. This causes the API to flush any remaining data from the query stream and release any resources that were used for the query.

Figure 9 on page 36 displays the state diagram for performing query operations.

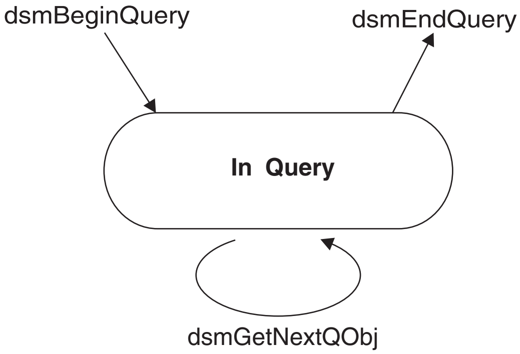


Figure 9. State diagram for general queries

Figure 10 displays the flowchart for performing query operations.

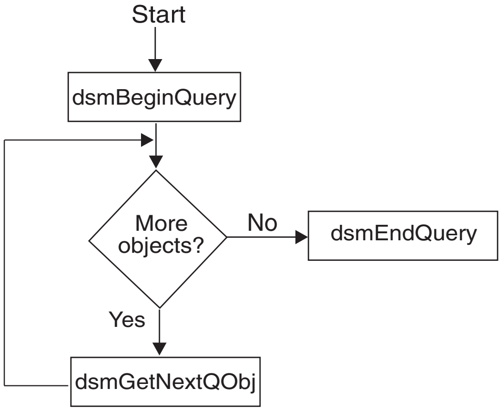


Figure 10. Flowchart for general queries

Example of querying the system

In this example a management class query prints out the values of all the fields in the backup and archive copy groups for a particular management class.

```
dsInt16                rc;
qryMCData            qMCData;
DataBlk              qData;
qryRespMCDetailData  qRespMCData, *mcResp;
char                 *mc, *s;
dsBool_t               done = bFalse;
dsUint32_t               qry_item;

/* Fill in the qMCData structure with the query criteria we want */
qMCData.stVersion = qryMCDataVersion;   /* structure version     */
qMCData.mcName    = mc;                 /* management class name */
qMCData.mcDetail  = bTrue;              /* want full details?    */

/* Set parameters of the data block used to get or send data     */
qData.stVersion = DataBlkVersion;
qData.bufferLen = sizeof(qryRespMCDetailData);
qData.bufferPtr = (char *)&qRespMCData;

qRespMCData.stVersion = qryRespMCDetailDataVersion;
```

```
if ((rc = dsmBeginQuery(dsmHandle,qtMC,(dsmQueryBuff *)&qMCData)))
{
   printf("*** dsmBeginQuery failed: ");
   rcApiOut(dsmHandle, rc);
   rc = (RC_SESSION_FAILED);
}
else
{
   done = bFalse;
   qry_item = 0;
   while (!done)
   {
      rc = dsmGetNextQObj(dsmHandle,&qData);
      if ((   (rc == DSM_RC_MORE_DATA)
           || (rc == DSM_RC_FINISHED))
          && qData.numBytes)
      {
         qry_item++;
         mcResp = (qryRespMCDetailData *)qData.bufferPtr;
         printf("Mgmt. Class
         printf("             Name:
         printf("   Backup CG Name:
            .
            .  /* other fields of backup and archive copy groups */
            .
         printf("   Copy Destination:
      }
      else
      {
         done = bTrue;
         if (rc != DSM_RC_FINISHED)
         {
            printf("*** dsmGetNextQObj failed: ");
            rcApiOut(dsmHandle, rc);
         }
      }
      if (rc == DSM_RC_FINISHED) done = bTrue;
   }
   rc = dsmendQuery (dsmHandle);
}
```

Figure 11. An example of performing a system query

Sending data to a server

The API permits application clients to send data or named objects and their associated data to Tivoli Storage Manager server storage.

Note: You can either back up or archive data. Perform all send operations within a transaction.

The transaction model

All data sent to Tivoli Storage Manager storage during a backup or archive operation is done within a transaction. A transaction model provides a high level of data integrity for the Tivoli Storage Manager product, but it does impose some restrictions that an application client must take into consideration.

Start a transaction by a call to **dsmBeginTxn** or end a transaction by a call to **dsmEndTxn**. A single transaction is an atomic action. Data sent within the boundaries of a transaction is either committed to the system at the end of the transaction or rolled back if the transaction ends prematurely.

Transactions can consist of either single object sends or multiple object sends. To improve system performance by decreasing system overhead, send smaller objects in a multiple object transaction. The application client determines whether single or multiple transactions are appropriate.

Send all objects within a multiple object transaction to the same copy destination. If you need to send an object to a different destination than the previous object, end the current transaction and start a new one. Within the new transaction, you can send the object to the new copy destination.

Note: Objects that do not contain any bit data (*sizeEstimate=0*) are not checked for copy destination consistency.

Tivoli Storage Manager limits the number of objects that can be sent in a multiple object transaction. To find this limit, call **dsmQuerySessInfo** and examine the **maxObjPerTxn** field. This field displays the value of the *TXNGroupmax* option that is set on your server.

The application client must keep track of the objects sent within a transaction to perform retry processing or error processing if the transaction ends prematurely. Either the server or the client can stop a transaction at any time. The application client must be prepared to handle sudden transaction ends that it did not start.

File aggregation

Tivoli Storage Manager servers use a function that is called file aggregation. With file aggregation, all objects sent in a single transaction are stored together, which saves space and improves performance. You can still query and restore the objects separately.

To use this function, all of the objects in a transaction should have the same file space name. If the file space name changes within a transaction, the server closes the existing aggregated object and begins a new one.

LAN-free data transfer

The API can take advantage of LAN-free data transfer if the **dsmSetUp** option for multithreading is ON. The API returns the existence of a LAN-free destination in the **Query Mgmt Class** response structure **archDetailCG** or **backupDetailCG** field **bLanFreeDest**.

You can use LAN-free operations on platforms that are supported by the storage agent. Excluded platforms are Macintosh, OS/400, and zOS/USS.

LAN-free information is provided in the following output structures. The out structure (**dsmEndGetDataExOut_t**) for **dsmEndGetData** includes the field, **totalLFBytesRecv**. This is the total number of LAN-free bytes that are received. The out structure (**dsmEndSendObjExOut_t**) for **dsmEndSendObjEx** includes the field, **totalLFBytesSent**. This is the total number of LAN-free bytes that were sent.

For more information about LAN-free data transfer, see the *IBM Tivoli Storage Manager for Storage Area Networks User's Guide*.

Simultaneous write

Tivoli Storage Manager server storage pools can be configured to write to a primary storage pool and copy storage pool(s) simultaneously during a backup or archive. This creates multiple copies of the object.

Note: For information on setting up simultaneous write, refer to *Tivoli Storage Manager Administrator's Guide*.

If a simultaneous write operation fails, the return code on the **dsmEndTxn** function might be DSM_RC_ABORT_STGPOOL_COPY_CONT_NO indicating that the write to one of the copy storage pools failed, and the Tivoli Storage Manager storage pool option COPYCONTINUE was set to NO. The application should terminate because the problem needs to be resolved by the Tivoli Storage Manager server administrator.

API performance considerations

You can use the *tcpbuffsize* and *tcpnodelay* client options and the **DataBlk** API parameter to enhance API performance.

Table 16. Backup-archive options and the API parameter that enhance performance

Backup-archive client options	**Recommendation**
tcpbuffsize	We recommend setting this to 32 KB. The default is 31 KB.
tcpnodelay	This option is valid for Windows and AIX only. Specifies whether to send small buffers to the server rather than holding them. We recommend setting this option to *yes* for all platforms.
API parameter	**Recommendation**
DataBlk	This parameter is used with the **dsmSendData** function call to determine the application buffer size. It should be a multiple of the *tcpbuffsize* minus 4 bytes (for example, 32 KB - 4) specified with the *tcpbuffsize* option.

Each **dsmSendData** call is synchronous and does return until the data transferred to the API in the *dataBlkPtr* is flushed to the network. The API adds a 4 byte overhead to each transaction buffer that is placed on the network.

For example, when the transaction buffer size is 32 KB and the application *DataBlk* buffer size is 31 KB, then each application *DataBlk* buffer fits in a communications buffer and be flushed immediately. However, if the application *DataBlk* buffer is exactly 32 KB, and since the API is adding a 4 byte overhead per transaction buffer, there are 2 flushes: one of 32 KB and one of 4 bytes. Also, if you set the *tcpnodelay* option to *no*, there could be a delay of up to 200 milliseconds until the 4 bytes are flushed.

Sending objects to the server

Application clients can send data or named objects and their associated data to Tivoli Storage Manager storage by using the API backup and archive functions. The backup and archive components of the system permit use of different management procedures for data that is sent to Tivoli Storage Manager storage.

The size estimate attribute is an estimate of the total size of the data object to send to the server. If the application does not know the exact object size, set the *sizeEstimate* to a higher estimate. If the estimate is smaller than the actual size, the Tivoli Storage Manager server uses extra resources to manage extra space allocations.

Important:

- It is important that you try to be as accurate as possible on this size estimate, because the Tivoli Storage Manager server uses this attribute for efficient space allocation and object placement within its storage resources.
- If the estimate is smaller than the actual size, a Tivoli Storage Manager server with caching does not allocate extra space and stops the send.

You might encounter problems if the *sizeEstimate* is much too large. The Tivoli Storage Manager server might not have enough space for the estimated size but does have space for the actual size; or the server might use slower devices.

You can back up or archive objects that are larger than two gigabytes in size. The objects can be either compressed or uncompressed.

To start a send operation, call **dsmSendObj**. If you have more data than you can send at one time, you can make repeated calls to **dsmSendData** to transfer the remainder of the information. Call **dsmEndSendObj** to complete the send operation.

Understanding backup and archive objects

The backup component of the Tivoli Storage Manager system supports several versions of named objects that are stored on the server.

Any object backed up to the server that has the same name as an object that is already stored on the server from that client is subject to version control. Objects are considered to be in active or inactive states on the server. The latest copy of an object on the server that has not been deactivated is in the active state. Any other object with the same name, whether it is an older version or a deactivated copy, is considered inactive. Management class constructs define different management criteria. They are assigned to active and inactive objects on the server.

Table 17 on page 41 lists the copy group fields that apply to active and inactive states:

Table 17. Backup copy group fields

Field	Description
VEREXISTS	The number of inactive versions if active versions exist.
VERDELETED	The number of inactive versions if active versions do not exist.
RETEXTRA	The number of days to keep inactive versions.
RETONLY	The number of days to keep the last inactive versions if active versions do not exist.

If backup versions each have a unique name, such as using a time stamp in the name, then versioning does not happen automatically: every object is active. Active objects never expire, so an application would be responsible for deactivating these with the **dsmDeleteObj** call. In this situation, the application would need the deactivated objects to expire as soon as possible. The user would define a backup copy group with VERDELETED=0 and RETONLY=0.

The archive component of the Tivoli Storage Manager system permits objects to be stored on the server with retention or expiration period controls instead of version control. Each object stored is unique, even though its name might be the same as an object already archived. Archive objects have a description field associated with the metadata that can be used during query to identify a specific object.

Every object on a Tivoli Storage Manager server is assigned a unique object ID. The persistence of the original value is not guaranteed during the life of an object (specifically, after an export or import). Therefore, an application should not query and save the original object ID for use on later restores. Rather, an application should save the object name and insert date. You can use this information during a restore to query objects and verify the insert date. Then, the current object ID can be used to restore the object.

Compression

The end user's configuration, along with the **dsmSendObj** *objCompressed* option, determines whether Tivoli Storage Manager compresses the object during a send. Also, objects with a *sizeEstimate* less than DSM_MIN_COMPRESS_SIZE are never compressed.

If the object is compressed already (*objCompressed=bTrue*), it is not compressed again. If it is not compressed, Tivoli Storage Manager decides whether to compress the object, based on the values of the compression option that is set by the Tivoli Storage Manager administrator and that is set in the API configuration sources.

The Tivoli Storage Manager server administrator can affect compression behavior with the register node command (*compression=yes*, *no*, or *client-determined*). If this is *client-determined*, then the compression behavior is determined by the compression option value in the configuration sources.

Some types of data, such as data that is already compressed, might actually get bigger when processed with the compression algorithm. When this happens, the return code DSM_RC_COMPRESS_GREW is generated. If you realize that this might happen, but you want the send operation to continue anyway, tell the end users to specify the following option in their options file:

```
COMPRESSAlways Yes
```

If, during **dsmSendData** with compression you get DSM_RC_COMPRESS_GREW, you might want to start over and send the object again without compression. To enforce this, set the **dsmSendObj** *ObjAttr.objCompressed* to *bTrue*.

Information about the actual compression behavior during a **dsmSendObj** is returned by the **dsmEndSendObjEx** call. *objCompressed* specifies if compression was done. *totalBytesSent* is the number of bytes sent by the application. *totalCompressedSize* is the number of bytes after compression. The **dsmEndSendObjEx** call also has a *totalLFBytesSent* field that contains the total bytes sent over LAN-free.

Attention: If your application plans to use partial object restore or retrieve, you cannot compress the data while sending it. To enforce this, set the **dsmSendObj** *ObjAttr.objCompressed* to *bTrue*.

Buffer copy elimination

The buffer copy elimination function removes the copy of data buffers between an application and the Tivoli Storage Manager server, which results in better CPU utilization. For maximum effect, this should be used in a LAN-free environment.

The basis for this function is a mechanism where the buffers for data movement are allocated by Tivoli Storage Manager and a pointer is passed back to the application. The application places the data in the provided buffer, and that buffer is passed through the communication layers to the storage agent (using shared memory). Then the data is moved to the tape device, which eliminates copies of data. This function can be used with either backup or archive operations.

Attention: When using this method, pay extra attention to proper buffer handling and sizes of buffers. The buffers are shared between the components and any memory overwrite that is a result of a programming error results in severe errors.

The overall sequence of calls for backup/archive is as follows:

```
dsmInitEx (UseTsmBuffers = True, numTsmBuffers = [how many Tivoli Storage Manager
           -allocated buffers the application needs to allocate])
dsmBeginTxn
for each object in the txn
  dsmBindMC
    dsmSendObject
      dsmRequestBuffer
      dsmSendBufferData (sends and release the buffer used)
    dsmEndSendObjEx
dsmEndTxn
for each buffer still held
      dsmReleaseBuffer
dsmTerminate
```

dsmRequestBuffer can be called multiple times (up to *numTsmBuffers*). An application can have two threads: a producer thread that fills buffers with data; and a consumer thread that sends those buffers to Tivoli Storage Manager with the **dsmSendBufferData** call. When a **dsmRequestBuffer** call is issued and the **numTsmBuffers** has been reached, the **dsmRequestBuffer** call blocks until a buffer is released. This could happen by either calling **dsmSendBufferData**, which sends and releases a buffer or by calling **dsmReleaseBuffer**. See callbuff.c in the API sample directory.

If at any point there is a failure in the send, the application must release all the buffers it is holding and terminate the session. For example:

```
  If failure
    for each TSM buffer held by application
      call dsmReleaseBuffer
  dsmTerminate
```

If an application calls **dsmTerminate** and it is still holding a buffer, the API does not exit and returns the return code: DSM_RC_CANNOT_EXIT_MUST_RELEASE_BUFFER. If the application cannot release the buffers it should exit the process to force a cleanup.

Buffer copy elimination and restore and retrieve

The amount of data to be placed in the buffer is controlled by the Tivoli Storage Manager server and is based on tape access optimization with restore and retrieve, so it is not as beneficial to the application as the normal method of getting data. After you prototype the buffer copy elimination method, check the performance and use it only if there is worthwhile improvement.

The maximum amount of data in a single buffer returned by the Tivoli Storage Manager server is (256K bytes – header overhead). This means that only applications that deal with small buffer writes benefit from this mechanism when getting data. The application must give special attention to the number of bytes in the provided buffer, because it is dependent on size of the object, the network, and other boundary conditions. In some situations, the use of buffer copy elimination can actually perform worse than the normal restore. The API normally caches the data and returns a fixed length to the application, which gives the application the ability to control the number of data writes back to the disk.

If you choose buffer copy elimination, create a data caching mechanism for buffers that are less than the preferred write buffer size. For example, if an application writes to disk 64K data blocks, it should call **dsmGetBufferData** and write out blocks of 64K. On the last block, it should copy the remainder to a **tempBuff**, issue another **dsmGetBufferData** call, and fill the **tempBuff** with the rest of the data and continue writing blocks of 64K:

```
dsmGetBufferData #1 get 226K    dsmGetBufferData #2 get 240K
Block1 64K – write to disk      Block1 30K – copy to tempbuff - write to disk
Block2 64K – write to disk     Block2 64K – write to disk
Block3 64K – write to disk      Block3 64K – write to disk
Block4 34K  - copy to tempbuff   Block4 64K – write to disk
Block5 18K – write to tempbuff etc
```

In this example, out of 7 disk writes, 6 were direct and 1 was cached.

The overall sequence of calls for restore/retrieve is as follows: **dsmInitEx** (*UseTsmBuffers* = *True*, *numTsmBuffers* = how many buffers the application wants to allocate).

```
dsmBeginGetData
While obj id
  dsmGetObj (no data restored on this call- buffer set to NULL)
 While data to read
   dsmGetBufferData (returns the data in the TSM buffer)
  ...process data...
  dsmReleaseBuffer
dsmEndGetObj
dsmEndGetData
```

For every **dsmGetBufferData** call implement a **dsmReleaseBuffer** call. The **dsmGetBufferData** and corresponding **dsmReleaseBuffer** do not need to be consecutive. An application might issue multiple **dsmGetBufferData** calls first to get several buffers, and then issue the corresponding **dsmReleaseBuffer** calls later. For sample code using this function, see callbuff.c in the API sample directory.

Restrictions: Since the API is providing the buffer and the goal is to minimize CPU utilization, additional processing of the data in the buffer is not permitted. This means that encryption and compression are not allowed when using buffer copy elimination since both of these operations require data processing and copies.

Recommendation: Implement both the regular data movement path and the buffer copy elimination to enable the user to switch between both paths, based on their needs. If the user needs to compress or encrypt data, they should use the existing mechanism. If there is a CPU constraint, they should use the new mechanism. Both of these mechanisms are complementary and are not intended to totally replace each other.

API encryption

There are two methods to handle encryption: application managed encryption and transparent encryption. You must ensure that your chosen method is the only method you use, so you avoid problems during restore and retrieve.

Note: If using encryption, the Tivoli Storage Manager server should be set with *Authentication* set to ON.

For example, if an application used application managed encryption to encrypt object A and transparent encryption to encrypt object B, during restore, since the methods are mutually exclusive, the application sets the option to use transparent encryption and tries to restore both objects. Object B is restored but object A fails to restore.

For both of the encryption methods, the encrypted objects must match an *include.encrypt* pattern. The two encryption methods are mutually exclusive and in a given API invocation you can use only one of the methods. If both methods are specified, the API fails with return code DSM_RC_ENCR_NOT_ALLOWED.

On most platforms, the API uses either 128 bit AES or 56 bit DES encryption. On OS/400, and zOS/USS, the API uses 56 bit DES encryption. Encryption can be enabled with or without compression. There is no support of partial object restore, retrieve or buffer copy elimination when using encryption.

Application managed encryption

With application managed encryption, the application provides the key password to the API (using key DSM_ENCRYPT_USER) and it is the application's responsibility to manage the key password.

Remember: If the encryption key is not saved, and you have forgotten the key, your data will be unrecoverable.

The application provides the key password in the **dsmInitEx** call and must provide the proper key password at restore time. **If the key password is lost, there is no way to restore the data.** The same key password must be used for backup and restore (or archive and retrieve) of the same object. This method does not have a Tivoli Storage Manager server level dependency. To set up this method, the application needs to do the following:

1. Set the *bEncryptKeyEnabled* variable to *bTrue* in the call to **dsmInitEx**, and set the *encryptionPasswordP* variable to point to a string with the encrypt key password.
2. Set the *include.encrypt* for the objects to *encrypt*. For example, to encrypt all data, set:

   ```
   include.encrypt /.../* (UNIX)
   ```

   and

   ```
   include.encrypt *\...\* (Windows)
   ```

   To encrypt the object /FS1/DB2/FULL, set:

   ```
   include.encrypt /FS1/DB2/FULL
   ```

3. Set *ENCRYPTKEY=PROMPT|SAVE* in the option string passed to the API in the **dsmInitEx** call on Windows. This option can also be set in dsm.opt (Windows) or dsm.sys (UNIX or Linux).

Note: By default, the *encryptkey* option is set to *prompt*. This is to ensure that the key does not get stored automatically. If *encryptkey* save is specified, the key is stored by Tivoli Storage Manager on the local machine but then only one key can be valid for all Tivoli Storage Manager operations with the same node name.

After a send of an object, the **dsmEndSendObjEx** specifies whether an object has been encrypted and which method was used. Possible values in the *encryptionType* field are:

- DSM_ENCRYPT_NO
- DSM_ENCRYPT_USER
- DSM_ENCRYPT_CLIENTENCRKEY

The following table lists the API encryption types, prerequisites, and functions available.

Table 18. API encryption types, prerequisites, and functions available

Type	Prerequisite	Function available
ENCRYPTIONTYPE	None	Set -ENCRYPTIONTYPE=DES56|AES128 in the option string passed to the API in the **dsmInitEx** call on Windows. This option can also be set in dsm.opt (Windows) or dsm.sys (UNIX). ENCRYPTIONTYPE is AES128 by default.
EncryptKey=save	None	API and backup-archive
EncryptKey=prompt	None	API and backup-archive
EncryptKey=generate	None	API and backup-archive
EnableClientEncryptKey	None	API only

Note: The server should have authentication turned ON. If authentication is turned OFF, the key is not encrypted, but the data is still encrypted. This is not recommended.

Table 19 on page 46 shows how both Authorized Users and non-Authorized Users can encrypt or decrypt data during a backup or restore operation, depending on the value specified for the *passwordaccess* option. The TSM.PWD file must exist to perform the following authorized-user and non-authorized-user operations. The authorized user creates the TSM.PWD file and sets the *encryptkey* option to save and the *passwordaccess* option to generate.

Table 19. Encrypting or decrypting data with application managed key on UNIX or Linux

Operation	passwordaccess option	encryptkey option	Result
Authorized user backup	generate	save	data encrypted
	generate	prompt	data encrypted if encryptionPasswordP contains an encryption password
	prompt	save	data encrypted if encryptionPasswordP contains an encryption password
	prompt	prompt	data encrypted if encryptionPasswordP contains an encryption password
Authorized user restore	generate	save	data encrypted
	generate	prompt	data encrypted if encryptionPasswordP contains an encryption password
	prompt	save	data encrypted if encryptionPasswordP contains an encryption password
	prompt	prompt	data encrypted if encryptionPasswordP contains an encryption password
Non-authorized user backup	generate	save	data encrypted
	generate	prompt	data encrypted if encryptionPasswordP contains an encryption password
	prompt	save	data encrypted if encryptionPasswordP contains an encryption password
	prompt	prompt	data encrypted if encryptionPasswordP contains an encryption password
Non-authorized user restore	generate	save	data encrypted
	generate	prompt	data encrypted if encryptionPasswordP contains an encryption password
	prompt	save	data encrypted if encryptionPasswordP contains an encryption password
	prompt	prompt	data encrypted if encryptionPasswordP contains an encryption password

Using transparent encryption

Using the transparent encryption method, with the key managed by Tivoli Storage Manager DSM_ENCRYPT_CLIENTENCRKEY, you can benefit from data encryption without having to change the code at all. It can be totally transparent to the application.

Note: For both transparent and application-managed encryption, the *encryptionPassword* refers to a string value that is used to generate the "real" encryption key. The *encryptionPassword* can be up to 63 characters in length, but the key generated from it is always 8 bytes for 56 DES and 16 bytes for 128 AES.

Attention: If the encryption key is not available, **data cannot be restored or retrieved under any circumstance**. When using ENABLECLIENTENCRYPTKEY for encryption, the encryption key is stored on the server database. This means that for objects using this method, the server database must exist and have the proper values for the objects for a proper restore. Ensure that you back up the server database frequently to prevent data loss.

This is the simpler method to implement, where one random encryption key is generated per session and it is stored on the Tivoli Storage Manager server with the object in the server database. During restore, the stored key is used for decryption. Using this method, the management of the key is the responsibility of Tivoli Storage Manager, and the application does not have to deal with the key at all. It is also important to note that since the key is stored in the server database, it is always required to have a valid Tivoli Storage Manager database for a restore of an encrypted object. When the key is transmitted between the API and the server it is also encrypted. The transmission of the key is secure, and when the key is stored in the Tivoli Storage Manager server database it is encrypted. The only time that the key is placed in the clear with the export data stream is when a node's data are exported between servers.

To enable transparent encryption, do the following:

1. Specify -ENABLECLIENTENCRYPTKEY=YES in the option string passed to the API on the dsmInitEx call *or* set the option in the system option file dsm.opt (Windows) or dsm.sys (UNIX or Linux).
2. Set the include.encrypt for the objects to encrypt. For example, to encrypt all data, set:

   ```
   include.encrypt /.../* (UNIX)
   ```

   and

   ```
   include.encrypt *\...\* (Windows)
   ```

   To encrypt the object /FS1/DB2/FULL, set:

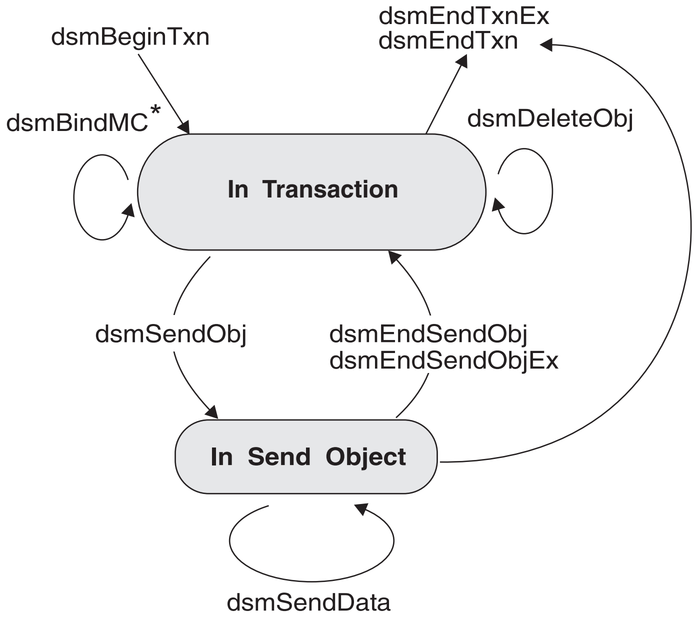
   ```
   include.encrypt /FS1/DB2/FULL
   ```

Example flow diagrams for backup and archive

The API is designed for straightforward logic flows and clear transitions between the various states of the application client. This clean state transition catches logic flaws and program errors early in the development cycle, greatly enhancing the quality and reliability of the system.

For example, you cannot make a **dsmSendObj** call unless a transaction was started and a **dsmBindMC** call was previously made for the object that you are backing up.

Figure 12 on page 48 displays the state diagram for performing backup or archive operations within a transaction. The arrow pointing from "In Send Object" to **dsmEndTxn** indicates that a **dsmEndTxn** call can be started after a call to **dsmSendObj** or **dsmSendData**. You might want to do this if an error condition occurred during the send of an object and you want to stop the entire operation. In this case, you must use a vote of DSM_VOTE_ABORT. In normal circumstances, however, call **dsmEndSendObj** before you end the transaction.



* May be inside or outside of a transaction

Figure 12. State diagram for backup and archive operations

Figure 13 on page 49 displays the flowchart for performing backup or archive operations within a transaction.

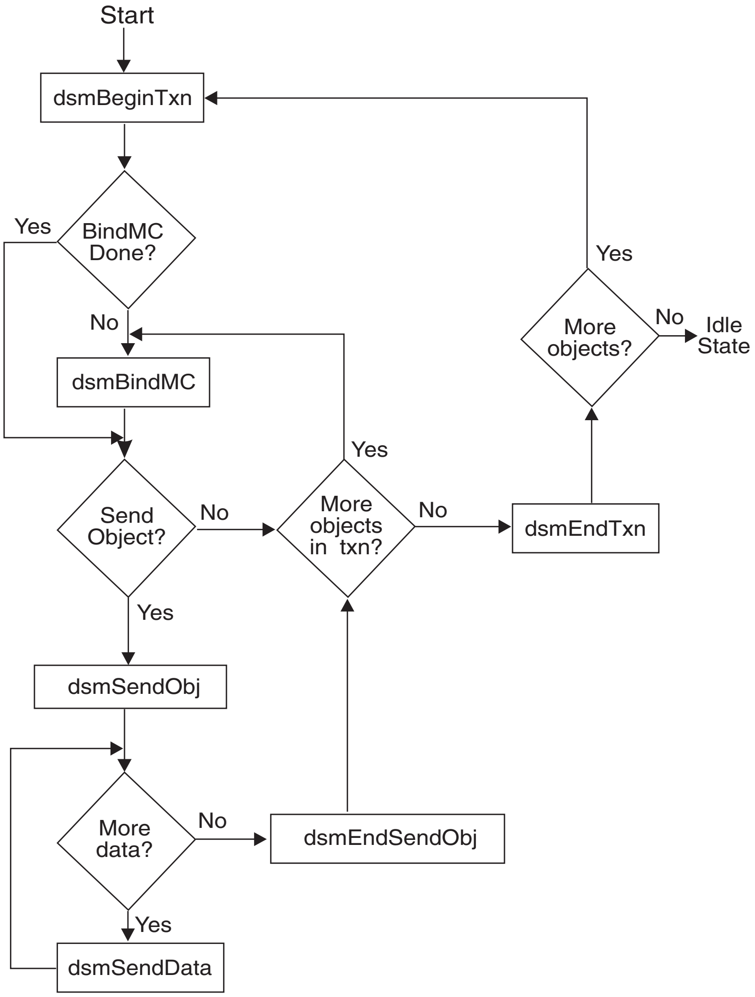


Figure 13. Flowchart for backup and archive operations

The primary feature in these two diagrams is the loop between the following API calls from within a transaction:

- **dsmBindMC**
- **dsmSendObj**
- **dsmSendData**
- **dsmEndSendObj**

The **dsmBindMC** call is unique in that you can start it from inside or outside of a transaction boundary. You can also start it from a different transaction, if required. The only requirement for the **dsmBindMC** call is that it is made prior to backing up or archiving an object. If the object that you are backing up or archiving is not associated with a management class, an error code is returned from **dsmSendObj**. In this situation, the transaction is ended by calling **dsmEndTxn** (this error condition is not shown in the flowchart).

The flowchart illustrates how an application would use multiple object transactions. It shows where decision points can be placed to determine if the object that is sent fits within the transaction or whether to start a new transaction.

Code example of API functions that send data to Tivoli Storage Manager storage

This example demonstrates the use of the API functions that send data to Tivoli Storage Manager storage. The **dsmSendObj** call appears inside a switch statement, so that different parameters can be called depending on whether a backup or archive operation is being performed.

The **dsmSendData** call is called from inside a loop that repeatedly sends data until a flag is set that permits the program execution to exit the loop. The entire send operation is performed from within the transaction.

The third parameter on the **dsmSendObj** call is a buffer that contains the archive description. Because backup objects do not have a description, this parameter is NULL when backing up an object.

Figure 8 on page 32 displays an example that shows the use of the **dsmBindMC** function call.

```
if ((rc = dsmBeginTxn(dsmHandle)) )       /* API session handle  */
{
   printf("*** dsmBeginTxn failed: ");
   rcApiOut(dsmHandle, rc);
   return;
}

/*  Call dsmBindMC if not done previously */
objAttr.sizeEstimate.hi = 0;       /* estimate of        */
objAttr.sizeEstimate.lo = 32000;   /*        object size */
switch (send_type)
{
   case (Backup_Send) :
      rc = dsmSendObj(dsmHandle,stBackup,
      NULL,&objName,&objAttr,NULL);
      break;
   case (Archive_Send) :
      archData.stVersion = sndArchiveDataVersion;
      archData.descr = desc;
      rc = dsmSendObj(dsmHandle,stArchive,
   &archData,&objName,&objAttr,NULL);
      break;
   default : ;
}
if (rc)
{
   printf("*** dsmSendObj failed: ");
   rcApiOut(dsmHandle, rc);
   return;
}
done = bFalse;
while (!done)
{
   dataBlk.stVersion = DataBlkVersion;
   dataBlk.bufferLen = send_amt;
   dataBlk.numBytes  = 0;
   dataBlk.bufferPtr = bkup_buff;
   rc = dsmSendData(dsmHandle,&dataBlk);
   if (rc)
   {
      printf("*** dsmSendData failed: ");
      rcApiOut(dsmHandle, rc);
      done = bTrue;
   }
   /* Adjust the dataBlk buffer for the next piece to send */
}
rc = dsmEndSendObj(dsmHandle);
if (rc)
{
   printf("*** dsmEndSendObj failed: ");
   rcApiOut(dsmHandle, rc);
}
txn_reason = 0;
rc = dsmEndTxn(dsmHandle,           /* API session handle       */
             DSM_VOTE_COMMIT,      /* Commit transaction       */
             &txn_reason);         /* Reason if txn aborted    */
if (rc || txn_reason)
{
   printf("*** dsmEndTxn failed: rc = ");
   rcApiOut(dsmHandle, rc);
   printf("   reason =
}
```

Figure 14. An example of sending data to a server

File grouping

The Tivoli Storage Manager API has a logical file grouping protocol that relates several individual objects together. You can reference and manage these groups as a logical group on the server. A logical group requires all group members and the group leader belong to the same node and file space on the server.

Each logical group has a group leader. If the group leader is deleted, the group is deleted. You cannot delete a member if it is part of a group. Expiration of all members in a group is dependent on the group leader. For example, if a member is marked for expiration, it does not expire unless the group leader expires. However, if a member is not marked for expiration, and the group leader is expired, then all members are expired.

File groups can only contain backup data, not archive data. Archive objects can use the **Archive Description** field to facilitate a type of grouping if required by an application.

The **dsmGroupHandler** call groups the operations. The **dsmGroupHandler** function must be called from within a transaction. Most group error conditions are caught on either the **dsmEndTxn**l or **dsmEndTxnEx** calls.

The out structure in **dsmEndTxnEx** includes a new field, **groupLeaderObjId**. This field contains the object ID of the group leader if a group was opened in that transaction. You can create a group across more than one transaction. A group is not committed, or saved, on the server until a close is performed. The **dsmGroupHandler** is an interface that can accept five different operations. They include:
- DSM_GROUP_ACTION_OPEN
- DSM_GROUP_ACTION_CLOSE
- DSM_GROUP_ACTION_ADD
- DSM_GROUP_ACTION_ASSIGNTO
- DSM_GROUP_ACTION_REMOVE

Table 20 lists the **dsmGroupHandler** function call actions:

Table 20. dsmGroupHanlder functions

Action	**Description**
OPEN	The OPEN action creates a new group. The next object that is sent becomes the group leader. The group leader cannot have content. All objects after the first object become members that are added to the group. To create a group, open a group and pass in a unique string to identify the group. This unique identifier allows several groups with the same name to be opened. After the group is opened, the next object that is sent is the group leader. All other objects that are sent are group members.
CLOSE	The CLOSE action commits and saves an open group. To close the group, pass in the object name and the unique string that is used in the open operation. The application must check for open groups and, if necessary, close or delete them. A group is not committed or saved until a close of the group is performed. You cannot close a new group with the same name as an existing open group. This causes the CLOSE action to fail. A CLOSE action can also fail if there is a management class incompatibility between the current closed group and the new group to be closed of the same name. Before issuing a close group, query the previous closed group and if the management class of the existing closed group is different than the management class associated with the current open group, issue a **dsmUpdateObject** with type DSM_BACKUPD_MC to update the existing group to the new management class. You can then issue the close.
ADD	The ADD action appends an object to a group. All objects that are sent after the ADD action are assigned to the group.

Table 20. dsmGroupHanlder functions (continued)

Action	Description
ASSIGNTO	The ASSIGNTO action permits the client to assign objects that exist on the server to the declared peer group. This is similar to the ADD action except that the add applies to objects within an in-flight transaction and the ASSIGNTO action applies to an object that is on the server. This transaction sets up the PEER group relationship.
REMOVE	The REMOVE action removes a member, or a list of members, from a group. A group leader cannot be removed from a group. A group member must be removed before it can be deleted.

There are two new query types for group support:

- **qtBackupGroups**
- **qtOpenGroups**

The **qtBackupGroups** queries groups that are closed while **qtOpenGroups** queries groups that are open. The query buffer for the new types has fields for **groupLeaderObjId** and **objType**. The query performs differently depending on the values for these two fields. The following table includes some query possibilities:

Table 21. Examples of queries

groupLeaderObjId.hi	groupLeaderObjId.lo	objType	Result
0	0	NULL	Returns a list of all group leaders
grpLdrObjId.hi	grpLdrObjId.lo	0	Returns a list for all group members that are assigned to the specified group leader (**grpLdrObjId**).
grpLdrObjId.hi	grpLdrObjId.lo	objType	Returns a list (using **BackQryRespEnhanced3**) for each group member that is assigned to the specified group leader (**grpLdrObjId**), and matching the object type (**objType**).

The response structure (**qryRespBackupData**) from **dsmGetNextQObj** includes two fields for group support:

- **isGroupLeader**
- **isOpenGroup**

These are Boolean flags. The following example displays the creation of the group, adding members to the group, and closing the group to commit it on the Tivoli Storage Manager server. Refer to the sample group program (dsmgrp.c) that is included in the API sampsrc directory for an actual code example.

```
dsmBeginTxn
 dsmGroupHandler (PEER, OPEN, leader, uniqueId)
 dsmBeginSendObj
  dsmEndSendObj
dsmEndTxnEx (With objId of leader)
Loop for multiple txns
{
 dsmBeginTxn
  dsmGroupHandler (PEER, ADD, member, groupLeaderObjID)
  Loop for multiple objects
  {
   dsmBeginSendObj
   Loop for data
   {
    dsmSendData
   }
   dsmEndSendObj
  }
 dsmEndTxn
}
dmBeginTxn
 dsmGroupHandler(CLOSE)
dsmEndTxn
```

Figure 15. Example of pseudo-code to create a group

Receiving data from a server

Application clients can receive data or named objects and their associated data from Tivoli Storage Manager storage by using the restore and retrieve functions of the product. The restore function accesses objects that previously were backed up, and the retrieve function accesses objects that previously were archived.

Note: The API can only restore or retrieve objects that were backed up or archived using API calls.

Both restore and retrieve functions start with a query operation. The query returns different information depending on whether the data was originally backed up or archived. For instance, a query on backup objects returns information on whether an object is active or inactive, while a query on archive objects returns information such as object descriptions. Both queries return object IDs that Tivoli Storage Manager uses to uniquely identify the object on the server.

Partial object restore or retrieve

The application client can receive only a portion of the object. This is called a partial object restore or a partial object retrieve.

Attention: Partial restore or retrieve of compressed or encrypted objects produces unpredictable results.

Note: If you code your application to use a partial object restore or retrieve, you cannot compress the data while sending it. To enforce this, set *ObjAttr.objCompressed* to *bTrue*.

To perform a partial object restore or retrieve, associate the following two data fields with each object **GetList** entry:

offset	The byte offset into the object from which to begin returning data.
length	The number of object bytes to return.

Use DSM_MAX_PARTIAL_GET_OBJ to determine the maximum number of objects that can perform a partial object restore or retrieve for a specific **dsmBeginGetData** list.

The following data fields, used on the **dsmBeginGetData** call, determine what portion of the object is restored or retrieved:

- If both the offset and length are zero, the entire object is restored or retrieved from Tivoli Storage Manager storage.
- If the offset is greater than zero, but the length is zero, the object is restored or retrieved from the offset to the end.
- If the length is greater than zero, only the portion of the object from the offset for the specified length is restored or retrieved.

Restoring or retrieving data

After a query is made and a session is established with the Tivoli Storage Manager server, you can run a procedure to restore or retrieve data.

Follow these steps:

1. Query the Tivoli Storage Manager server for either backup or archive data.
2. Determine the objects to restore or retrieve from the server.
3. Sort the objects on the **Restore Order** field.
4. Send the **dsmBeginGetData** call with the list of objects that you want to access.
5. Send the **dsmGetObj** call to obtain each object from the system. Multiple **dsmGetData** calls might be needed for each object to obtain all associated object data. Send the dsmEndGetObj call after all data for an object is obtained.
6. Send the **dsmEndGetData** call after all data for all objects is received, or to end the receive operation.

Querying the server

Before you can begin any restore or retrieve operation, first query the Tivoli Storage Manager server to determine what objects you can receive from storage.

To send the query, the application must enter the proper parameter lists and structures for the **dsmBeginQuery** call. This includes the file space that the query examines and pattern-match entries for the high-level and low-level name fields. If the session was initialized with a NULL owner name, you do not need to specify the owner field. However, if the session was initialized with an explicit owner name, only objects that explicitly have that owner name associated with them are returned.

The point-in-time **BackupQuery** supplies a snapshot of the system at a given time. By specifying a valid date, you can query all files that were backed up to that time. Even if an object has an active backup from a later date, point-in-time overrides an object state so that the previous inactive copy is returned. An example of this is in pitDate. You must be connected to at least a Version 3 server to use point-in-time **BackupQuery**.

A query returns all information that was originally stored with the object, in addition to the following in Table 22 on page 56.

Table 22. Query to the server return information

Field	Description
copyId	The **copyIdHi** and **copyIdLo** values provide an eight-byte number that uniquely identifies this object for this node in Tivoli Storage Manager storage. Use this ID to request a specific object from storage for restore or retrieve processing.
restoreOrderExt	The **restoreOrderExt** value provides a mechanism for receiving objects from Tivoli Storage Manager storage in the most efficient manner possible. Sort the objects to restore on this value to ensure that tapes are mounted only once and are read from front to back.

You must keep some or all of the query information for later processing. Keep the **copyId** and **restoreOrderExt** fields because they are needed for the actual restore operation. You must also keep any other information needed to properly open a data file or identify a destination.

Call **dsmEndQuery** to finish the query operation.

Selecting and sorting objects by restore order

Once the backup or archive query is performed, the application client must determine which objects, if any, are to be restored or retrieved. Then you sort them in ascending order (low to high). This sorting is very important to the performance of the restore operation. Sorting the objects on the **restoreOrderExt** fields ensures that the data are read from the server in the most efficient order.

All data on disk is restored first, followed by data on media classes that require volume mounts (such as tape). The **restoreOrderExt** field also ensures that data on tape is read in order with processing starting at the front of a tape and progressing towards the end.

Properly sorting on the **restoreOrderExt** field means that duplicate tape mounts and unnecessary tape rewinds do not occur.

A non-zero value in the **restoreOrderExt**.top field correlates to a unique serial access device on the Tivoli Storage Manager server. Since a serial access device can only be used by one session / mount point at a time, the application should ensure that if it uses multiple sessions there are not concurrent restores with the same **restoreOrderExt**.top value. Otherwise the first session are able to access the objects, but other sessions wait until the first session terminates and the device becomes available.

Following is an example of sorting objects by using **Restore Order** fields.

Figure 16. An example of sorting objects with the restore order fields

```
typedef struct {
dsStruct64_t      objId;
dsUint160_t      restoreOrderExt;

}  SortOrder;                /* struct used for sorting */

===================================================================
/* the code for sorting starts from here */
dsmQueryType      queryType;
qryBackupData     queryBuffer;
```

```
DataBlk           qDataBlkArea;
qryRespBackupData qbDataArea;
dsInt16_t   rc;
dsBool_t  done = bFalse;
int i = 0;
int qry_item;
SortOrder  sortorder[100]; /* sorting can be done up to 100 items
                                  only right now.  Set appropriate
                                  array size to fit your needs */
/*-----------------------------------------------------------------+
| NOTE: Make sure that proper initializations have been done to
|  queryType,
|       queryBuffer, qDataBlkAre, and qbDataArea.
|
------------------------------------------------------------------*/

   qDataBlkArea.bufferPtf = (char*) &qbDataArea;

   rc = dsmBeginQuery(dsmHandle, queryType, (void *) &queryBuffer);

   /*----------------------------------------+
   | Make sure to check rc from dsmBeginQuery
   +-----------------------------------------*/
   while (!done)
   {
      rc = dsmGetNextQObj(dsmHandle, &qDataBlkArea);
 if ((rc == DSM_RC_MORE_DATA) ||
          (rc == DSM_RC_FINISHED))
           &&( qDataBlkArea.numBytes))
      {
         /******************************************/
         /* transferring restoreOrderExt and objId */
         /******************************************/
         sortorder[i].restoreOrderExt = qbDataArea.restoreOrderExt;
         sortorder[i].objId = qbDataArea.objId;

      } /*  if ((rc == DSM_RC_MORE_DATA) || (rc == DSM_RC_FINISHED)) */
      else
      {
          done = bTrue;
          /****************************/
          /* take appropriate action. */
          /****************************/
      }

      i++;
      qry_item++;

   } /* while (!done) */
   rc = dsmEndQuery(dsmHandle);
  /*check rc  */
   /*****************************************************/
   /* sorting the array using qsort.  After the call,   */
   /* sortorder will be sorted by restoreOrderExt field */
   /*****************************************************/

   qsort(sortorder, qry_item, sizeof(SortOrder), SortRestoreOrder);

 /*-----------------------------------------------------------------+
| NOTE: Make sure to extract sorted object ids and store them in
|       any data structure you want.
------------------------------------------------------------------*/

/*----------------------------------------------------------------+
| int SortRestoreOrder(SortOrder *a, SortOrder *b)
|
|
| This function compares restoreOrder fields from two structures.
| if (a > b)
|   return(GREATERTHAN);
|| if (a < b)
|   return(LESSTHAN);
|| if (a == b)
|   return(EQUAL);
|+-----------------------------------------------------------------*/
int SortRestoreOrder(SortOrder *a, SortOrder *b)
{
    if (a->restoreOrderExt.top > b->restoreOrderExt.top)
       return(GREATERTHAN);
    else if (a->restoreOrderExt.top < b->restoreOrderExt.top)
       return(LESSTHAN);
    else if (a->restoreOrderExt.hi_hi > b->restoreOrderExt.hi_hi)
       return(GREATERTHAN);
    else if (a->restoreOrderExt.hi_hi < b->restoreOrderExt.hi_hi)
       return(LESSTHAN);
    else if (a->restoreOrderExt.hi_lo > b->restoreOrderExt.hi_lo)
       return(GREATERTHAN);
    else if (a->restoreOrderExt.hi_lo < b->restoreOrderExt.hi_lo)
       return(LESSTHAN);
     else if (a->restoreOrderExt.lo_hi > b->restoreOrderExt.lo_hi)
       return(GREATERTHAN);
    else if (a->restoreOrderExt.lo_hi < b->restoreOrderExt.lo_hi)
       return(LESSTHAN);
    else if (a->restoreOrderExt.lo_lo > b->restoreOrderExt.lo_lo)
       return(GREATERTHAN);
    else if (a->restoreOrderExt.lo_lo < b->restoreOrderExt.lo_lo)
       return(LESSTHAN);
    else
       return(EQUAL);
}
```

Starting the dsmBeginGetData call

Once you select and sort the objects to receive, submit them to Tivoli Storage Manager for either a restore or retrieve. The **dsmBeginGetData** call begins a restore or retrieve operation. The objects are returned to the application client in the order you requested.

Complete the information for these two parameters in these calls:

mountWait

This parameter tells the server whether the application client waits for offline media to be mounted in order to obtain data for an object, or whether that object should be skipped during processing of the restore or retrieve operation.

dsmGetObjListP

This parameter is a data structure that contains the **objId** field which is a list of all object IDs that are restored or retrieved. Each **objId** is associated with a **partialObjData** structure that describes whether the entire **objId** or only a particular section of the object will be retrieved.

Each **objId** is eight bytes in length, so a single restore or retrieve request can contain thousands of objects. The number of objects to request in a single call is limited to DSM_MAX_GET_OBJ or DSM_MAX_PARTIAL_GET_OBJ.

Receiving each object to restore or retrieve

Once the **dsmBeginGetData** call is sent, you can perform a procedure to receive each object that is sent from the server

Note:

- The DSM_RC_MORE_DATA return code means that a buffer was returned and you should call **dsmGetData** again. Check the **DataBlk.numBytes** for the actual number of returned bytes.
- When you obtain all data for an object, you must send a **dsmEndGetObj** call. If more objects will be received, send the **dsmGetObj** call again.
- If you need to stop the process (normally or abnormally), for example you want to discard any remaining data in the restore stream for all objects not yet received, send the **dsmEndGetData** call. This flushes the data from the server to the client. However, using this method might take time to complete. If you need to end a restore, use **dsmTerminate** to close the session.

1. Send the **dsmGetObj** call to identify the object that you requested from the data stream and, to obtain the first block of data that is associated with the object.
2. Send more **dsmGetData** calls, as necessary, to obtain the remaining object data.

Example flow diagrams for restore and retrieve

View these state diagram and flowchart for a visual example of how to perform restore or retrieve operations.

The arrow pointing from “In Get Object” to **dsmEndGetData** indicates that you can send a **dsmEndGetData** call after a call to **dsmGetObj** or dsmGetData. You might need to do this if an error condition occurred while getting an object from Tivoli Storage Manager storage and you want to stop the operation. In normal circumstances, however, call **dsmEndGetObj** first.

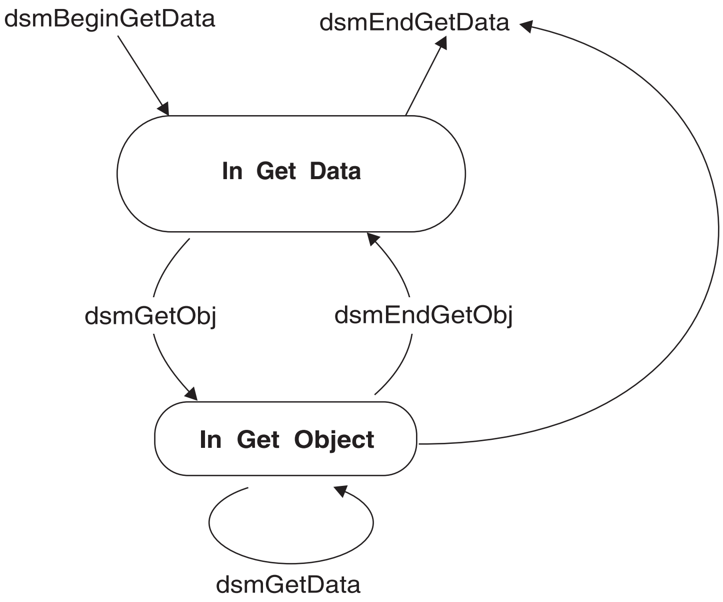


Figure 17. State diagram for restore and retrieve operations

Figure 18 on page 60 displays the flowchart for performing restore or retrieve operations.

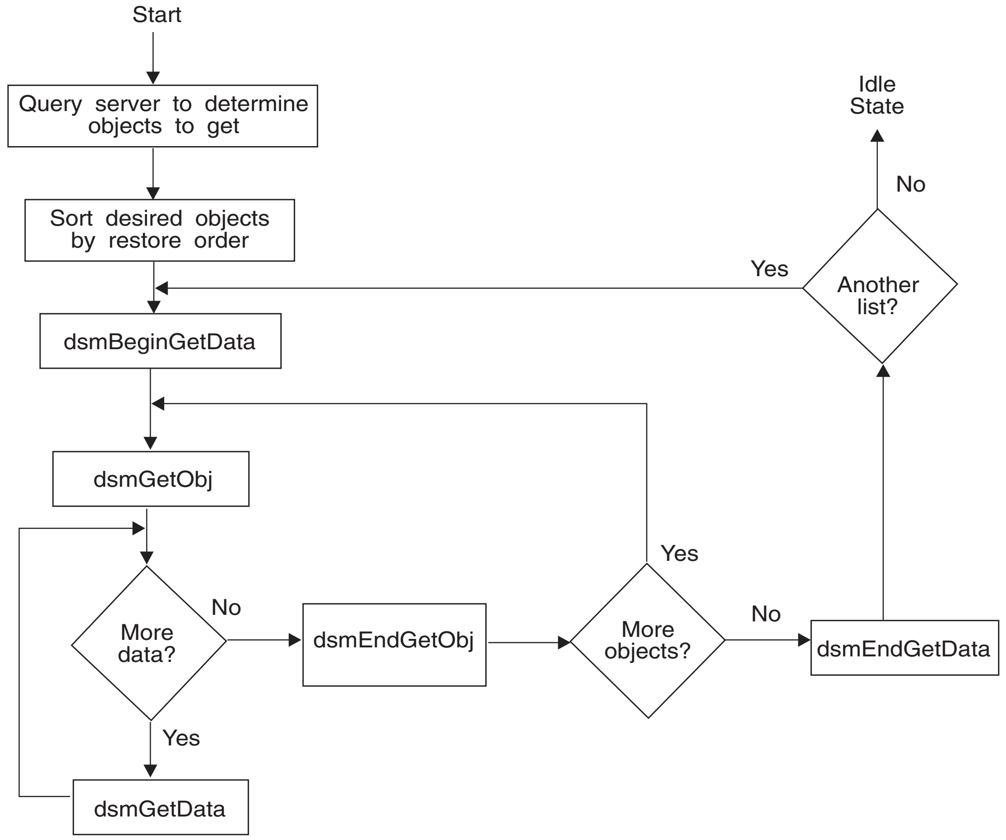


Figure 18. Flowchart for restore and retrieve operations

Code example of receiving data from a server

This example demonstrates using the API functions to retrieve data from Tivoli Storage Manager storage.

The **dsmBeginGetData** function call appears inside a switch statement, so that different parameters can be called depending on whether a restore or retrieve operation is being performed. The **dsmGetData** function call is called from inside a loop that repeatedly gets data from the server until a flag is set that permits the program execution to exit the loop.

Figure 19. An example of receiving data from a server

```
/* Call dsmBeginQuery and create a linked list of objects to restore. */
/* Process this list to create the proper list for the GetData calls. */
/* Set up the getList structure to point to this list.                */
/* This example is set up to perform a partial object retrieve.  To   */
/* retrieve only complete objects, set up:                            */
/*      getList.stVersion = dsmGetListVersion;                        */
/*      getList.partialObjData = NULL;                                */
dsmGetList getList;
getList.stVersion = dsmGetListPORVersion;   /* structure version      */
getList.numObjId  = items;               /* number of items in list   */
getList.objId     = (ObjID *)rest_ibuff;
                                    /* list of object IDs to restore  */
getList.partialObjData = (PartialObjData *) part_ibuff;
                                      /* list of partial object data  */
switch(get_type)
{
   case (Restore_Get) :
     rc = dsmBeginGetData(dsmHandle,bFalse,gtBackup,&getList);
     break;
   case (Retrieve_Get) :
     rc = dsmBeginGetData(dsmHandle,bFalse,gtArchive,&getList);
     break;
   default : ;
}
if (rc)
{
   printf("*** dsmBeginGetData failed: ");
   rcApiOut(dsmHandle, rc);
   return rc;
}
/* Get each object from the list and verify whether it is on the  */
/* server.  If so, initialize structures with object attributes for */
/* data validation checks.  When done, call dsmGetObj.            */
 rc = dsmGetObj(dsmHandle,objId,&dataBlk);
done = bFalse;
while(!done)
{
   if (   (rc == DSM_RC_MORE_DATA)
       || (rc == DSM_RC_FINISHED))
   {
      if (rc == DSM_RC_MORE_DATA)
      {
         dataBlk.numBytes = 0;
         rc = dsmGetData(dsmHandle,&dataBlk);
      }
      else
         done = bTrue;
   }
   else
   {
      printf("*** dsmGetObj or dsmGetData failed: ");
      rcApiOut(dsmHandle, rc);
      done = bTrue;
   }
} /* while */
rc = dsmEndGetObj(dsmHandle);
/* check rc from dsmEndGetObj */
/* check rc from dsmEndGetData */
rc = dsmEndGetData(dsmHandle);
return 0;
```

Updating and deleting objects on the server

Your API applications can use the**dsmUpdateObj** or **dsmUpdateObjEx** function call to update objects that were archived or backed up. Use either call in the session state only, updating one object at a time. Use **dsmUpdateObjEx** to update any of several archive objects containing the same name.

To select an archive object, set the **dsmSendType** function call to **stArchive**.

- With **dsmUpdateObj**, only the latest archive object with the assigned name is updated.
- With **dsmUpdateObjEx**, any archived object can be updated by specifying the proper object ID.

For an archived object, the application can update the following fields:

- Description
- Object information
- Owner

To select a backup object, set **dsmSendType** to **stBackup**. For backed-up objects, only the active copy is updated.

For a backed-up object, the application can update the following fields:
- Management class
- Object information
- Owner

Deleting objects from the server

API applications can make calls to either delete objects that were archived or turn off objects that were backed up. Deleting archived objects is dependent on the node authorization that was given when the Tivoli Storage Manager administrator registered the node. Administrators can specify that nodes can delete archived objects.

Use the **dsmDeleteObj** function call to delete archived objects and turn off backup objects. Using this **delType** removes the backup object from the server. This is based on **objID**, deletes an object from the server database. Only an owner of an object can delete it. You can delete any version (active or inactive) of an object. The server reconciles the versions. If you delete an active version of an object, the first inactive version becomes active. If you delete an inactive version of an object, all older versions advance. The node must be registered with **backDel** permission.

An archived object is marked for deletion in storage when the system performs its next object expiration cycle. Once you delete an archived object from the server, you cannot retrieve it.

When you inactivate a backup object at the server, the object moves from an active state to an inactive state. These states have different retention policies associated with them that are based on the management class that is assigned.

Similar to the **dsmSendObj** call, a call to **dsmDeleteObj** is sent within the boundary of a transaction. The state diagram in Figure 12 on page 48 displays how a call to **dsmDeleteObj** is preceded by a call to **dsmBeginTxn** and followed by a call to **dsmEndTxn**.

Logging events

An API application can log event messages to central locations. It can direct logging to the Tivoli Storage Manager server, the local machine, or both. The **dsmLogEventEx** function call is performed inside a session. To view messages logged on the server, use the query **actlog** command through the administrative client.

Note: See the Tivoli Storage Manager Administrator's Reference for more information.

We recommend that you use the Tivoli Storage Manager client option, *errorlogretention*, to prune the client error log file if the application generates numerous client messages that are written to the client log (*dsmLogType* either *logLocal* or *logBoth*).

Putting it all together - a summary diagram

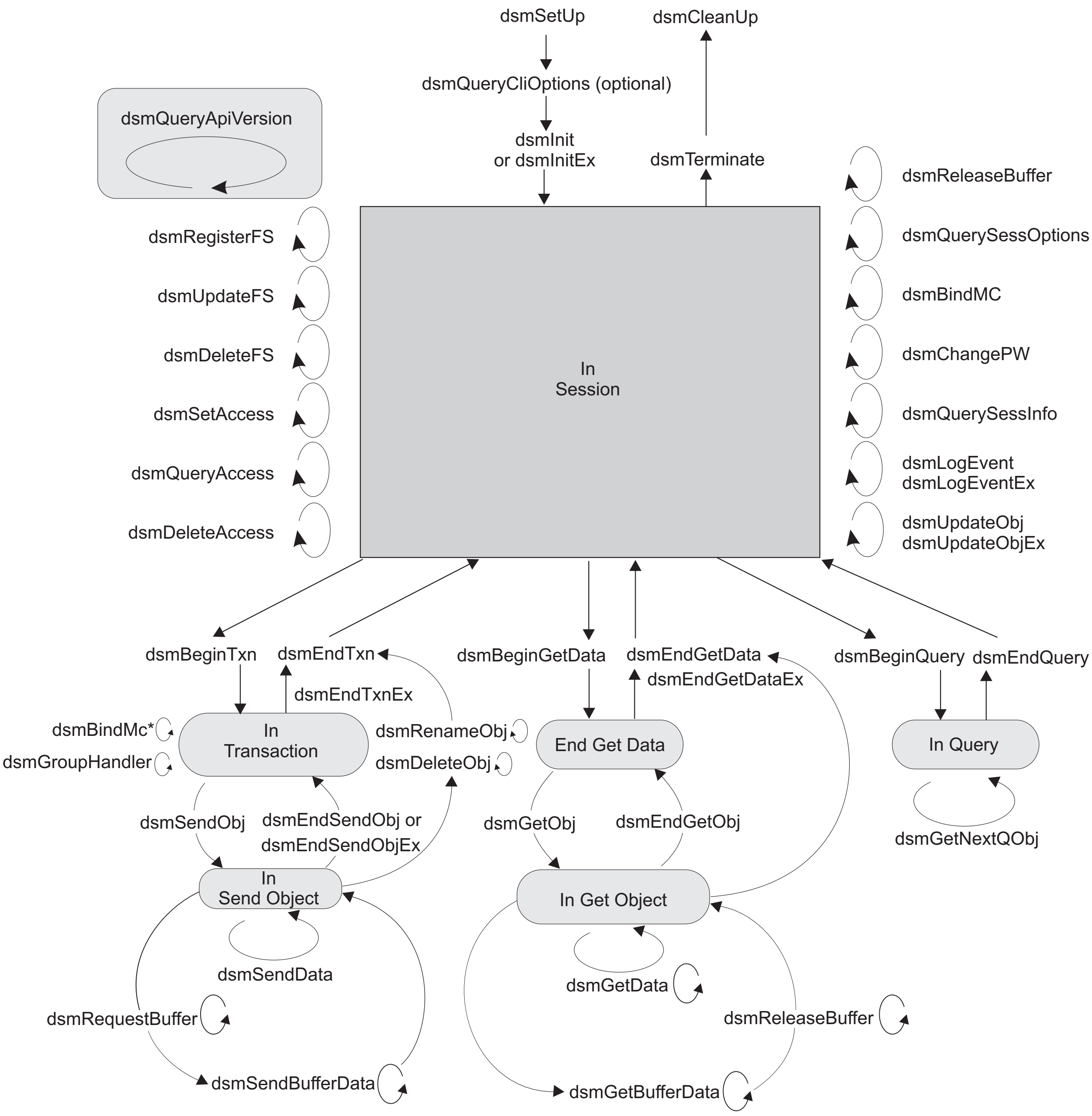
Once you review all the considerations for creating your own application with the Tivoli Storage Manager API, review this state diagram summary of an entire application.

Figure 20 on page 64 contains the state diagram for the API. It contains all previously displayed state diagrams in addition to several other calls previously not displayed.

The points in this diagram include:

- Call **dsmQueryApiVersionEx** at any time. It has no state associated with it. See Figure 1 on page 17 for an example.
- Call **dsmQueryCliOptions** before a **dsmInitEx** call only.
- Use **dsmRegisterFS**, **dsmUpdateFS**, and **dsmDeleteFS** to manage file spaces. These calls are made from within an idle session state. Use the **dsmBeginQuery** call to query file spaces. For more information about file space calls, see "Managing file spaces" on page 28.
- Send the **dsmBindMC** call from within an idle session state or from within a send object transaction state. See the example in Figure 8 on page 32.
- Send the **dsmChangePW** call from within an idle session state.

  Note: If the **dsmInitEx** call returns with a password-expired return code, the **dsmChangePW** call must be made before you start a valid session. See Figure 4 on page 23 for an example that uses **dsmChangePW**.
- If a call returns with an error, the state remains as it was. For example, if **dsmGetObj** returns with an error, the state remains In Get Data, and a call to **dsmEndGetObj** is a call sequence error.



* Can be inside or outside of a transaction

Figure 20. Summary state diagram for the API

Chapter 4. Understanding interoperability

The API has two types of interoperability: between the backup-archive client and API applications and between different operating systems.

Backup-archive client interoperability

The backup-archive command line can access API objects to provide limited interoperability. API objects can only be viewed and accessed from the backup-archive command line client and cannot be viewed or accessed from any of the GUI interfaces (native, web Java™). The backup-archive command-line client can only restore content of the file and nothing else, so you should only use it for a salvage type of operation.

The following command-line actions are provided:
- Delete archive
- Delete filespace
- Query
- Restore
- Retrieve
- Set access

The path information is actual directories for backup-archive client objects. In contrast, the API object path information might not have any relationship to existing directories: the path might be completely contrived. Interoperability does not change this aspect of these object types. To use this feature successfully, follow the restrictions and conventions.

Notes:
1. There is no interoperability between the backup-archive client and API objects stored on a retention protection server.
2. You cannot use the backup-archive client GUIs to access files that were stored using the API client. You can only use the command line to access these files.

Naming your API objects

Establish a consistent naming convention for API object names that contain the file space name, the high-level qualifier, and the low-level qualifier. The file space name and high-level qualifiers can refer to actual directory names, although this is not a requirement. Each can consist of more than one directory name that applies to the low-level qualifier.

We recommend that the low-level qualifier be the name of the object that is not prefixed with directory information. See "Object names and IDs" on page 24 for more information.

File space names must be fully qualified when they are referred from either the API or the backup-archive command line. For example, on a UNIX or Linux operating system, if you register file space /a and another file space, /a/b, then, when you refer to /a, it displays objects that are related only to file space /a. To view objects that are related to /a/b, specify /a/b as the file space name. After you register both file spaces, if you back up object b into file space /a, then a query for /a/b continues to display objects that are related to file space /a/b only.

The exception to this restriction occurs in file space references when you attempt to query or delete file spaces with the API. In both cases, it is not necessary for file space names to be fully qualified if you use a wildcard character. For example, /a* refers to both /a and /a/b.

Note: If interoperability is important to you, then avoid file space names that overlap.

On Windows-based operating systems, enclose file space names in braces { } for API objects when you access them from the backup-archive command line. Windows-based operating systems automatically place file space names in uppercase letters when you register or refer them. However, this is not true for the remainder of the object name specification. If you want full interoperability, place the high-level qualifier and the low-level qualifier in uppercase letters in the application when you back up API objects.

The examples that follow demonstrate these concepts. In both environments, it is not necessary to specify completely either the high-level or the low-level qualifier. However, if you do not, then you must use the wildcard character.

Platform	Example
Windows	To query all backed-up files in file space MYFS, enter: `dsmc q ba "{MYFS}\*\*"` **Note:** There is at least one asterisk (*) for each of the high-level and low-level qualifiers.
UNIX or Linux	To query all backed-up files in file space /A, enter: `dsmc q ba "/A/*/*"` **Note:** There is at least one asterisk (*) for each of the high-level and low-level qualifiers.

Backup-archive client commands you can use with the API

You can use a subset of backup-archive client commands within an application. For example, you can view and manage objects that other users own either on the same node or on a different node.

To view and manage objects that other users own either on the same node or on a different node, perform these steps:

1. Give access with the **set access** command.
2. Specify the owner and the node. Use the *fromowner* and *fromnode* options from the backup-archive command line to specify the owner and the node. For example:

   ```
   dsmc q ba "/A/*/*" -fromowner=other_owner -fromnode=other_node
   ```

Table 23 describes the commands that you can use with API objects.

Table 23. Backup-archive client commands you can use with API objects

Command	Description
Delete Archive	Archived files that the current user owns can be deleted. The set access command settings have no effect on this command.
Delete Filespace	The delete filespace command affects API objects.

Table 23. Backup-archive client commands you can use with API objects (continued)

Command	Description
Query	From the backup-archive command line, you can query backed up and archived API objects and objects that other users own, or that reside on other nodes. See "Naming your API objects" on page 65 for information about querying API objects. Use the existing *–fromowner* option to query objects that a different user owns for which the set access permission has been given. Use the existing *–fromnode* option to query objects that reside on another node for which the set access permission has been given. See "dsmInitEx" on page 99 for more information.
Restore **Retrieve**	**Note:** Only use these commands for exception situations. API objects encrypted using the application managed key can be restored or retrieved if the encryption key is known or saved in the password file or registry. API objects encrypted using transparent encryption cannot be restored or retrieved using the backup-archive client. These commands return data as bit files that are created using default file attributes. You can restore or retrieve API objects that other users own, or that are from a different node. The set access command determines which objects qualify.
dSet Access	The set access command permits users to manage API objects that another user owns, or that are from another node.

Operating system interoperability

The Tivoli Storage Manager API supports cross-platform interoperability. Applications on a UNIX or Linux system can operate on file spaces and objects that are backed up from a Windows system, and a Windows system can operate on a UNIX or Linux system.

To achieve interoperability, perform the following setup tasks:

1. Establish a consistent naming convention. Select a character for the dir delimiter and place it in front of the file space name, the high-level qualifier, and the low-level qualifier.
2. When calling **dsmInitEx**, set the value of the **dirDelimiter** field to the character that you selected: for example, / or \ and set **bCrossPlatform** to **bTrue**.
3. Set the **useUnicode** flag to **bFalse** when you use the Tivoli Storage Manager interface. Unicode filenames and non-Unicode filenames do not interoperate.

Backing up multiple nodes with client node proxy support

Backups of multiple nodes which share storage can be consolidated to a common target nodename on the Tivoli Storage Manager server. This is useful when the machine responsible for performing the backup can change over time, such as with a cluster. The *asnodename* option also allows data to be restored from a different system than the one which performed the backup.

Use the *asnodename* option on the **dsmInitEx** option string to backup, archive, restore, and retrieve, query or delete data under the target node name on the Tivoli Storage Manager server. You can also specify the *asnodename* option in the dsm.opt or dsm.sys file.

Recommendation: Do not use target nodes as traditional nodes, especially if you encrypt your files before backing them up to the server.

For more information, see the *IBM Tivoli Storage Manager Backup-Archive Clients Installation and User's Guide* for your operating system.

To enable this option, follow these steps:

1. Install the API client on all nodes in a shared data environment.
2. If not already registered, register each node with the Tivoli Storage Manager server. Register the common ″target″ nodename to be shared by each of the agent nodes used in your shared data environment.
3. Register each of the agent nodes in the shared data environment with the Tivoli Storage Manager server. This is the agent nodename which is used for authentication purposes. Data is not be stored using the agent nodename when the *asnodename* option is used.
4. Ask your Tivoli Storage Manager administrator to grant proxy authority to all nodes in the shared environment to access the target node name on the Tivoli Storage Manager server, using the **grant proxynode** command.
5. Use the **query proxynode** administrative client command to display the client nodes, granted by the **grant proxynode** command. Or use the **dsmQuery** command with the query type **qtProxyNodeAuth** to see the nodes to which this node can proxy
6. If the application is using user encryption (not TSMENCRKEY) of data, make sure that all nodes are using the exact encryption key. You must use the same encryption key for all files backed up in the shared node environment.

Chapter 5. Using the API with Unicode

The Tivoli Storage Manager API supports Unicode UCS2, a fixed length, double-byte code page that has code points for all known code pages, such as Japanese, Chinese, or German. It supports as many as 65,535 unique code points.

Note: This feature is only available on Windows.

With Unicode, your application can back up and restore file names in any character set from the same machine. For example, on an English machine, you can back up and restore file names in any other language code page.

When you should use Unicode

You can simplify your application that supports multiple languages by writing a Unicode application and by taking advantage of the Tivoli Storage Manager Unicode interface.

Use the Tivoli Storage Manager Unicode interface if any of the following conditions are true.

- If your application is already compiled for Unicode and it was converting to a multibyte character set (mbcs) before calling the Tivoli Storage Manager API.
- If you are writing a new application and want to enable your application to support Unicode.
- If your application uses a string passed to it from an operating system or other application that uses Unicode.

If you do not need Unicode, it is not necessary to compile your application again.

The API continues to support the dsm interface. The API SDK contains callmtu1.c and callmtu2.c sample programs that demonstrate how to use the Unicode API. Use **makemtu** to compile these programs.

Setting up Unicode

To set up and use Unicode you must perform a particular procedure so the API registers a Unicode file space on the server and all file names in that file space become Unicode strings.

Note: You cannot store Unicode and non-Unicode file names in the same file space.

1. Compile the code with the -DUNICODE flag.
2. All strings in your application must be wchar strings.
3. Follow the structures in the tsmapitd.h file, and the function definitions in the tsmapifp.h file for calls to the API.
4. Set the *useUnicode* flag to *bTrue* on the **tsmInitEx** function call. Any new file space is registered as a Unicode file space.

When you send data to previously registered, non-Unicode file spaces, the API continues to send file names as non-Unicode. Rename the old file spaces on the server to fsname_old and start a new Unicode file space for new data. The API restores non-Unicode data from the old file spaces. Use the **bIsUnicode** field in the **tsmQryRespFSData** structure that is returned on a query file space to determine whether or not a file space is Unicode.

Each **dsmXXX** function call has a matching **tsmXXX** function call. The difference between the two are the structures that are used. All tsm structures have dsChar_t types for string values when they are compiled with the UNICODE flag. The dsChar_r maps to wchar. There is no other difference between these interfaces.

Note: Use either one interface or the other. Do not mix the dsm and tsm interfaces. Ensure that you use the Tivoli Storage Manager structures and Tivoli Storage Manager version definitions.

Some constants continue to be defined in the dsmapitd.h file, so you need both the dsmapitd.h and the tsmapitd.h files when you compile.

You can use the Tivoli Storage Manager interface on other operating systems, such as UNIX or Linux, but on these operating systems, the dsChar_t type maps to char because Unicode is supported on Windows only. You can write only one variation of the application and compile on more than one operating system using the Tivoli Storage Manager interface. If you are writing a new application, use the Tivoli Storage Manager interface.

If you are upgrading an existing application:
1. Convert the dsm structures and calls to the Tivoli Storage Manager interface
2. Migrate existing file spaces
3. Back up new file spaces with the *useUnicode* flag set to *true*.

Note: After you use a Unicode-enabled client to access a node, you cannot connect to the same node name with an older version of the API or with an API from another operating system. If your application uses cross-platform capability, do not use the Unicode flag. There is no cross-platform support between Unicode and non-Unicode operating systems.

When you enable the *useUnicode* flag, all string structures are treated as Unicode strings. On the server, only the following fields are true Unicode:
- File space name
- High level
- Low level
- Archive description

All remaining fields convert to mbcs in the local code page before they are sent to the server. Fields, such as nodename, are wchar strings. They must be valid in the current locale. For example, on a Japanese machine, you can back up files with Chinese names, but the node name must be a valid string in Japanese. The option file remains in the current code page. If you need to create a Unicode include-exclude list, use the *inclexcl* option with a file name and create a Unicode file with Unicode patterns in it. For more information, see the *Tivoli Storage Manager Installing and Using the Backup-Archive Client* for your operating system.

Chapter 6. API function calls

Table 24 provides an alphabetical list of the API function calls, a brief description and the location of more detailed information about the function call, which includes:

Element	Description
Purpose	Describes the function call.
Syntax	Contains the actual C code for the function call. This code is copied from the UNIX or Linux version of the dsmapifp.h header file. See Appendix D, “API function definitions source file,” on page 179. This file differs slightly on other operating systems. Application programmers for other operating systems should check their version of the header file, dsmapifp.h, for the exact syntax of the API definitions.
Parameters	Describes each parameter in the function call, identifying it as either input (I) or output (O), depending on how it is used. Some parameters are designated as both input and output (I/O). The data types that are referenced in this section are defined in the dsmapitd.h header file. See Appendix C, “API type definitions source files,” on page 143.
Return codes	Contains a list of the return codes that are specific to the function call. General system errors, such as communication errors, server problems, or user errors that might appear on any call are not listed. The return codes are defined in the dsmrc.h header file. For a complete list of all the return codes with explanations, see ″Appendix A. API return codes with explanations″.

Table 24. API function calls

Function call and location	Description
“dsmBeginGetData” on page 73	Starts a restore or retrieve operation on a list of objects in storage.
“dsmBeginQuery” on page 75	Starts a query request to Tivoli Storage Manager for information.
“dsmBeginTxn” on page 78	Starts one or more transactions that begins a complete action. Either all of the actions succeed, or none succeed.
“dsmBindMC” on page 79	Associates, or binds, a management class to the object that is passed.
“dsmChangePW” on page 80	Changes a Storage Manager password.
“dsmCleanUp” on page 81	This call is used if **dsmSetUp** was called.
“dsmDeleteAccess” on page 81	Deletes current authorization rules for backup versions or archived copies of your objects.
“dsmDeleteFS” on page 82	Deletes a file space from storage.
“dsmDeleteObj” on page 83	Turns off backup objects, or deletes archive objects in storage.
“dsmEndGetData” on page 84	Ends a **dsmBeginGetData** session that gets objects from storage.
“dsmEndGetDataEx” on page 85	Provides the total of LAN-free bytes that were sent.
“dsmEndGetObj” on page 85	Ends a **dsmGetObj** session that obtains data for a specified object.

Table 24. API function calls (continued)

Function call and location	Description
"dsmEndQuery" on page 86	Signifies the end of a **dsmBeginQuery** action.
"dsmEndSendObj" on page 86	Indicates the end of data that is sent to storage.
"dsmEndSendObjEx" on page 87	Provides compression information and the number of bytes that were sent.
"dsmEndTxn" on page 87	Ends a Storage Manager transaction.
"dsmEndTxnEx" on page 89	Provides group leader object ID information to use with the **dsmGroupHandlerfunction** call.
"dsmGetData" on page 90	Obtains a byte stream of data from Tivoli Storage Manager and place it in the caller's buffer.
"dsmGetBufferData" on page 91	Gets a Tivoli Storage Manager-allocated buffer of data from the Tivoli Storage Manager server.
"dsmGetNextQObj" on page 92	Gets the next query response from a previous **dsmBeginQuery** call and places it in the caller's buffer.
"dsmGetObj" on page 94	Obtains the requested object data from the data stream and places it in the caller's buffer.
"dsmGroupHandler" on page 95	Performs an action on a logical file group depending on the input that is given.
"dsmInit" on page 96	Starts an API session and connects the client to storage.
"dsmInitEx" on page 99	Starts an API session using the additional parameters that permit extended verification.
"dsmLogEvent" on page 103	Logs a user message to the server log file, to the local error log, or to both.
"dsmLogEventEx" on page 103	Logs a user message to the server log file, to the local error log, or to both.
"dsmQueryAccess" on page 105	Queries the server for all access authorization rules for either backup versions or archived copies of your objects.
"dsmQueryApiVersion" on page 105	Performs a query request for the API library version that the application client accesses.
"dsmQueryApiVersionEx" on page 106	Performs a query request for the API library version that the application client accesses.
"dsmQueryCliOptions" on page 107	Queries important option values in the user's option files.
"dsmQuerySessInfo" on page 107	Starts a query request to Storage Manager for information that is related to the operation of the specified session in **dsmHandle**.
"dsmQuerySessOptions" on page 108	Queries important option values that are valid in the specified session in **dsmHandle**.
"dsmRCMsg" on page 109	Obtains the message text that is associated with an API return code.
"dsmRegisterFS" on page 110	Registers a new file space with the server.
"dsmReleaseBuffer" on page 111	Returns a Tivoli Storage Manager-allocated buffer.
"dsmRenameObj" on page 112	Renames the high-level or low-level object name.
"dsmRequestBuffer" on page 113	Obtains a Tivoli Storage Manager-allocated buffer for buffer copy elimination.

Table 24. API function calls (continued)

Function call and location	Description
"dsmRetentionEvent" on page 114	Sends a list of object IDs to the server with a retention event operation to be performed on these objects.
"dsmSendBufferData" on page 115	Sends data from a Tivoli Storage Manager-allocated buffer.
"dsmSendData" on page 116	Sends a byte stream of data to Storage Manager via a buffer.
"dsmSendObj" on page 117	Starts a request to send a single object to storage.
"dsmSetAccess" on page 120	Gives other users, or nodes, access to backup versions or archived copies of your objects, access to all your objects, or access to a selective set.
"dsmSetUp" on page 121	Overwrites environment variable values.
"dsmTerminate" on page 123	Ends a session with the server and cleans up the Storage Manager environment.
"dsmUpdateFS" on page 123	Updates a file space in storage.
"dsmUpdateObj" on page 124	Updates the objInfo information that is associated with an active backup object already on the server, or it updates archived objects.
"dsmUpdateObjEx" on page 125	Updates the objInfo information that is associated with a specific archive object even when there are multiple objects with same name, or it updates active backup objects.

dsmBeginGetData

The **dsmBeginGetData** function call starts a restore or retrieve operation on a list of objects in storage. This list of objects is contained in the **dsmGetList** structure. The application creates this list with values from the query that preceded a call to **dsmBeginGetData**.

The caller first must use the restore order fields that are obtained from the object query to sort the list that is contained in this call. This ensures that the objects are restored from storage in the most efficient way possible without rewinding or remounting data tapes.

When getting whole objects, the maximum *dsmGetList.numObjID* is DSM_MAX_GET_OBJ. When getting partial objects, the maximum is DSM_MAX_PARTIAL_GET_OBJ.

Follow the call to **dsmBeginGetData** with one or more calls to **dsmGetObj** to obtain each object within the list. After each object is obtained, or additional data for the object is not needed, the **dsmEndGetObj** call is sent.

When all objects are obtained, or the **dsmEndGetObj** is canceled, the **dsmEndGetData** call is sent. You then can start the cycle again.

Syntax

```
dsInt16_t dsmBeginGetData (dsUint32_t       dsmHandle,
   dsBool_t     mountWait,
   dsmGetType   getType,
   dsmGetList  *dsmGetObjListP);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous dsmInitEx call.

dsBool_t mountWait (I)
A Boolean true or false value indicates whether or not the application client waits for offline media to be mounted if the data that is needed is currently offline. If mountWait is true and the server device is not available, the server option, IDLETIMEOUT, determines how long the application waits.

dsmGetType getType (I)
An enumerated type consisting of gtBackup and gtArchive that indicates what type of object to get.

dsmGetList *dsmGetObjListP (I)
The structure that contains information about the objects or partial objects to restore or retrieve. The structure points to a list of object IDs and, in the case of a partial object restore or retrieve, a list of associated offsets and lengths. If your application uses the partial object restore or retrieve function, set the **dsmGetList.stVersion** field to **dsmGetListPORVersion**. In a partial object restore or retrieve, you cannot compress data while sending it. To enforce this, set **ObjAttr.objCompressed** to *bTrue*.

See Figure 19 on page 60 and Appendix C, "API type definitions source files," on page 143 for more information on this structure.

See "Partial object restore or retrieve" on page 54 for more information on partial object restore or retrieve.

Return codes

The return code numbers are provided in parentheses ( ).

Table 25. Return codes for dsmBeginGetData

Return code	Explanation
DSM_RC_ABORT_INVALID_OFFSET (33)	The offset that was specified during a partial object retrieve is greater than the length of the object.
DSM_RC_ABORT_INVALID_LENGTH (34)	The length that was specified during a partial object retrieve is greater than the length of the object, or the offset in addition to the length extends past the end of the object.
DSM_RC_NO_MEMORY (102)	There is no RAM remaining to complete the request.
DSM_RC_NUMOBJ_EXCEED (2029)	The **dsmGetList.numObjId** is greater than DSM_MAX_GET_OBJ.
DSM_RC_OBJID_NOTFOUND (2063)	The object ID was not found. The object was not restored.
DSM_RC_WRONG_VERSION_PARM (2065)	The API version of the application client is different from the Tivoli Storage Manager library version.

dsmBeginQuery

The **dsmBeginQuery** function call starts a query request to the server for information about data, file spaces, and management classes.

Specifically, **dsmBeginQuery** can query:
- Archived data
- Backed-up data
- Active backed-up data
- File spaces
- Management classes

The query data that is returned from the call is obtained by one or more calls to **dsmGetNextQObj**. When the query is complete, the **dsmEndQuery** call is sent.

Syntax

```
dsInt16_t dsmBeginQuery  (dsUint32_t       dsmHandle,
   dsmQueryType  queryType,
   dsmQueryBuff  *queryBuffer);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsmQueryType queryType (I)
Identifies the type of query to perform. Select from one of the following:

qtArchive	Queries archived objects.
qtBackup	Queries backed-up objects.
qtBackupActive	Queries active, backed-up objects only for the entire file space name that you pass. This is referred to as “fast path” and is an efficient way to query active objects from storage. **Note:** You must be a root user on a UNIX or Linux operating system.
qtFilespace	Queries registered file spaces.
qtMC	Queries defined management classes.
qtBackupGroups	Queries groups that are closed.
qtOpenGroups	Queries groups that are open.
qtProxyNodeAuth	Queries nodes to which this node can proxy.
qtProxyNodePeer	Queries peer nodes with the same target.

dsmQueryBuff *queryBuffer (I)
Identifies a pointer to a buffer that is mapped to a particular data structure. This structure is associated with the query type that you pass. These structures contain the selection criteria for each query type. Complete the fields in each structure to specify the scope of the query that you want to perform. The first field of each structure is **stVersion**, the structure version number.

The data structures and their related fields include:

qryArchiveData:

objName	The complete object name. You can use a wildcard character, such as an asterisk (*) or question mark (?), in the high- or low-level portion of the name. See “High-level and low-level names” on page 26. An asterisk matches zero or more characters, and a question mark matches exactly one character. The objType field of the objName can be DSM_OBJ_FILE, DSM_OBJ_DIRECTORY, or DSM_OBJ_ANY_TYPE.
owner	The owner name of the object.
insDateLowerBound	The lower boundary for the archive insert date (the date the object was archived). To obtain the default lower boundary, set the year component to DATE_MINUS_INFINITE.
insDateUpperBound	The upper boundary for the archive insert date (the date the object was archived). To obtain the default upper boundary, set the year component to DATE_PLUS_INFINITE.
expDateLowerBound	The lower boundary for the expiration date. The default values for both expiration date fields are the same as for the insert date fields.
expDateUpperBound	The upper boundary for the expiration date.
descr	The archive description. Enter an asterisk (*) to search on all descriptions.

qryBackupData:

objName	The complete object name. You can use a wildcard character, such as an asterisk (*) or question mark (?) in the high- or low-level portion of the name. See “High-level and low-level names” on page 26. An asterisk matches zero or more characters, and a question mark matches exactly one character. The objType field of objName can be DSM_OBJ_FILE, DSM_OBJ_DIRECTORY, or DSM_OBJ_ANY_TYPE.
owner	The object owner name.
objState	This field can have one of three values: DSM_ACTIVE, DSM_INACTIVE, or DSM_ANY_MATCH.
pitDate	The point-in-time value. A query using this field returns the latest object that was backed up before this date and time. The **objState** can be active or inactive. Objects that were deleted before the **pitDate** are not be returned. For example:

```
Mon - backup ABC(1), DEF, GHI
Tue - backup ABC(2), delete DEF
Thr - backup ABC(3)
```

On Friday, call the query with a point-in-time value of Wednesday at 12:00:00 a.m. It returns the following information:

```
ABC(2) - an Inactive copy
GHI    - an Active copy
```

It does not return DEF because it was deleted.

qryABackupData:

	objName	The complete object name. You can use a wildcard character, such as an asterisk (*) or question mark (?) in the high- or low-level portion of the name. See “High-level and low-level names” on page 26. An asterisk matches zero or more characters, and a question mark matches exactly one character. The objType field of objName can be DSM_OBJ_FILE, DSM_OBJ_DIRECTORY or DSM_OBJ_ANY_TYPE.
qryFSData:	**fsName**	Enter the name of a specific file space in this field, or enter an asterisk (*) to retrieve information about all registered file spaces.
qryMCData:	**mcName**	Enter the name of a specific management class, or enter an empty string (“ ”) to retrieve information about all management classes. **Note:** You cannot use an asterisk (*).
	mcDetail	This field has a value of *bTrue* or *bFalse*. The value determines whether information on the backup and archive copy groups of the management class is returned.

qryBackupGroup:

groupType	The group type is DSM_GROUPTYPE_PEER
fsName	The File Space name
owner	The owner ID
groupLeaderObjId	The group leader object ID
objType	The object type

qryProxyNodeAuth:

targetNodeName	The target node name
peerNodeName	The peer node name
hlAddress	The peer address of the high level name
llAddress	The peer address of the low level name

qryProxyNodePeer:

targetNodeName	The target node name
peerNodeName	The peer node name
hlAddress	The peer address of the high level name
llAddress	The peer address of the low level name

Return codes

The return code numbers are provided in parentheses ( ).

Table 26. Return codes for dsmBeginQuery

Return code	Explanation
DSM_RC_NO_MEMORY (102)	There is no RAM remaining to complete the request.
DSM_RC_FILE_SPACE_NOT_FOUND (124)	The specified file space was not found.
DSM_RC_NO_POLICY_BLK (2007)	Server policy information was not available.
DSM_RC_INVALID_OBJTYPE (2010)	Invalid object type.

Table 26. Return codes for dsmBeginQuery (continued)

Return code	Explanation
DSM_RC_INVALID_OBJOWNER (2019)	Invalid object owner name.
DSM_RC_INVALID_OBJSTATE (2024)	Invalid object condition.
DSM_RC_WRONG_VERSION_PARM (2065)	The API version of the application client is different from the Tivoli Storage Manager library version.

dsmBeginTxn

The **dsmBeginTxn** function call begins one or more Tivoli Storage Manager transactions that begin a complete action; either all the actions succeed or none succeed. An action can be either a single call or a series of calls. For example, a **dsmSendObj** call that is followed by a number of **dsmSendData** calls can be considered a single action. Similarly, a **dsmSendObj** call with a **dataBlkPtr** that indicates a data area containing the object to back up is also considered a single action.

Try to group more than one object together in a single transaction for data transfer operations. Grouping objects results in significant performance improvements in the Tivoli Storage Manager system. From both a client and a server perspective, a certain amount of overhead is incurred by starting and ending each transaction.

There are limits to what you can perform within a single transaction. These restrictions include:

- A maximum number of objects that you can send or delete in a single transaction. This limit is located in the data that **dsmQuerySessInfo** returns in the *ApiSessInfo.maxObjPerTxn* field. This corresponds to the *TxnGroupMax* server option.
- All objects that are sent to the server (either backup or archive) within a single transaction must have the same copy destination that is defined in the management class binding for the object. This value is located in the data that **dsmBindMC** returns in the **mcBindKey.backup_copy_dest** or **mcBindKey.archive_copy_dest** fields.

With the API, either the application client can monitor and control these restrictions, or the API can monitor these restrictions. If the API is monitoring restrictions, appropriate return codes from the API calls inform the application client when one or more restrictions are reached.

Always match a **dsmBeginTxn** call with a **dsmEndTxn** call to optimize the set of actions within a pair of **dsmBeginTxn** and **dsmEndTxn** calls.

Syntax

```
dsInt16_t dsmBeginTnx  (dsUint32_t dsmHandle);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

Return codes

The return code numbers are provided in parentheses ( ).

Table 27. Return codes for dsmBeginTxn

Return code	Explanation
DSM_RC_ABORT_NODE_NOT_AUTHORIZED (36)	FROMNODE or FROMOWNER is not allowed for TXN operations.

dsmBindMC

The **dsmBindMC** function call associates, or binds, a management class to the passed object. The object is passed through the ixclude-exclude list that is pointed to in the options file. If a match is not found in the Include list for a specific management class, the default management class is assigned. The Exclude list can prevent objects from a backup but not from an archive.

The application client can use the parameters that are returned in the `mcBindKey` structure to determine if this object should be backed up or archived, or whether a new transaction must be started because of different copy destinations. See **dsmBeginTxn** for more information.

Call **dsmBindMC** before you call **dsmSendObj** because every object must have a management class associated with it. This call can be performed within a transaction or outside of a transaction. For example, within a multiple object transaction, if **dsmBindMC** indicates that the object has a different copy destination than the previous object, the transaction must be ended and a new transaction started. In this case, another **dsmBindMC** is not required because one has already been performed for this object.

Syntax

```
dsInt16_t dsmBindMC  (dsUint32_t       dsmHandle,
   dsmObjName  *objNameP,
   dsmSendType  sendType,
   mcBindKey   *mcBindKeyP);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsmObjName *objNameP (I)
A pointer to the structure that contains the file space name, high-level object name, low-level object name, and object type.

dsmSendType sendType (I)
Identifies whether this management class bind is performed for archive or backup sends. The possible values for this call include:

Name	Description
stBackup	A backup object
stArchive	An archive object
stBackupMountWait	A backup object
stArchiveMountWait	An archive object

For the **dsmBindMC** call, `stBackup` and `stBackupMountWait` are equivalent, and `stArchive` and `stArchiveMountWait` are equivalent.

mcBindKey *mcBindKeyP (O)
This is the address of an `mcBindKey` structure where the management class information is returned. The application client can use the information that is

returned here to determine if this object fits within a multiple object transaction, or to perform a management class query on the management class that is bound to the object.

Return codes

The return code numbers are provided in parentheses ( ).

Table 28. Return codes for dsmBindMC

Return code	Explanation
DSM_RC_NO_MEMORY (102)	There is no RAM remaining to complete the request.
DSM_RC_INVALID_PARM (109)	One of the parameters that was passed has an invalid value.
DSM_RC_TL_EXCLUDED (185)	The backup object is excluded and cannot be sent.
DSM_RC_INVALID_OBJTYPE (2010)	Invalid object type.
DSM_RC_INVALID_SENDTYPE (2022)	Invalid send type.
DSM_RC_WRONG_VERSION_PARM (2065)	Application client API version is different from the Tivoli Storage Manager library version.

dsmChangePW

The **dsmChangePW** function call changes a Tivoli Storage Manager password. On a multiple-user operating system such as UNIX or Linux, only the root user or the TSM-Authorized user can use this call.

On the Windows and Novell operating systems, you can specify the password in the dsm.opt file. In this situation, **dsmChangePW** does not update the dsm.opt file. After the call to **dsmChangePW** is made, you must update the dsm.opt file separately.

This call must process successfully if **dsmInitEx** returns DSM_RC_VERIFIER_EXPIRED. The session ends if the **dsmChangePW** call fails in this situation.

If **dsmChangePW** is called for some other reason, the session remains open regardless of the return code.

Syntax

```
dsInt16_t dsmChangePW  (dsUint32_t   dsmHandle,
   char    *oldPW,
   char    *newPW);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

char *oldPW (I)
The old password of the caller. The maximum length is DSM_MAX_VERIFIER_LENGTH.

char *newPW (I)
The new password of the caller. The maximum length is DSM_MAX_VERIFIER_LENGTH.

Return codes

The return code numbers are provided in parentheses ( ).

Table 29. Return codes for dsmChangePW

Return code	Explanation
DSM_RC_ABORT_BAD_VERIFIER (6)	An incorrect password was entered.
DSM_RC_AUTH_FAILURE (137)	Authentication failure. Old password is incorrect.
DSM_RC_NEWPW_REQD (2030)	A value must be entered for the new password.
DSM_RC_OLDPW_REQD (2031)	A value must be entered for the old password.
DSM_RC_PASSWD_TOOLONG (2103)	The specified password is too long.
DSM_RC_NEED_ROOT (2300)	The API caller must be a root user or a TSM-Authorized user.

dsmCleanUp

The **dsmCleanUp** function call is used if **dsmSetUp** was called. The **dsmCleanUp** function call should be called after **dsmTerminate**. You cannot make any other calls after you call **dsmCleanUp**.

There are no return codes that are specific to this call.

Syntax

```
dsInt16_t DSMLINKAGE dsmCleanUp
   (dsBool_t       mtFlag);
```

Parameters

dsBool_t mtFlag (I)
This parameter specifies that the API was used either in a single thread or a multithread mode. Possible values include:

- DSM_SINGLETHREAD
- DSM_MULTITHREAD

dsmDeleteAccess

The **dsmDeleteAccess** function call deletes current authorization rules for backup versions or archived copies of your objects. When you delete an authorization rule, you revoke the access a user has to any files that are specified by the rule.

When you use **dsmDeleteAccess**, you can only delete one rule at a time. Obtain the rule ID through the **dsmQueryAccess** command.

There are no return codes that are specific to this call.

Syntax

```
dsInt16_t DSMLINKAGE dsmDeleteAccess
           (dsUint32_t         dsmHandle,
            dsUint32_t         ruleNum) ;
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous dsmInitEx call.

dsUint32_t ruleNum (I)
The rule ID for the access rule that is deleted. This value is obtained from a **dsmQueryAccess** function call.

dsmDeleteFS

The **dsmDeleteFS** function call deletes a file space from storage. To delete a file space, you must have the appropriate permissions that your Tivoli Storage Manager administrator gave you. To determine whether you have the necessary permissions, call **dsmQuerySessInfo**. This function call returns a data structure of type *ApiSessInfo*, that includes two fields, *archDel* and *backDel*.

Note:

- On a UNIX or Linux operating system, only a root user or a TSM-Authorized user can delete a file space.
- If the file space that you need to delete contains backup versions, you must have backup delete authority (***backDel*** = BACKDEL_YES). If the file space contains archive copies, you must have archive delete authority (*archDel* = ARCHDEL_YES). If the file space contains both backup versions and archive copies, you must have both types of delete authority.
- When using an archive manager server, a file space cannot actually be removed. This function call returns *rc=0* even though the file space was not actually deleted. The only way to verify that the file space has been deleted is to issue a filespace query to the server.
- The Tivoli Storage Manager server delete file space function is a background process. If errors other than those detected before passing a return code happen, they are recorded in the Tivoli Storage Manager server log.

Syntax

```
dsInt16_t dsmDeleteFS  (dsUint32_t          dsmHandle,
   char           *fsName,
   unsigned char   repository);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

char *fsName (I)
A pointer to the file space name to delete. The wildcard character is not permitted.

unsigned char repository (I)
Indicates whether the file space to delete is a backup repository, archive repository, or both. The possible values for this field include:

```
DSM_ARCHIVE_REP     /* archive repository   */
DSM_BACKUP_REP      /* backup repository    */
DSM_REPOS_ALL       /* all repository types */
```

Return codes

The return code numbers are provided in parentheses ( ).

Table 30. Return codes for dsmDeleteFS

Return code	Explanation
DSM_RC_ABORT_NOT_AUTHORIZED (27)	You do not have the necessary authority to delete the file space.
DSM_RC_INVALID_REPOS (2015)	Invalid value for repository.
DSM_RC_FSNAME_NOTFOUND (2060)	File space name not found.
DSM_RC_NEED_ROOT (2300)	API caller must be a root user.

dsmDeleteObj

The **dsmDeleteObj** function call inactivates backup objects, deletes backup objects, or it deletes archive objects in storage. The **dtBackup** type inactivates the currently active backup copy only. The **dtBackupID** type removes from the server whichever object ID is specified. Call this function from within a transaction.

See **dsmBeginTxn** for more information.

Before you send **dsmDeleteObj**, send the query sequence that is described in "Querying the Tivoli Storage Manager system" on page 35 to obtain the information for **delInfo**. The call to **dsmGetNextQObj** returns a data structure named **qryRespBackupData** for backup queries or **qryRespArchiveData** for archive queries. These data structures contain the information that you need for **delInfo**.

The value of **maxObjPerTxn** determines the maximum number of objects that you can delete in a single transaction. To obtain this value, call **dsmQuerySessInfo**.

Note: Your node must have the appropriate permission that your Tivoli Storage Manager administrator set. To delete archive objects, you must have archive delete authority. You do not need backup delete authority to inactivate a backup object.

Syntax

```
dsInt16_t dsmDeleteObj  (dsUint32_t      dsmHandle,
   dsmDelType  delType,
   dsmDelInfo  delInfo)
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsmDelType delType (I)
Indicates what type of object (backup or archive) to delete. Possible values include:

Name	Description
dtArchive	The object to delete was previously archived. To use this delete type, you must have a Tivoli Storage Manager server, V3.7.4 or later.
dtBackup	The object to inactivate was previously backed up. To use this delete type, you must have a Tivoli Storage Manager server, V3.7.3 or later.

Name	Description
dtBackupID	The object to delete was previously backed up. To use this delete type, you must have a Tivoli Storage Manager server, V3.7.3 or later. **Attention:** Using this **delType** with *objID* removes the backup object from the server. Only an owner of an object can delete it. You can delete any version (active or inactive) of an object. The server reconciles the versions. If you delete an active version of an object, the first inactive version becomes active. If you delete an inactive version of an object, all older versions will advance. The node must be registered with **backDel** permission.

dsmDelInfo delInfo (I)

A structure whose fields identify the object. The fields are different, depending on whether the object is a backup object or an archive object. The structure to inactivate a backup object, `delBack`, contains the object name and the object copy group. The structure for an archive object, `delArch`, contains the object ID.

The structure to remove a backup object, `delBackID`, contains the object ID.

Return codes

The return code numbers are provided in parentheses ( ).

Table 31. Return codes for dsmDeleteObj

Return code	Explanation
DSM_RC_FS_NOT_REGISTERED (2061)	File space name is not registered.
DSM_RC_WRONG_VERSION_PARM (2065)	Application client API version is different from the Tivoli Storage Manager library version.

dsmEndGetData

The **dsmEndGetData** function call ends a **dsmBeginGetData** session that obtains objects from storage.

The **dsmEndGetData** function call starts after all objects that you want to restore are processed, or ends the get process prematurely. Call **dsmEndGetData** to end a **dsmBeginGetData** session before you can continue other processing.

Depending on when **dsmEndGetData** is called, the API might need to finish processing a partial data stream before the process can be stopped. The caller, therefore, should not expect an immediate return from this call. Use **dsmTerminate** if the application needs to close the session and end the restore immediately.

There are no return codes that are specific to this call.

Syntax

```
dsInt16_t dsmEndGetData  (dsUint32_t dsmHandle);
```

Parameters

dsUint32_t dsmHandle (I)

The handle that associates this call with a previous **dsmInitEx** call.

dsmEndGetDataEx

The **dsmEndGetDataEx** function call provides the total of LAN-free bytes that were sent. It is an extension of the **dsmEndGetData** function call.

Syntax

There are no return codes that are specific to this call.

```
dsInt16_t dsmEndGetDataEx (dsmEndGetDataExIn_t * dsmEndGetDataExInP,
                           dsmEndGetDataExOut_t * dsmEndGetDataExOutP);
```

Parameters

dsmEndGetDataExIn_t *dsmEndGetDataExInP (I)
Passes the end get object dsmHandle that identifies the session and associates it with subsequent calls.

dsmEndGetDataExOut_t *dsmEndGetDataExOutP (O)
This structure contains this input parameter:

totalLFBytesRecv
The total LAN-free bytes that are received.

dsmEndGetObj

The **dsmEndGetObj** function call ends a **dsmGetObj** session that obtains data for a specified object.

Start the **dsmEndGetObj** call after an end of data is received for the object. This indicates that all data was received, or that no more data will be received for this object. Before you can start another **dsmGetObj** call, you must call **dsmEndGetObj**.

Depending on when **dsmEndGetObj** is called, the API might need to finish processing a partial data stream before the process can stop. Do not expect an immediate return from this call.

Syntax

```
dsInt16_t dsmEndGetObj  (dsUint32_t dsmHandle);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

Return codes

The return code numbers are provided in parentheses ( ).

Table 32. Return codes for dsmEndGetObj

Return code	Explanation
DSM_RC_NO_MEMORY (102)	There is no RAM remaining to complete the request.

dsmEndQuery

The **dsmEndQuery** function call signifies the end of a **dsmBeginQuery** action. The application client sends **dsmEndQuery** to complete a query. This call either is sent after all query responses are obtained through **dsmGetNextQObj**, or it is sent to end a query before all data are returned.

Note: The Tivoli Storage Manager continues to send the query data from the server to the client in this case, but the API discards any remaining data.

Once a **dsmBeginQuery** is sent, a **dsmEndQuery** must be sent before any other activity can start.

There are no return codes that are specific to this call.

Syntax

```
dsInt16_t dsmEndQuery  (dsUint32_t dsmHandle);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsmEndSendObj

The **dsmEndSendObj** function call indicates the end of data that is sent to storage.

Enter the **dsmEndSendObj** function call to indicate the end of data from the **dsmSendObj** and **dsmSendData** calls. A protocol violation occurs if this is not performed. The exception to this rule is if you call **dsmEndTxn** to end the transaction. Doing this discards all data that was sent for the transaction.

Syntax

```
  dsInt16_t dsmEndSendObj  (dsUint32_t dsmHandle);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

Return codes

The return code numbers are provided in parentheses ( ).

Table 33. Return codes for dsmEndSendObj

Return code	Explanation
DSM_RC_NO_MEMORY (102)	There is no RAM remaining to complete this request.

dsmEndSendObjEx

The **dsmEndSendObjEx** function call provides compression information and the number of bytes that were sent. It is an extension of the **dsmEndSendObj** function call.

Syntax

```
dsInt16_t dsmEndSendObjEx (dsmEndSendObjExIn_t  *dsmEndSendObjExInP,
                           dsmEndSendObjExOut_t *dsmEndSendObjExOutP);
```

Parameters

dsmEndSendObjExIn_t *dsmEndSendObjExInP (I)
This parameter passes the end send object dsmHandle that identifies the session and associates it with subsequent calls.

dsmEndSendObjExOut_t *dsmEndSendObjExOutP (O)
This parameter passes the end send object information:

Name	Description
totalBytesSent	The total number of bytes that are read from the application.
objCompressed	A flag that displays if the object was compressed.
totalCompressedSize	The total byte size after compression.
totalLFBytesSent	The total LAN-free bytes that were sent.

Return codes

The return code numbers are provided in parentheses ( ).

Table 34. Return codes for dsmEndSendObjEx

Return code	Explanation
DSM_RC_NO_MEMORY (102)	There is no RAM remaining to complete this request.

dsmEndTxn

The **dsmEndTxn** function call ends a Tivoli Storage Manager transaction. Pair the **dsmEndTxn** function call with **dsmBeginTxn** to identify the call or set of calls that are considered a transaction. The application client can specify on the dsmEndTxn call whether or not the transaction should be committed or ended.

Perform all of the following calls within the bounds of a transaction:

- **dsmSendObj**
- **dsmSendData**
- **dsmEndSendObj**
- **dsmDeleteObj**

Syntax

```
dsInt16_t dsmEndTxn (dsUint32_t   dsmHandle,
   dsUint8_t    vote,
   dsUint16_t  *reason);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsUint8_t vote (I)

Indicates whether or not the application client commits all the actions that are done between the previous **dsmBeginTxn** call and this call. The possible values are:

```
DSM_VOTE_COMMIT     /* commit current transaction    */
DSM_VOTE_ABORT      /* roll back current transaction */
```

Use DSM_VOTE_ABORT only if your application has found a reason to stop the transaction.

dsUint16_t *reason (O)

If the call to dsmEndTxn ends with an error, or the value of `vote` is not agreed to, this parameter has a reason code indicating why the vote failed.

Note: The return code for the call might be zero, and the reason code might be non-zero. Therefore, the application client must always check for errors on both the return code and the reason (`if (rc || reason)`) before you can assume a successful completion.

If the application specifies a vote of `DSM_VOTE_ABORT`, the reason code is DSM_RS_ABORT_BY_CLIENT (3). See Appendix B, “API return codes source file dsmrc.h,” on page 131 for a list of the possible reason codes. Numbers 1 through 50 in the return codes list are reserved for the reason codes. If the server ends the transaction, the return code is DSM_RC_CHECK_REASON_CODE. In this case, the reason value contains more information on the cause of the abort.

Return codes

The return code numbers are provided in parentheses ( ).

Table 35. Return codes for dsmEndTxn

Return code	Explanation
DSM_RC_ABORT_CRC_FAILED (236)	The CRC that was received from the server does not match the CRC that was calculated by the client.
DSM_RC_INVALID_VOTE (2011)	The value that was specified for `vote` is not valid.
DSM_RC_CHECK_REASON_CODE (2302)	The transaction was aborted. Check the reason field.
DSM_RC_ABORT_STGPOOL_COPY_CONT_NO (241)	The write to one of the copy storage pools failed, and the Tivoli Storage Manager storage pool option COPYCONTINUE was set to NO. The transaction terminates.
DSM_RC_ABORT_RETRY_SINGLE_TXN (242)	This abort code indicates that the current transaction was aborted because of a problem during a store operation. The problem could be resolved by sending each file in an individual transaction. This error is typical when the next storage pool has a different copy storage pool list and we switch to this pool in the middle of a transaction.

dsmEndTxnEx

The **dsmEndTxnEx** function call provides group leader object ID information for you to use with the **dsmGroupHandler** function call. It is an extension of the **dsmEndTxn** function call.

Syntax

```
dsInt16_t dsmEndTxnEx (dsmEndTxnExIn_t *dsmEndTxnExInP
                       dsmEndTxnExOut_t *dsmEndTxnExOutP);
```

Parameters

dsmEndTxnExIn_t *dsmEndTxnExInP (I)
This structure contains the following parameters:

dsmHandle
The handle that identifies the session and associates it with subsequent Tivoli Storage Manager calls.

dsUint8_t vote (I)
Indicates whether or not the application client commits all the actions that are done between the previous dsmBeginTxn call and this call. The possible values are:

```
DSM_VOTE_COMMIT     /* commit current transaction    */
DSM_VOTE_ABORT      /* roll back current transaction */
```

Use `DSM_VOTE_ABORT` only if your application has found a reason to stop the transaction.

dsmEndTxnExOut_t *dsmEndTxnExOutP (O)
This structure contains the following parameters:

dsUint16_t *reason (O)
If the call to **dsmEndTxnEx** ends with an error or the value of *vote* is not agreed to, this parameter has a reason code indicating why the vote failed.

Note: The return code for the call might be zero, and the reason code might be non-zero. Therefore, the application client must always check for errors on both the return code and the reason (`if (rc || reason)`) before you can assume a successful completion.

If the application specifies a vote of DSM_VOTE_ABORT, the reason code is DSM_RS_ABORT_BY_CLIENT (3). See Appendix B, “API return codes source file dsmrc.h,” on page 131 for a list of the possible reason codes. Numbers 1 through 50 in the return codes list are reserved for the reason codes. If the server ends the transaction, the return code is DSM_RC_CHECK_REASON_CODE. In this case, the reason value contains more information on the cause of the abort.

groupLeaderObjId
The group leader object ID that is returned when the DSM_ACTION_OPEN flag is used with the **dsmGroupHandler** call.

Return codes

The return code numbers are provided in parentheses ( ).

Table 36. Return codes for dsmEndTxnEx

Return code	Explanation
DSM_RC_INVALID_VOTE (2011)	The value that was specified for vote is invalid.
DSM_RC_CHECK_REASON_CODE (2302)	The transaction was aborted. Check the reason field.
DSM_RC_ABORT_STGPOOL_COPY_CONT_NO (241)	The write to one of the copy storage pools failed, and the Tivoli Storage Manager storage pool option COPYCONTINUE was set to NO. The transaction terminates.
DSM_RC_ABORT_RETRY_SINGLE_TXN (242)	During a simultaneous write operation, an object in the transaction is going to a destination with different copy storage pools. End the current transaction and send each object again in its own transaction.

dsmGetData

The **dsmGetData** function call obtains a byte stream of data from Tivoli Storage Manager and places it in the caller's buffer. The application client calls **dsmGetData** when there is more data to receive from a previous **dsmGetObj** or **dsmGetData** call.

Syntax

```
dsInt16_t dsmGetData  (dsUint32_t   dsmHandle,
   DataBlk *dataBlkPtr);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

DataBlk *dataBlkPtr (I/O)
Points to a structure that includes both a pointer to the buffer for the data that is received and the size of the buffer. On return, this structure contains the number of bytes that is actually transferred. See Appendix C, "API type definitions source files," on page 143 for the type definition.

Return codes

The return code numbers are provided in parentheses ( ).

Table 37. Return codes for dsmGetData

Return code	Explanation
DSM_RC_ABORT_INVALID_OFFSET (33)	The offset that was specified during a partial object retrieve is greater than the length of the object.
DSM_RC_ABORT_INVALID_LENGTH (34)	The length that was specified during a partial object retrieve is greater than the length of the object, or the offset in addition to the length extends beyond the end of the object.
DSM_RC_FINISHED (121)	Finished processing. The last buffer was received. Check `numBytes` for the amount of data and then call Tivoli Storage ManagerdsmEndGetObj.
DSM_RC_NULL_DATABLKPTR (2001)	Datablock pointer is null.
DSM_RC_ZERO_BUFLEN (2008)	Buffer length is zero for datablock pointer.
DSM_RC_NULL_BUFPTR (2009)	Buffer pointer is null for datablock pointer.
DSM_RC_WRONG_VERSION_PARM (2065)	The application client's API version is different from the Tivoli Storage Manager library version.

Table 37. Return codes for dsmGetData (continued)

Return code	Explanation
DSM_RC_MORE_DATA (2200)	There is more data to get.

dsmGetBufferData

The **dsmGetBufferData** function call receives a byte stream of data from the Tivoli Storage Manager through a Tivoli Storage Manager buffer. After each call the application needs to copy the data and release the buffer through a call to **dsmReleaseBuffer**. If the number of buffers held by the application equals the numTsmBuffers specified in the **dsmInitEx** call, the **dsmGetBufferData** function blocks until a **dsmReleaseBuffer** is called.

Syntax

```
dsInt16_t  dsmGetBufferData   (getDatatExIn_t        *dsmGetBufferDataExInP,
                               getDataExOut_t        *dsmGetBufferDataExOutP) ;
```

Parameters

getDataExIn_t * dsmGetBufferDataExInP (I)
This structure contains the following input parameter.

dsUint32_t dsmHandle
The handle that identifies the session and associates it with a previous **dsmInitEx** call.

getDataExOut_t * dsmGetBufferDataExOutP (0)
This structure contains the following output parameters.

dsUint8_t tsmBufferHandle(0)
The handle that identifies the buffer received.

char *dataPtr(0)
The address to which Tivoli Storage Manager data was written.

dsUint32_t numBytes(0)
Actual number of bytes written by Tivoli Storage Manager.

Return codes

The return code numbers are provided in parentheses ( ).

Table 38. Return codes for dsmGetBufferData

Return code	Explanation
DSM_RC_BAD_CALL_SEQUENCE (2041)	The call was not issued in the proper state.
DSM_RC_OBJ_ENCRYPTED (2049)	This function cannot be used for encrypted objects.
DSM_RC_OBJ_COMPRESSED (2048)	This function cannot be used for compressed objects.
DSM_RC_BUFF_ARRAY_ERROR (2045)	A buffer array error occurred.

dsmGetNextQObj

The **dsmGetNextQObj** function call gets the next query response from a previous **dsmBeginQuery** call and places it in the caller's buffer. The **dsmGetNextQObj** call is called one or more times. Each time it is called, a single query record is retrieved. If the application client needs to end the query before retrieving all of the data, you can send a **dsmEndQuery** call.

The **dataBlkPtr** must point to a buffer that is defined with the **qryResp*Data** structure type. The context in which **dsmGetNextQObj** is called determines the type of structure that is entered on the query response.

Syntax

```
dsInt16_t dsmGetNextQObj (dsUint32_t   dsmHandle,
   DataBlk *dataBlkPtr);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

DataBlk *dataBlkPtr (I/O)
Points to a structure that includes both a pointer to the buffer for the data to be received and the size of the buffer. This buffer is the **qryResp*Data** structure that is described in Table 39. On return, this structure contains the number of bytes that is actually transferred. See Appendix C, "API type definitions source files," on page 143 for the type definition of **DataBlk**. The structure associated with each type of query is:

Table 39. DataBlk pointer structure

Query	Response structure	Fields of special interest
qtArchive	**qryRespArchiveData**	**sizeEstimate** contains the value that is passed on a previous **dsmSendObj** call.
		mediaClass can have a value of MEDIA_FIXED if the object is on disk, or MEDIA_LIBRARY if the object is on tape.
qtBackup	**qryRespBackupData**	**restoreOrderExt** is of type dsUint16_t. Sort on this field when restoring several objects on a **dsmBeginGetData** call. An example of sorting code for this is in the API sample, dapiqry.c. Also see Figure 16 on page 56 for a sorting example.
		sizeEstimate contains the value that is passed on a previous **dsmSendObj** call.
		mediaClass can have a value of MEDIA_FIXED if the object is on disk or MEDIA_LIBRARY if the object is on tape.
qtBackupActive	**qryARespBackupData**	-
qtBackupGroups	**qryRespBackupData**	**dsBool_t isGroupLeader**, if true, signifies this object is a group leader.

Table 39. DataBlk pointer structure (continued)

Query	Response structure	Fields of special interest
qtOpenGroups	**qryRespBackupData**	**dsBool_t isOpenGroup;**, if true, signifies this group is open and not complete.
qtFilespace	**qryRespFSData**	**backStartDate** contains the server's time stamp when the file space was updated with the **backStartDate** action.
		backCompleteDate contains the server time stamp when the file space was updated with the **backCompleteDate** action.
qtMC	**qryRespMCData** **qryRespMCDetailData**	-
qtProxyNodeAuth	**qryRespProxyNodeData** **targetNodeName** **peerNodeName** **hlAddress** **llAddress**	-
qtProxyNodePeer	**qryRespProxyNodeData** **targetNodeName** **peerNodeName** **hlAddress** **llAddress**	-

Return codes

The return code numbers are provided in parentheses ( ).

Table 40. Return codes for dsmGetNextQObj

Return code	Explanation
DSM_RC_ABORT_NO_MATCH (2)	No match for the query was requested.
DSM_RC_FINISHED (121)	Finished processing (start **dsmEndQuery**).
DSM_RC_UNKNOWN_FORMAT (122)	The file that Tivoli Storage Manager attempted to restore or retrieve has an unknown format.
DSM_RC_COMM_PROTOCOL_ERROR (136)	Communication protocol error.
DSM_RC_NULL_DATABLKPTR (2001)	Pointer is not pointing to a data block.
DSM_RC_INVALID_MCNAME (2025)	Invalid management class name.
DSM_RC_BAD_CALL_SEQUENCE (2041)	The sequence of calls is invalid.
DSM_RC_WRONG_VERSION_PARM (2065)	Application client's API version is different from the Tivoli Storage Manager library version.
DSM_RC_MORE_DATA (2200)	There is more data to get.
DSM_RC_BUFF_TOO_SMALL (2210)	Buffer is too small.

dsmGetObj

The **dsmGetObj** function call obtains the requested object data from the Tivoli Storage Manager data stream and places it in the caller's buffer. The **dsmGetObj** call uses the object ID to obtain the next object or partial object from the data stream.

The data for the indicated object is placed in the buffer to which **DataBlk** points. If more data is available, you must make one or more calls to **dsmGetData** to receive the remaining object data until a return code of DSM_RC_FINISHED is returned. Check the `numBytes` field in **DataBlk** to see whether any data remains in the buffer.

Objects should be asked for in the order that they were listed on the **dsmBeginGetData** call in the **dsmGetList** parameter. The exception is when the application client needs to pass over an object in the data stream to get to an object later in the list. If the object that is indicated by the object ID is not the next object in the stream, the data stream is processed until the object is located, or the stream is completed. Use this feature with care, because it might be necessary to process and discard large amounts of data to locate the requested object.

Note: If **dsmGetObj** returns a failure code (NOT FINISHED or MORE_DATA), the session needs to be terminated to abort the restore operation. This is especially important when using encryption and receiving a RC_ENC_WRONG_KEY. A new session with the proper key must be started.

Syntax

```
dsInt16_t dsmGetObj (dsUint32_t dsmHandle,
   ObjID    *objIdP,
   DataBlk  *dataBlkPtr);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

ObjID *objIdP (I)
A pointer to the ID of the object to restore.

DataBlk *dataBlkPtr (I/O)
A pointer to the buffer where the restored data are placed.

Return codes

The return code numbers are provided in parentheses ( ).

Table 41. Return codes for dsmGetObj

Return code	Explanation
DSM_RC_ABORT_INVALID_OFFSET (33)	The offset that is specified during a partial object retrieve is greater than the length of the object.
DSM_RC_ABORT_INVALID_LENGTH (34)	The length that is specified during a partial object retrieve is greater than the length of the object, or the offset in addition to the length extends past the end of the object.
DSM_RC_FINISHED (121)	Finished processing (start **dsmEndGetObj**).
DSM_RC_WRONG_VERSION_PARM (2065)	Application client's API version is different from the Tivoli Storage Manager library version.
DSM_RC_MORE_DATA (2200)	There is more data to get.

Table 41. Return codes for dsmGetObj (continued)

Return code	Explanation
RC_ENC_WRONG_KEY (4580)	The key provided in the **dsmInitEx** call, or the saved key, does not match the key that was used to encrypt this object. Terminate the session and provide the proper key.

dsmGroupHandler

The **dsmGroupHandler** function call performs an action on a logical file group depending on the input that is given. The client relates a number of individual objects together to reference and manage on the Tivoli Storage Manager server as a logical group.

Note: For more information, see "File grouping" on page 51.

Syntax

```
dsInt16_t dsmGroupHandler (dsmGroupHandlerIn_t   *dsmGroupHandlerInP,
                           dsmGroupHandlerOut_t  *dsmGroupHandlerOutP);
```

Parameters

dsmGroupHandlerIn_t *dsmGroupHandlerInP (I)
Passes group attributes to the API.

groupType
The type of the group. Values include:
- DSM_GROUPTYPE_PEER - peer group

actionType
The action to be executed. Values include:
- DSM_GROUP_ACTION_OPEN - creates a new group
- DSM_GROUP_ACTION_CLOSE - commits and saves an open group
- DSM_GROUP_ACTION_ADD - appends to a group
- DSM_GROUP_ACTION_ASSIGNTO - assigns to another group
- DSM_GROUP_ACTION_REMOVE- removes a member from a group

memberType.
The group type of the object. Values include:
- DSM_MEMBERTYPE_LEADER - group leader
- DSM_MEMBERTYPE_MEMBER - group member

***uniqueGroupTagP**
A unique string ID that is associated with a group.

leaderObjId
The Object ID for the group leader.

***objNameP**
A pointer to the object name of the group leader.

memberObjList
A list of objects to remove or assign.

dsmGroupHandlerOut_t *dsmGroupHandlerOutP (O)
Passes the address of the structure that the API completes. The structure version number is returned.

Return codes

The return code numbers are provided in parentheses ( ).

Table 42. Return codes for dsmGroupHandler

Return code	Explanation
DSM_RC_ABORT_INVALID_GROUP_ACTION (237)	An invalid operation was attempted on a group leader or member.

dsmInit

The **dsmInit** function call starts an API session and connects the client to Tivoli Storage Manager storage. The application client can have only one active session open at a time. To open another session with different parameters, use the **dsmTerminate** call first to end the current session.

To permit cross-node query and restore or retrieve, use the *-fromnode* and *-fromowner* string options. See “Accessing objects across nodes and owners” on page 27 for more information.

Syntax

```
dsInt16_t  dsmInit  (dsUint32_t        *dsmHandle,
   dsmApiVersion *dsmApiVersionP,
   char          *clientNodeNameP,
   char          *clientOwnerNameP,
   char          *clientPasswordP,
   char          *applicationType,
   char          *configfile,
   char          *options);
```

Parameters

dsUint32_t *dsmHandle (O)
The handle that identifies this initialization session and associates it with subsequent Tivoli Storage Manager calls.

dsmApiVersion *dsmApiVersionP (I)
A pointer to the data structure identifying the version of the API that the application client is using for this session. The structure contains the values of the three constants, DSM_API_VERSION, DSM_API_RELEASE, and DSM_API_LEVEL, that are set in the dsmapitd.h file. A previous call to **dsmQueryApiVersion** must be performed to ensure that compatibility exists between the application client API version and the version of the API library that is installed on the user’s workstation.

char *clientNodeNameP (I)
This parameter is a pointer to the node for the Tivoli Storage Manager session. All sessions must have a node name associated with them. The constant, DSM_MAX_NODE_LENGTH, in the dsmapitd.h file sets the maximum size that is permitted for a node name.

The node name is not case-sensitive.

If this parameter is set both to NULL and *passwordaccess* is set to *prompt*, the API attempts to obtain the node name first from the options string that was passed. If it is not there, the API then attempts to obtain the node name from the configuration file or options files. If these attempts to find the node name fail, the UNIX or Linux API uses the system host name, while APIs on other operating systems return the DSM_RC_REJECT_ID_UNKNOWN code.

This parameter must be NULL if the *passwordaccess* option in the dsm.sys file is set to *generate*. The API uses the system host name.

char *clientOwnerNameP (I)
This parameter is a pointer to the owner of the Tivoli Storage Manager session. If the operating system on which the session starts is a multi-user operating system, an owner name of NULL (the root user) has the authority to back up, archive, restore, or retrieve any objects belonging to the application, regardless of the owner of the object.

The owner name is case-sensitive.

This parameter must be NULL if the *passwordaccess* option in the dsm.sys file is set to *generate*. The API then uses the login user ID.

Note: On a multi-user operating system, if *passwordaccess* is set to *prompt*, it is not necessary for the owner name to match the active user ID of the session running the application.

char *clientPasswordP (I)
This parameter is a pointer to the password of the node on which the Tivoli Storage Manager session runs. The DSM_MAX_VERIFIER_LENGTH constant in the dsmapitd.h file sets the maximum size that is permitted for a password.

The password is not case-sensitive.

Except when the password file is first started, the value of this parameter is ignored if *passwordaccess* is set to *generate*.

char *applicationType (I)
This parameter identifies the application that is running the session. The application client defines the value.

Each time an API application client starts a session with the server, the application type (or platform) of the client is updated on the server. We recommend that the application type value contain an operating system abbreviation because this value is entered in the **platform** field on the server. The maximum string length is DSM_MAX_PLATFORM_LENGTH.

To see the current value of the application type, call **dsmQuerySessInfo**.

char *configfile (I)
This parameter points to a character string that contains the fully-qualified name of an API configuration file. Options specified in the API configuration file override their specification in the client options file. Options files are defined when Tivoli Storage Manager (client or API) is installed.

For the description and use of configuration files, see "Understanding configuration and options files" on page 2 or the *Tivoli Storage Manager Installing and Using the Backup-Archive Client* for your operating system.

char *options (I)
Points to a character string that can contain user options such as:

- *Compressalways*
- *Servername* (UNIX or Linux only)
- *TCPServeraddr*
- *Fromnode*
- *Fromowner*

- *EnableClientEncryptKey*

The application client can use the option list to override the values of these options that the configuration file sets.

The format of the options is:
1. Each option that is specified in the option list begins with a dash (-) and is followed by the option keyword.
2. The keyword, in turn, is followed by an equal sign (=) and then followed by the option parameter.
3. If the option parameter contains a blank space, enclose the parameter with single or double quotes.
4. If more than one option is specified, separate the options with blanks.

If options are NULL, values for all options are taken from the user options file or the API configuration file. For a description and use of each option, see the *Tivoli Storage Manager Installing and Using the Backup-Archive Client* for your operating system.

Return codes

The return code numbers are provided in parentheses ( ).

Table 43. Return codes for dsmInit

Return code	Explanation
DSM_RC_ABORT_SYSTEM_ERROR (1)	The server has detected a system error and has notified the clients.
DSM_RC_REJECT_VERIFIER_EXPIRED (52)	Password has expired and must be updated.
DSM_RC_REJECT_ID_UNKNOWN (53)	Could not find the node name.
DSM_RC_AUTH_FAILURE (137)	There was an authentication failure.
DSM_RC_NO_STARTING_DELIMITER (148)	There is no starting delimiter in pattern.
DSM_RC_NEEDED_DIR_DELIMITER (149)	A directory delimiter is needed immediately before and after the “match directories” meta-string (“...”) and one was not located.
DSM_RC_NO_PASS_FILE (168)	The password file is not available.
DSM_RC_UNMATCHED_QUOTE (177)	An unmatched quote is in the option string.
DSM_RC_NLS_CANT_OPEN_TXT (0610)	Unable to open the message text file.
DSM_RC_INVALID_OPT (400)	An entry in the option string is invalid.
DSM_RC_INVALID_DS_HANDLE (2014)	Invalid DSM handle.
DSM_RC_NO_OWNER_REQD (2032)	Owner parameter must be NULL when *passwordaccess* is set to *generate*.
DSM_RC_NO_NODE_REQD (2033)	Node parameter must be NULL when *passwordaccess* is set to *generate*.
DSM_RC_WRONG_VERSION (2064)	The API version for the application client has a higher value than the Tivoli Storage Manager version.
DSM_RC_PASSWD_TOOLONG (2103)	The password that was specified is too long.
DSM_RC_NO_OPT_FILE (2220)	A configuration file could not be located.
DSM_RC_INVALID_KEYWORD (2221)	A keyword that was specified in an options string is invalid.
DSM_RC_PATTERN_TOO_COMPLEX (2222)	The include-exclude pattern is too complex for Tivoli Storage Manager to interpret.
DSM_RC_NO_CLOSING_BRACKET (2223)	There is no closing bracket in the pattern.
DSM_RC_INVALID_SERVER (2225)	For a multi-user environment, the server in the system configuration file was not found.

Table 43. Return codes for dsmInit (continued)

Return code	Explanation
DSM_RC_NO_HOST_ADDR (2226)	Not enough information to connect to host.
DSM_RC_MACHINE_SAME (2227)	The nodename that is defined in the options file cannot be the same as the system host name.
DSM_RC_NO_API_CONFIGFILE (2228)	Cannot open the configuration file.
DSM_RC_NO_INCLEXCL_FILE (2229)	The include-exclude file was not found.
DSM_RC_NO_SYS_OR_INCLEXCL (2230)	Either the dsm.sys file or the include-exclude file was not found.

dsmInitEx

The dsmInitEx function call starts an API session using the additional parameters that permit extended verification.

Syntax

```
dsInt16_t  dsmInitEx   (dsUint32_t       *dsmHandleP,
                        dsmInitExIn_t    *dsmInitExInP,
                        dsmInitExOut_t   *dsmInitExOutP) ;
```

Parameters

dsUint32_t *dsmHandleP (O)
The handle that identifies this initialization session and associates it with subsequent Tivoli Storage Manager calls.

dsmInitExIn_t *dsmInitExInP
This structure contains the following input parameters:

dsmApiVersion *dsmApiVersionP (I)
This parameter is a pointer to the data structure that identifies the version of the API that the application client is using for this session. The structure contains the values of the four constants, DSM_API_VERSION, DSM_API_RELEASE, DSM_API_LEVEL, and DSM_API_SUBLEVEL that are set in the dsmapitd.h file. A previous call to **dsmQueryApiVersionEx** must be performed to ensure that compatibility exists between the API version of the application client and the version of the API library installed on the user's workstation.

char *clientNodeNameP (I)
This parameter is a pointer to the node for the Tivoli Storage Manager session. All sessions must have a node name associated with them. The constant, DSM_MAX_NODE_LENGTH in the dsmapitd.h file sets the maximum size that is permitted for a node name.

The node name is not case sensitive.

If this parameter is set to NULL, and *passwordaccess* is set to *prompt*, the API attempts to obtain the node name first from the options string that was passed. If it is not there, the API then attempts to obtain the node name from the configuration file or options files. If these attempts to find the node name fail, the UNIX or Linux API uses the system host name, while the APIs from other operating systems return the code, DSM_RC_REJECT_ID_UNKNOWN.

This parameter must be NULL if the *passwordaccess* option in the dsm.sys file is set to *generate*. The API then uses the system host name.

char *clientOwnerNameP (I)

This parameter is a pointer to the owner of the Tivoli Storage Manager session. If the operating system is a multi-user platform on which the session is started, an owner name of NULL (the root user) has the authority to back up, archive, restore, or retrieve any objects belonging to the application, regardless of the owner of the object.

The owner name is case sensitive.

This parameter must be NULL if the *passwordaccess* option in the dsm.sys file is set to *generate*. The API then uses the login user ID.

Note: On a multi-user platform, if *passwordaccess* is set to *prompt*, it is not necessary for the owner name to match the active user ID of the session running the application.

char *clientPasswordP (I)

A pointer to the password of the node on which the Tivoli Storage Manager session runs. The DSM_MAX_VERIFIER_LENGTH constant in the dsmapitd.h file sets the maximum size that is allowed for a password.

The password is not case sensitive.

Except when the password file is first started, the value of this parameter is ignored if *passwordaccess* is set to *generate*.

char *userNameP;

A pointer to the administrative user name that has client authority for this node.

char *userPasswordP;

A pointer to the password for the **userName**, if a value is supplied.

char *applicationType (I)

Identifies the application that is running the Tivoli Storage Manager session. The application client identifies the value.

Each time an API application client starts a session with the server, the application type (or operating system) of the client is updated on the server. We recommend that the application type value contain an operating system abbreviation because this value is entered in the **platform** field on the server. The maximum string length is DSM_MAX_PLATFORM_LENGTH.

To view the current value of the application type, call dsmQuerySessInfo.

char *configfile (I)

Points to a character string that contains the fully-qualified name of an API configuration file. Options specified in the API configuration file override their specification in the client options file. Options files are defined when Tivoli Storage Manager (client or API) is installed.

For a description and use of configuration files, see "Understanding configuration and options files" on page 2 and the *Tivoli Storage Manager Installing and Using the Backup-Archive Client* for your operating system.

char *options (I)

Points to a character string that can contain user options such as:

- *Compressalways*
- *Servername* (UNIX and Linux only)
- *TCPServeraddr* (non-UNIX)
- *Fromnode*

- *Fromowner*

The application client can use the options list to override the values of these options that the configuration file sets.

The format of the options is:

1. Each option that is specified in the option list begins with a dash (-) and is followed by the option keyword.
2. The keyword is followed by an equal sign (=) and then the option parameter.
3. If the option parameter contains a blank space, enclose the parameter with single or double quotes.
4. If more than one option is specified, separate the options with blanks.

If options are NULL, values for all options are taken from the user options file or the API configuration file. You can find descriptions and use of each option in the *Tivoli Storage Manager Installing and Using the Backup-Archive Client* for your operating system.

dirDelimiter

The directory delimiter that is prefixed on the file space, high-level or low-level names. You need to specify this only if the application overrides the system defaults. In a UNIX or Linux environment, this is /. In a Windows environment, this is\.

useUnicode

A Boolean flag that indicates if Unicode is enabled.

bCrossPlatform

A Boolean flag that indicates if cross-platform is enabled.

UseTsmBuffers

Indicates whether to use buffer copy elimination.

numTsmBuffers

Number of buffers when *useTsmBuffers = bTrue*.

bEncryptKeyEnabled

Indicates whether encryption with application-managed key is used.

encryptionPasswordP

The encryption password.

Note: When using *encryptkey=save*, if an encrypt key already exists, the value specified in the *encryptionPasswordP* is ignored.

dsmInitExOut_t *dsmInitExOut P

This structure contains the output parameters.

dsUint32_t *dsmHandle (0)

The handle that identifies this initialization session and associates it with subsequent API calls.

infoRC

Additional information about the return code. Check both the function return code and **infoRC**. If **infoRC** is DSM_RC_REJECT_LASTSESS_CANCELED (69), the Tivoli Storage Manager administrator cancelled the last session. The application should decide if it will cancel this session attempt by calling **dsmTerminate** immediately.

Return codes

The return code numbers are provided in parentheses ( ).

Table 44. Return codes for dsmInitEx

Return code	Explanation
DSM_RC_ABORT_SYSTEM_ERROR (1)	The Tivoli Storage Manager server has detected a system error and has notified the clients.
DSM_RC_REJECT_VERIFIER_EXPIRED (52)	Password has expired and must be updated. The next call must be **dsmChangePW** with the handle returned on this call.
DSM_RC_REJECT_ID_UNKNOWN (53)	Could not find the node name.
DSM_RC_TA_COMM_DOWN (103)	The communications link is down.
DSM_RC_AUTH_FAILURE (137)	There was an authentication failure.
DSM_RC_NO_STARTING_DELIMITER (148)	There is no starting delimiter in pattern.
DSM_RC_NEEDED_DIR_DELIMITER (149)	A directory delimiter is needed immediately before and after the “match directories” meta-string (“...”) and one was not found.
DSM_RC_NO_PASS_FILE (168)	The password file is not available.
DSM_RC_UNMATCHED_QUOTE (177)	An unmatched quote is in the option string.
DSM_RC_NLS_CANT_OPEN_TXT (0610)	Unable to open the message text file.
DSM_RC_INVALID_OPT (2013)	An entry in the option string is invalid.
DSM_RC_INVALID_DS_HANDLE (2014)	Invalid DSM handle.
DSM_RC_NO_OWNER_REQD (2032)	Owner parameter must be NULL when *passwordaccess* is set to *generate*.
DSM_RC_NO_NODE_REQD (2033)	Node parameter must be NULL when *passwordaccess* is set to *generate*.
DSM_RC_WRONG_VERSION (2064)	Application client’s API version has a higher value than the Tivoli Storage Manager version.
DSM_RC_PASSWD_TOOLONG (2103)	The specified password is too long.
DSM_RC_NO_OPT_FILE (2220)	No configuration file could be found.
DSM_RC_INVALID_KEYWORD (2221)	A keyword specified in an options string is invalid.
DSM_RC_PATTERN_TOO_COMPLEX (2222)	Include-exclude pattern too complex to be interpreted by Tivoli Storage Manager.
DSM_RC_NO_CLOSING_BRACKET (2223)	There is no closing bracket in the pattern.
DSM_RC_INVALID_SERVER (2225)	For a multi-user environment, the server in the system configuration file was not found.
DSM_RC_NO_HOST_ADDR (2226)	Not enough information to connect to the host.
DSM_RC_MACHINE_SAME (2227)	The nodename defined in the options file cannot be the same as the system host name.
DSM_RC_NO_API_CONFIGFILE (2228)	Cannot open the configuration file.
DSM_RC_NO_INCLEXCL_FILE (2229)	The include-exclude file was not found.
DSM_RC_NO_SYS_OR_INCLEXCL (2230)	Either the dsm.sys or the include-exclude file was not found.

dsmLogEvent

The **dsmLogEvent** function call logs a user message (ANE4991 I) to the server log file, to the local error log, or to both. A structure of type **logInfo** is passed in the call. This call must be performed while at **InSession** state inside a session. Do not perform it within a send, get, or query. To retrieve messages logged on the server, use the **query actlog** command through the administrative client.

Note:
- See the summary state diagram, Figure 20 on page 64.
- See the *Tivoli Storage Manager Administrator's Reference* for more information.

Syntax

```
dsInt16_t dsmLogEvent
     (dsUint32_t    dsmHandle,
     logInfo        *logInfoP);
```

Parameters

dsUint32_t dsmHandle(I)
The handle that associates this call with a previous **dsmInitEx** call.

logInfo *logInfoP (I)
Passes the message and destination. The application client is responsible for allocating storage for the structure.

The fields in the **logInfo** structure are:

message
The text of the message to be logged. This must be a null-ended string. The maximum length is DSM_MAX_RC_MSG_LENGTH.

dsmLogtype
Specifies where to log the message. Possible values include: **logServer**, **logLocal**, **logBoth**.

Return codes

The return code numbers are provided in parentheses ( ).

Table 45. Return codes for dsmLogEvent

Return code	Explanation
DSM_RC_STRING_TOO_LONG (2120)	The message string is too long.

dsmLogEventEx

The **dsmLogEventEx** function call logs a user message to the server log file, to the local error log, or to both. This call must be performed while at an **InSession** state inside a session. Do not perform it within a send, get, or query.

Note: See the summary state diagram, Figure 20 on page 64.

The severity determines the Tivoli Storage Manager message number. To view messages that are logged on the server, use the **query actlog** command through the administrative client. Use the Tivoli Storage Manager client option, *errorlogretention*, to prune the client error log file if the application generates numerous client messages written to the client log (*dsmLogType* either *logLocal* or *logBoth*). Refer to the *Tivoli Storage Manager Administrator's Reference* for more information.

Syntax

```
extern dsInt16_t DSMLINKAGE  dsmLogEventEx(
       dsUint32_t            dsmHandle,
       dsmLogExIn_t          *dsmLogExInP,
       dsmLogExOut_t         *dsmLogExOutP
);
```

Parameters

dsUint32_t dsmHandle(I)
The handle that associates this call with a previous dsmInitEx call.

dsmLogExIn_t *dsmLogExInP
This structure contains the input parameters.

dsmLogSeverity severity;
This parameter is the event severity. The possible values are:

```
logSevInfo,       /* information ANE4990 */
logSevWarning,    /* warning     ANE4991 */
logSevError,      /* Error       ANE4992 */
logSevSevere      /* severe      ANE4993 */
```

char appMsgID[8];
This parameter is a string to identify the specific application message. The format we recommend is three characters that are followed by four numbers. For example DSM0250.

dsmLogType logType;
This parameter specifies where to direct the event. The possible values include: logServer, logLocal, or logBoth.

char *message;
This parameter is the text of the event message to log. This must be a null-ended string. The maximum length is DSM_MAX_RC_MSG_LENGTH.

Note: Messages that go to the server should be in English. Non-English messages do not display correctly.

dsmLogExOut_t *dsmLogExOutP
This structure contains the output parameters.

Note: Currently, there are no output parameters.

Return codes

The return code numbers are provided in parentheses ( ).

Table 46. Return codes for dsmLogEventEx

Return code	Explanation
DSM_RC_STRING_TOO_LONG (2120)	The message string is too long.

dsmQueryAccess

The **dsmQueryAccess** function call queries the server for all access authorization rules for either backup versions or archived copies of your objects. A pointer to an array of access rules is passed in to the call, and the completed array is returned. A pointer to the number of rules is passed in to indicate how many rules are in the array.

There are no return codes that are specific to this call.

Syntax

```
dsInt16_t DSMLINKAGE dsmQueryAccess
                  (dsUint32_t          dsmHandle),
                   qryRespAccessData    **accessListP,
                   dsUint16_t          *numberOfRules) ;
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

qryRespAccessData **accessListP (O)
A pointer to an array of `qryRespAccessData` elements that the API library allocates. Each element corresponds to an access rule. The number of elements in the array is returned in the **numberOfRules** parameter. The information that is returned in each `qryRespAccessData` element includes the following:

Name	Description
ruleNumber	The ID for the access rule. This identifies the rule for deletion.
AccessType	The backup or archive type.
Node	The node on which you gave access.
Owner	The user to whom you gave access.
objName	The high-level, or low-level file space descriptors.

dsUint32_t *numberOfRules (O)
Returns the number of rules in the `accessList` array.

dsmQueryApiVersion

The **dsmQueryApiVersion** function call performs a query request for the API library version that the application client accesses.

All updates to the API are made in an upward-compatible format. Any application client with an API version or release less than, or equal to, the API library on the end user's workstation operates without change. Be aware before you proceed that should the **dsmQueryApiVersion** call return a version or version release older than that of the application clients, some API calls might be enhanced in a manner that is not supported by the end user's older version of the API.

The application API version number is stored in the dsmapitd.h header file as constants DSM_API_VERSION, DSM_API_RELEASE, and DSM_API_LEVEL.

There are no return codes that are specific to this call.

Syntax

```
void dsmQueryApiVersion  (dsmApiVersion *apiVersionP);
```

Parameters

dsmApiVersion *apiVersionP (O)
This parameter is a pointer to the structure that contains the API library version, release, and level components. For example, if the library is version 1.1.0, then, after returning from the call, the fields of the structure contain the following values:

```
dsmApiVersionP->version  = 1
dsmApiVersionP->release  = 1
dsmApiVersionP->level    = 0
```

dsmQueryApiVersionEx

The **dsmQueryApiVersionEx** function call performs a query request for the API library version that the application client accesses.

All updates to the API are made in an upward-compatible format. Any application client that has an API version or release less than or equal to the API library on the end user's workstation operates without change. See Summary of Code Changes in the README_api_enu file for exceptions to upward compatibility. If the **dsmQueryApiVersionEx** call returns a version or version release that is different from that of the application client, be aware before you proceed that some API calls might be enhanced in a manner that is not supported by the end user's older version of the API.

The application API version number is stored in the dsmapitd.h header file as constants DSM_API_VERSION, DSM_API_RELEASE, DSM_API_LEVEL, and DSM_API_SUBLEVEL.

There are no return codes that are specific to this call.

Syntax

```
void dsmQueryApiVersionEx (dsmApiVersionEx *apiVersionP);
```

Parameters

dsmApiVersionEx *apiVersionP (O)
This parameter is a pointer to the structure that contains the API library's version, release, level, and sublevel components. For example, if the library is version 5.5.0.0, then, after returning from the call, the fields of the structure contain the following values:

- `ApiVersionP->version  = 5`
- `ApiVersionP->release  = 5`
- `ApiVersionP->level    = 0`
- `ApiVersionP->subLevel = 0`

dsmQueryCliOptions

The **dsmQueryCliOptions** function call queries important option values in the user's option files. A structure of type **optStruct** is passed in the call and contains the information. This call is performed before **dsmInitEx** is called, and it determines the setup before the session.

Note: For more information about options, see the *Tivoli Storage Manager Installing and Using the Backup-Archive Client* for your operating system.

There are no return codes that are specific to this call.

Syntax

```
dsInt16_t dsmQueryCliOptions
    (optStruct     *optstructP);
```

Parameters

optStruct *optstructP (I/O)

This parameter passes the address of the structure that the API completes. The application client is responsible for allocating storage for the structure. On successful return, the appropriate information is entered in the fields in the structure.

The information returned in the **optStruct** structure is:

Name	Description
dsmiDir	The value of the environment DSMI_DIR variable.
dsmiConfig	The client option file as specified by the DSMI_CONFIG environment variable.
serverName	The name of the Tivoli Storage Manager server.
commMethod	The communication method selected. See the #defines for DSM_COMM_* in the dsmapitd.h file.
serverAddress	The address of the server that is based on the communication method.
nodeName	The name of the client's node (machine).
compression	This field provides information regarding the compression option.
passwordAccess	The values are: *bTrue* for generate, and *bFalse* for prompt.

dsmQuerySessInfo

The **dsmQuerySessInfo** function call starts a query request to Tivoli Storage Manager for information related to the operation of the specified session in **dsmHandle**. A structure of type **ApiSessInfo** is passed in the call, with all available session related information entered. This call is started after a successful **dsmInitEx** call.

The information that is returned in the **ApiSessInfo** structure includes the following:

- Server information: port number, date and time, and type
- Client defaults: application type, delete permissions, delimiters, and transaction limits
- Session information: login ID, and owner
- Policy data: domain, active policy set, and retention grace period

See Appendix C, "API type definitions source files," on page 143 for information about the content of the structure that is passed and each field within it.

Syntax

```
dsInt16_t dsmQuerySessInfo  (dsUint32_t        dsmHandle,
   ApiSessInfo   *SessInfoP);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

ApiSessInfo *SessInfoP (I/O)
This parameter passes the address of the structure that the API enters. The application client is responsible for allocating storage for the structure and for completing the field entries that indicate the version of the structure that is used. On successful return, the fields in the structure are completed with the appropriate information. The adsmServerName is the name that is given in the **define server** command on the Tivoli Storage Manager server. If the archiveRetentionProtection field is true, the server is enabled for retention protection.

Return codes

The return code numbers are provided in parentheses ( ).

Table 47. Return codes for dsmQuerySessInfo

Return code	Explanation
DSM_RC_NO_SESS_BLK (2006)	No server session block information.
DSM_RC_NO_POLICY_BLK (2007)	No server policy information available.
DSM_RC_WRONG_VERSION_PARM (2065)	Application client's API version is different from the Tivoli Storage Manager library version.

dsmQuerySessOptions

The dsmQuerySessOptions function call queries important option values that are valid in the specified session in dsmHandle. A structure of type `optStruct` is passed in the call and contains the information.

This call is started after a successful **dsmInitEx** call. The values that are returned might be different from the values returned on a dsmQueryCliOptions call, depending on values that are passed to the **dsmInitEx** call, primarily *optString*, and *optFile*. For information about option precedence, see "Understanding configuration and options files" on page 2.

There are no return codes that are specific to this call.

Syntax

```
dsInt16_t dsmQuerySessOptions
    (dsUint32_t    dsmHandle,
    optStruct      *optstructP);
```

Parameters

dsUint32_t dsmhandle(I)
The handle that associates this call with a previous dsmInitEx call.

optStruct *optstructP (I/O)

This parameter passes the address of the structure that the API completes. The application client is responsible for allocating storage for the structure. On successful return, the fields in the structure are completed with the appropriate information.

The information returned in the optStruct structure is:

Name	Description
dsmiDir	The value of the DSMI_DIR environment variable.
dsmiConfig	The dsm.opt file that the DSMI_CONFIG environment variable specifies.
serverName	The name of the Tivoli Storage Manager server stanza in the options file.
commMethod	The communication method that was selected. See the #defines for DSM_COMM_* in the dsmapitd.h file.
serverAddress	The address of the server that is based on the communication method.
nodeName	The name of the client's node (machine).
compression	The value of the compression option (*bTrue=on* and *bFalse=off*).
compressAlways	The value of the *compressalways* option (*bTrue=on* and *bFalse=off*).
passwordAccess	Value *bTrue* for generate, and *bFalse* for prompt.

For more information about options, see *Tivoli Storage Manager Installing and Using the Backup-Archive Client* for your operating system.

dsmRCMsg

The dsmRCMsg function call obtains the message text that is associated with an API return code.

The **msg** parameter displays the message prefix return code in parentheses ( ), followed by the message text. For example, a call to dsmRCMsg might return the following:

```
ANS0264E (RC2300) Only root user can execute dsmChangePW or dsmDeleteFS.
```

For some languages where characters are different in ANSII and OEM code pages, it might be necessary to convert strings from ANSII to OEM before printing them out (for example, Eastern European single-byte character sets). The following is an example:

```
   dsmRCMsg(dsmHangle, rc, msgBuf);
   #ifdef WIN32
   #ifndef WIN64
   CharToOemBuff(msgBuf, msgBuf, strlen(msgBuf));
   #endif
   #endif
   printf("
```

Syntax

```
dsInt16_t dsmRCMsg  (dsUint32_t       dsmHandle,
   dsInt16_t        dsmRC,
   char       *msg);
```

Parameters

dsUint32_t dsmHandle (I)

The handle that associates this call with a previous dsmInitEx call.

dsInt16_t dsmRC (I)
The API return code of the associated message text. The API return codes are listed in the dsmrc.h file. See Appendix B, “API return codes source file dsmrc.h,” on page 131 for more information.

char *msg (O)
This parameter is the message text that is associated with the return code, dsmRC. The caller is responsible for allocating enough space for the message text.

The maximum length for **msg** is defined as DSM_MAX_RC_MSG_LENGTH.

On platforms that have National Language Support and a choice of language message files, the API returns a message string in the national language.

Return codes

The return code numbers are provided in parentheses ( ).

Table 48. Return codes for dsmRCMsg

Return code	Explanation
DSM_RC_NULL_MSG (2002)	The **msg** parameter for dsmRCMsg call is a NULL pointer.
DSM_RC_INVALID_RETCODE (2021)	Return code that was passed to **dsmRCMsg** call is an invalid code.
DSM_RC_NLS_CANT_OPEN_TXT (0610)	Unable to open the message text file.

dsmRegisterFS

The **dsmRegisterFS** function call registers a new file space with the Tivoli Storage Manager server. Register a file space first before you can back up any data to it.

Application clients should not use the same file space names that a backup-archive client would use.
- On UNIX or Linux, run the **df** command for these names.
- On Windows, these names are generally the volume labels that are associated with the different drives on your system.
- On OS/400, there is no backup-archive client.

Syntax

```
dsInt16_t dsmRegisterFS  (dsUint32_t        dsmHandle,
   regFSData     *regFilespaceP);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

regFSData *regFilespaceP (I)
This parameter passes the name of the file space and associated information that you need to register with the Tivoli Storage Manager server.

Note: The *fstype* field includes the prefix, **“API:”.** All file space queries display this string. For example, if the user passes *myfstype* for *fstype* in **dsmRegisterFS**, the actual value string on the server is returned as `API:myfstype` when queried. This prefix distinguishes API objects from backup-archive objects.

The usable area for **fsInfo** is now DSM_MAX_USER_FSINFO_LENGTH.

Return codes

The return code numbers are provided in parentheses ( ).

Table 49. Return codes for dsmRegisterFS

Return code	Explanation
DSM_RC_INVALID_FSNAME (2016)	Invalid file space name.
DSM_RC_INVALID_DRIVE_CHAR (2026)	Drive letter is not an alphabetic character.
DSM_RC_NULL_FSNAME (2027)	Null file space name.
DSM_RC_FS_ALREADY_REGED (2062)	File space is already registered.
DSM_RC_WRONG_VERSION_PARM (2065)	Application client's API version is different from the Tivoli Storage Manager library version.
DSM_RC_FSINFO_TOOLONG (2106)	File space information is too long.

dsmReleaseBuffer

The **dsmReleaseBuffer** function returns a buffer to Tivoli Storage Manager. The application calls **dsmReleaseBuffer** after a **dsmGetDataEx** was called and the application has moved all the data out of the buffer and is ready to release it. **dsmReleaseBuffer** requires that **dsmInitEx** was called with the ***UseTsmBuffers*** set to *btrue* and a non-zero value was provided for *numTsmBuffers*. **dsmReleaseBuffer** should also be called if the application is about to call **dsmTerminate** and it still holds Tivoli Storage Manager buffers.

dsmReleaseBufferSyntax

```
dsInt16_t  dsmReleaseBuffer  (releaseBufferIn_t     *dsmReleaseBufferInP,
                              releaseBufferOut_t    *dsmReleaseBufferOutP) ;
```

Parameters

releaseBufferIn_t * dsmReleaseBufferInP (I)
This structure contains the following input parameters.

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsUint8_t tsmBufferHandle(I)
The handle that identifies this buffer.

char *dataPtr(I)
The address to which the application is written.

Return codes

The return code numbers are provided in parentheses ( ).

Table 50. Return codes for dsmReleaseBuffer

Return code	Explanation
DSM_RC_BAD_CALL_SEQUENCE	The call was not issued in the proper state.
DSM_RC_INVALID_TSMBUFFER	The handle or the value of **dataPtr** are invalid.
DSM_RC_BUFF_ARRAY_ERROR	A buffer array error occurred.

dsmRenameObj

The **dsmRenameObj** function call renames the high-level or low-level object name. For backup objects, pass in the current object name and changes either for high-level or low-level object names. For archive objects, pass in the current object file space name and object ID, and changes either for high-level or low-level object names. Use this function call within **dsmBeginTxn** and **dsmEndTxn** calls.

The merge flag determines whether or not a duplicate backup object name is merged with the existing backups. If the new name corresponds to an existing object and merge is true, the current object is converted to the new name and it becomes the active version of the new name while the existing active object that had that name becomes the top most inactive copy of the object. If the new name corresponds to an existing object and merge is false, the function then returns the return code, DSM_RC_ABORT_DUPLICATE_OBJECT.

Note: Only the owner of the object can rename it.

The **dsmRenameObj** function call tests for these merge conditions:
- The current **dsmObjName** object and the new high-level or low-level object must match on owner, copy group, and management class.
- The current **dsmObjName** must have been backed up more recently than the currently active object with the new name.
- There must be only an active copy of the current **dsmObjName** with no inactive copies.

Syntax

```
dsInt16_t dsmRenameObj  (dsmRenameIn_t     *dsmRenameInP,
                         dsmRenameOut_t    *dsmRenameOutP);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsmRenameIn_t *dsmRenameInP
This structure contains the input parameters.

dsUint8_t repository (I);
This parameter indicates whether the file space to delete is in the backup repository or the archive repository.

dsmObjName *objNameP (I);
This parameter is a pointer to the structure that contains the current file space name, high-level object name, low-level object name, and object type.

char newHl [DSM_MAX_HL_LENGTH + 1];
This parameter specifies the new high-level name.

char newLl [DSM_MAX_LL_LENGTH + 1];
This parameter specifies the new low-level name.

dsBool_t merge;
This parameter determines whether or not a backup object is merged with duplicate named objects. The values are either true or false.

ObjID;
The object ID for archive objects.

dsmRenameOut_t *dsmRnameOutP

This structure contains the output parameters.

Note: Currently, there are no output parameters.

Return codes

The return code numbers are provided in parentheses ( ).

Table 51. Return codes for dsmRenameObj

Return code	Explanation
DSM_RC_ABORT_MERGE_ERROR (45)	Server detected a merge error.
DSM_RC_ABORT_DUPLICATE_OBJECT (32)	Object already exists and merge is false.
DSM_RC_ABORT_NO_MATCH (2)	Object not found.
DSM_RC_REJECT_SERVER_DOWNLEVEL (58)	The Tivoli Storage Manager server must be at the 3.7.4.0 level or higher for this function to work.

dsmRequestBuffer

The **dsmRequestBuffer** function returns a buffer to Tivoli Storage Manager. The application calls **dsmRequestBuffer** after a **dsmGetDataEx** was called and the application has moved all the data out of the buffer and is ready to release it.

dsmReleaseBuffer requires that **dsmInitEx** was called with the *UseTsmBuffers* set to *btrue* and a non-zero value was provided for *numTsmBuffers*. **dsmReleaseBuffer** should also be called if the application is about to call **dsmTerminate** and it still holds Tivoli Storage Manager buffers.

Syntax

```
dsInt16_t  dsmRequestBuffer   (getBufferIn_t      *dsmRequestBufferInP,
                               getBufferOut_t     *dsmRequestBufferOutP) ;
```

Parameters

getBufferIn_t * dsmRequestBufferInP (I)

This structure contains the following input parameter:

dsUint32_t dsmHandle

The handle that identifies the session and associates it with a previous **dsmInitEx** call.

getBufferOut_t *dsmRequestBufferOut P (0)

This structure contains the output parameters.

dsUint8_t tsmBufferHandle(0)

The handle that identifies this buffer.

char *dataPtr(0)

The address to which application is written.

dsUint32_t *bufferLen(0)

Maximum number of bytes that can be written to this buffer.

Return codes

The return code numbers are provided in parentheses ( ).

Table 52. Return codes for dsmRequestBuffer

Return code	Explanation
DSM_RC_BAD_CALL_SEQUENCE (33)	The call was not issued in the proper state.
DSM_RC_SENDDATA_WITH_ZERO_SIZE (34)	If the object being sent is 0 length, no calls to **dsmReleaseBuffer** are allowed.
DSM_RC_BUFF_ARRAY_ERROR (121)	A valid buffer could not be obtained.

dsmRetentionEvent

The **dsmRetentionEvent** function call sends a list of object IDs to the server, with a retention event operation to be performed on these objects. Use this function call within **dsmBeginTxn** and **dsmEndTxn** calls.

Note: The Tivoli Storage Manager server must be at the Version 5.2.2.0 level or higher for this function to work.

The maximum number of objects in a call is limited to the value of *maxObjPerTxn* that is returned in the *ApisessInfo* structure from a **dsmQuerySessInfo** call.

Only an owner of an object can send an event on that object.

The following events are possible:

eventRetentionActivate
Can be issued only for objects that are bound to an event based management class. Sending this event activates the event for this object and the state of the retention for this object changes from DSM_ARCH_RETINIT_PENDING to DSM_ARCH_RETINIT_STARTED.

eventHoldObj
This event issues a retention or deletion hold on the object so that, until a release is issued, the object is not expired and cannot be deleted.

eventReleaseObj
This event can only be issued for an object that has a value of DSM_ARCH_HELD_TRUE in the *objectHeld* field and removes the hold on the object resuming the original retention policy.

Before you send *dsmRetentionEvent*, send the query sequence that is described in "Querying the Tivoli Storage Manager system" on page 35 to obtain the information for the object. The call to **dsmGetNextQObj** returns a data structure named **qryRespArchiveData** for archive queries. This data structure contains the information that is needed for **dsmRetentionEvent**.

Syntax

```
extern dsInt16_t DSMLINKAGE dsmRetentionEvent(
  dsmRetentionEventIn_t         *ddsmRetentionEventInP,
  dsmRetentionEventOut_t        *dsmRetentionEventOutP
   );
```

Parameters

dsmRetentionEventIn_t *dsmRetentionEventP
This structure contains the following input parameters:

dsUint16_t stVersion;
This parameter indicates the structure version.

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous dsmInitEx call.

dsmEventType_t evenType (I);
This parameter indicates the event type. See the beginning of this section for the meaning of these possible values: **eventRetentionActivate**, **eventHoldObj**, **eventReleaseObj**

dsmObjList_t objList;
This parameter indicates a list of object IDs to signal.

Return codes

The return code numbers are provided in parentheses ( ).

Table 53. Return codes for dsmRetentionEvent

Return code	Explanation
DSM_RC_ABORT_NODE_NOT_AUTHORIZED (36)	The node or user does not have proper authority.
DSM_RC_ABORT_TXN_LIMIT_EXCEEDED (249)	Too many objects in the transaction.
DSM_RC_ABORT_OBJECT_ALREADY_HELD (250)	Object is already held, cannot issue another hold.
DSM_RC_REJECT_SERVER_DOWNLEVEL (58)	The Tivoli Storage Manager server must be at the Version 5.2.2.0 level or higher for this function to work.

dsmSendBufferData

The **dsmSendBufferData** function call sends a byte stream of data to Tivoli Storage Manager through a buffer that was provided in a previous **dsmReleaseBuffer** call. The application client can pass any type of data for storage on the server. Usually this data are file data, but it is not limited to file data. You can call **dsmSendBufferData** several times, if the byte stream of data that you are sending is large. Regardless of whether the call succeeds or fails, the buffer is released.

Note: When using *useTsmBuffers*, even if an object is included for compression, the object is not compressed.

Syntax

```
dsInt16_t  dsmSendBufferData   (sendBufferDataIn_t        *dsmSendBufferDataExInP,
                                sendBufferDataOut_t       *dsmSendBufferDataOutP) ;
```

Parameters

sendBufferDataIn_t * dsmSendBufferDataInP (I)
This structure contains the following input parameters.

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsUint8_t tsmBufferHandle(I)
The handle that identifies the buffer to send.

char *dataPtr(I)
The address to which application data was written.

dsUint32_t numBytes(I)
The actual number of bytes written by the application (should always be less than the value provided in **dsmReleaseBuffer**).

Return codes

The return code numbers are provided in parentheses ( ).

Table 54. Return codes for dsmSendBufferData

Return code	Explanation
DSM_RC_BAD_CALL_SEQUENCE (2041)	The call was not issued in the proper state.
DSM_RC_INVALID_TSMBUFFER (2042)	The handle or the value of **dataPtr** are invalid.
DSM_RC_BUFF_ARRAY_ERROR (2045)	A buffer array error occurred.
DSM_RC_TOO_MANY_BYTES (2043)	The value of *numBytes* is bigger than the size of the buffer provided in the **dsmReleaseBuffer** call.

dsmSendData

The **dsmSendData** function call sends a byte stream of data to Tivoli Storage Manager through a buffer. The application client can pass any type of data for storage on the server. Usually, these data are file data, but are not limited to such. You can call **dsmSendData** several times, if the byte stream of data that you want to send is large.

Note: The application client cannot reuse the buffer that is specified in **dsmSendData** until the **dsmSendData** call returns.

Note: If Tivoli Storage Manager returns code 157 (DSM_RC_WILL_ABORT), start a call to **dsmEndSendObj** and then to **dsmEndTxn** with a vote of DSM_VOTE_COMMIT. The application should then receive return code 2302 (DSM_RC_CHECK_REASON_CODE) and pass the reason code back to the application user. This informs the user why the server is ending the transaction.

Syntax

```
dsInt16_t dsmSendData  (dsUint32_t   dsmHandle,
   DataBlk *dataBlkPtr);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

DataBlk *dataBlkPtr (I/O)
This parameter points to a structure that includes both a pointer to the buffer from which the data are to be sent, as well as the size of the buffer. On return, this structure contains the number of bytes that is actually transferred. See Appendix C, “API type definitions source files,” on page 143 for the type definition.

Return codes

The return code numbers are provided in parentheses ( ).

Table 55. Return codes for dsmSendData

Return code	Explanation
DSM_RC_NO_COMPRESS_MEMORY (154)	Insufficient memory available to perform data compression or expansion.

Table 55. Return codes for dsmSendData (continued)

Return code	Explanation
DSM_RC_COMPRESS_GREW (155)	During compression the compressed data grew in size compared to the original data.
DSM_RC_WILL_ABORT (157)	An unknown and unexpected error occurred, causing the transaction to halt.
DSM_RC_WRONG_VERSION_PARM (2065)	Application client's API version is different than the Tivoli Storage Manager library version.
DSM_RC_NEEDTO_ENDTXN (2070)	Need to end the transaction.
DSM_RC_OBJ_EXCLUDED (2080)	The include-exclude list excludes the object.
DSM_RC_OBJ_NOBCG (2081)	The object has no backup copy group and will not be sent to the server.
DSM_RC_OBJ_NOACG (2082)	The object has no archive copy group and is not sent to the server.
DSM_RC_SENDDATA_WITH_ZERO_SIZE (2107)	The object cannot send data with a zero byte *sizeEstimate*.

dsmSendObj

The **dsmSendObj** function call starts a request to send a single object to storage. Multiple **dsmSendObj** calls and associated **dsmSendData** calls can be made within the bounds of a transaction for performance reasons.

The **dsmSendObj** call processes the data for the object as a byte stream passed in memory buffers. The **dataBlkPtr** parameter in the **dsmSendObj** call permits the application client to either:

- Pass the data and the attributes (the attributes are passed through the **objAttrPtr**) of the object in a single call.
- Specify part of the object data through the **dsmSendObj** call and the remainder of the data through one or more **dsmSendData** calls.

Alternatively, the application client can specify only the attributes through the **dsmSendObj** call and specify the object data through one or more calls to **dsmSendData**. For this method, set **dataBlkPtr** to NULL on the **dsmSendObj** call.

Note: For certain object types, byte stream data might not be associated with the data; for example, a directory entry with no extended attributes.

Before **dsmSendObj** is called, a preceding **dsmBindMC** call must be made to properly bind a management class to the object that you want to back up or archive. The API keeps this binding so that it can associate the proper management class with the object when it is sent to the server. If you permit the management class that is bound on a **dsmSendObj** call to default for an object type of directory (DSM_OBJ_DIRECTORY), the default might not be the default management class. Instead, the management class with the greatest retention time is used. If more than one management class exists with this retention time, the first one that is encountered is used.

Follow all object data that is sent to storage with a **dsmEndSendObj** call. If you do not have object data to send to the server, or all data was contained within the **dsmSendObj** call, start a **dsmEndSendObj** call before you can start another **dsmSendObj** call. If multiple data sends were required through the **dsmSendData** call, the **dsmEndSendObj** follows the last send to indicate the state change.

Note: If Tivoli Storage Manager returns code 157 (DSM_RC_WILL_ABORT), start a call to **dsmEndTxn** with a vote of DSM_VOTE_COMMIT. The application should then receive return code 2302 (DSM_RC_CHECK_REASON_CODE) and pass the reason code back to the application user. This informs the user why the server is ending the transaction.

If the reason code is 11 (DSM_RS_ABORT_NO_REPOSIT_SPACE), it is possible that the *sizeEstimate* is too small for the actual amount of data. The application needs to determine a more accurate *sizeEstimate* and send the data again.

Syntax

```
dsInt16_t dsmSendObj  (dsUint32_t       dsmHandle,
   dsmSendType  sendType,
   void        *sendBuff,
   dsmObjName  *objNameP,
   ObjAttr     *objAttrPtr,
   DataBlk     *dataBlkPtr);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsmSendType sendType (I)
This parameter specifies the type of send that is being performed. Possible values include:

Name	Description
stBackup	A backup object that is sent to the server.
stArchive	An archive object that is sent to the server.
stBackupMountWait	A backup object for which you want the server to wait until the necessary device, such as a tape, is mounted.
stArchiveMountWait	An archive object for which you want the server to wait until the necessary device, such as a tape, is mounted.

Note: Use the **MountWait** types if there is any possibility that your application user might send data to a tape.

void *sendBuff (I)
This parameter is a pointer to a structure that contains other information specific to the **sendType** on the call. Currently, only a **sendType** of **stArchive** has an associated structure. This structure is called **sndArchiveData** and it contains the archive description.

dsmObjName *objNameP (I)
This parameter is a pointer to the structure that contains the file space name, high-level object name, low-level object name, and object type. See “Object names and IDs” on page 24 for more information.

ObjAttr *objAttrPtr (I)
This parameter passes object attributes of interest to the application. See Appendix C, “API type definitions source files,” on page 143 for the type definition.

The attributes are:

- **owner** refers to the owner of the object. Determining whether the owner is declared to be a specific name or an empty string is important when getting the object back from Tivoli Storage Manager storage. See “Accessing objects as session owner” on page 26 for more information.

- **sizeEstimate** is a best estimate of the total size of the data object to send to the server. Be as accurate as possible on this size, because the server uses this attribute for efficient space allocation and object placement within its storage resources.

  If the size estimate that you specified is significantly smaller than the actual number of bytes that are sent, the server might have difficulty allocating enough space and end the transaction with a reason code of 11 (DSM_RS_ABORT_NO_REPOSIT_SPACE).

  Note: The size estimate is for the total size of the data object in bytes.

  Objects with a size smaller than DSM_MIN_COMPRESS_SIZE do not compress.

  If your object has no bit data (only the attribute information from this call), the **sizeEstimate** should be zero.

  Note: Starting with version 5.1.0, the copy destination within a transaction is not checked for consistency on zero-length objects.
- **objCompressed** is a Boolean value that states whether or not the object data have already been compressed.

  If the object is compressed (object *compressed=bTrue*), Tivoli Storage Manager does not try to compress it again. If it is not compressed, Tivoli Storage Manager decides whether to compress the object, based on the values of the compression option set by the Tivoli Storage Manager administrator and set in the API configuration sources.

  If your application plans to use partial object restore or retrieve, you cannot compress the data while sending it. To enforce this, set *ObjAttr.objCompressed* to *bTrue*.
- **objInfo** saves information about the particular object.

  Note: Information is not stored here automatically. When this attribute is used, the attribute, *objInfoLength*, also must be set to show the length of *objInfo*.
- **mcNameP** contains the name of a management class that overrides the management class that is obtained from **dsmBindMC**.

DataBlk *dataBlkPtr (I/O)

This parameter points to a structure that includes both a pointer to the buffer of data that is to be backed up or archived and the size of that buffer. This parameter applies to **dsmSendObj** only. If you want to begin sending data on a subsequent **dsmSendData** call, rather than on the **dsmSendObj** call, set the buffer pointer in the DataBlk structure to NULL. On return, this structure contains the number of bytes that is actually transferred. See Appendix C, “API type definitions source files,” on page 143 for the type definition.

Return codes

The return code numbers are provided in parentheses ( ).

Table 56. Return codes for dsmSendObj

Return code	Explanation
DSM_RC_NO_COMPRESS_MEMORY (154)	Insufficient memory available to perform data compression or expansion.
DSM_RC_COMPRESS_GREW (155)	During compression, the compressed data grew in size compared to the original data.

Table 56. Return codes for dsmSendObj (continued)

Return code	Explanation
DSM_RC_WILL_ABORT (157)	An unknown and unexpected error occurred, causing the transaction to be halted.
DSM_RC_TL_NOACG (186)	The management class for this file does not have a valid copy group for the send type.
DSM_RC_NULL_OBJNAME (2000)	Null object name.
DSM_RC_NULL_OBJATTRPTR (2004)	Null object attribute pointer.
DSM_RC_INVALID_OBJTYPE (2010)	Invalid object type.
DSM_RC_INVALID_OBJOWNER (2019)	Invalid object owner.
DSM_RC_INVALID_SENDTYPE (2022)	Invalid send type.
DSM_RC_WILDCHAR_NOTALLOWED (2050)	Wildcard characters not allowed.
DSM_RC_FS_NOT_REGISTERED (2061)	File space not registered.
DSM_RC_WRONG_VERSION_PARM (2065)	Application client's API version is different from the Tivoli Storage Manager library version.
DSM_RC_NEEDTO_ENDTXN (2070)	Need to end transaction.
DSM_RC_OBJ_EXCLUDED (2080)	The include-exclude list excluded the object.
DSM_RC_OBJ_NOBCG (2081)	The object has no backup copy group, and it is not sent to the server.
DSM_RC_OBJ_NOACG (2082)	The object has no archive copy group, and it is not sent to the server.
DSM_RC_DESC_TOOLONG (2100)	Description is too long.
DSM_RC_OBJINFO_TOOLONG (2101)	Object information is too long.
DSM_RC_HL_TOOLONG (2102)	High-level qualifier is too long.
DSM_RC_FILESPACE_TOOLONG (2104)	File space name is too long.
DSM_RC_LL_TOOLONG (2105)	Low-level qualifier is too long.
DSM_RC_NEEDTO_CALL_BINDMC (2301)	**dsmBindMC** must be called first.

dsmSetAccess

The **dsmSetAccess** function call gives other users or nodes access to backup versions or archived copies of your objects, access to all your objects, or access to a selective set. When you give access to another user, that user can query, restore, or retrieve your files. This command supports wildcards for the following fields: *fs*, *hl*, *ll*, *node*, *owner*.

Note: You cannot give access to both backup versions and archive copies by using a single command. You must specify either backup or archive.

Syntax

```
dsInt16_t DSMLINKAGE dsmSetAccess
        (dsUint32_t         dsmHandle,
         dsmSetAccessType   accessType,
         dsmObjName         *objNameP,
         char               *node,
         char               *owner);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsmAccessType accessType (I)
This parameter specifies the type of objects for which you want to give access. Possible values include:

Name	Description
atBackup	Specifies that access is being set to backup objects.
atArchive	Specifies that the access is being set for archive objects.

dsmObjName *objNameP (I)
This parameter is a pointer to the structure that contains the file space name, the high-level object name, and the low-level object name.

Note: To specify all file spaces, use an asterisk (*) for the file space name.

char *node (I)
This parameter is a pointer to the node name for which access is given. For any node, specify an asterisk (*).

char *owner (I)
This parameter is a pointer to the user name on the node to which you gave access. For all users, specify an asterisk (*).

Return codes

The return code numbers are provided in parentheses ( ).

Table 57. Return codes for dsmSetAccess

Return code	Explanation
DSM_RC_INVALID_ACCESS_TYPE (2110)	Invalid access type specified.
DSM_RC_FILE_SPACE_NOT_FOUND (124)	Specified file space was not found on the server.
DSM_RC_QUERY_COMM_FAILURE (2111)	Communication error during server query.
DSM_RC_NO_FILES_BACKUP (2112)	No files were backed up for this file space.
DSM_RC_NO_FILES_ARCHIVE (2113)	No files were archived for this file space.
DSM_RC_INVALID_SETACCESS (2114)	Invalid formulation of set access.

dsmSetUp

The **dsmSetUp** function call overwrites environment variable values. Call **dsmSetUp** before **dsmInitEx**. The values that were passed in the **envSetUp** structure overwrite any existing environment variables or defaults. If you specify NULL for a field, values are taken from the environment. If you do not set a value, the values are taken from the defaults.

Note:

1. If you use **dsmSetUp**, always call **dsmTerminate** before **dsmCleanUp**.
2. API instrumentation is only be activated if the testflag INSTRUMENT: API is set in the configuration file and the **dsmSetUp** or **dsmCleanUp** calls are used in the application.

Syntax

```
dsInt16_t DSMLINKAGE dsmSetUp
        (dsBool_t    mtFlag,
         envSetUp    *envSetUpP);
```

Parameters

dsBool_t mtFlag (I)

This parameter specifies if the API will be used in a single thread, or a multithread mode. Values include:

```
DSM_SINGLETHREAD
DSM_MULTITHREAD
```

Note: The multithread flag must be on for LAN-free data transfer to occur.

envSetUp *envSetUpP(I)

This parameter is a pointer to the structure that holds the overwrite values. Specify NULL if you do not want to override existing environment variables. The fields in the **envSetUp** structure include:

Name	Description
dsmiDir	A fully-qualified directory path that contains a message file on UNIX or Linux. It also specifies the dsmtca and the dsm.sys directories.
dsmiConfig	The fully-qualified name of the client options file.
dsmiLog	The fully-qualified path of the error log directory.
argv	Pass the argv[0] name of the calling program if the application must run as TSM-Authorized. See "Using passwordaccess generate without TCA" on page 23 for more information.
logName	The file name for an error log if the application does not use dsierror.log.
inclExclCaseSensitive	Indicates whether include/exclude rules are case-sensitive or case-insensitive. This parameter can be used on Windows only, it is ignored elsewhere.

Return codes

The return code numbers are provided in parentheses ( ).

Table 58. Return codes for dsmSetUp

Return code	Explanation
DSM_RC_ACCESS_DENIED (106)	Access to the specified file or directory is denied.
DSM_RC_INVALID_OPT (0400)	An invalid option was found.
DSM_RC_NO_HOST_ADDR (0405)	The TCPSERVERADDRESS for this server is not defined in the server name stanza in the system options file.
DSM_RC_NO_OPT_FILE (0406)	The options file specified by filename cannot be found.
DSM_RC_MACHINE_SAME (0408)	The NODENAME defined in the options file cannot be the same as the system *HostName*.
DSM_RC_INVALID_SERVER (0409)	The system options file does not contain the SERVERNAME option.
DSM_RC_INVALID_KEYWORD (0410)	An invalid option keyword was found in the **dsmInitEx** configuration file, the option string, dsm.sys, or dsm.opt.
DSM_RC_PATTERN_TOO_COMPLEX (0411)	The include or exclude pattern issued is too complex to be accurately interpreted by Tivoli Storage Manager.

Table 58. Return codes for dsmSetUp (continued)

Return code	Explanation
DSM_RC_NO_CLOSING_BRACKET (0412)	The include or exclude pattern is incorrectly constructed. The closing bracket is missing.
DSM_RC_NLS_CANT_OPEN_TXT (0610)	The system is unable to open the message text file.
DSM_RC_NLS_INVALID_CNTL_REC (0612)	The system is unable to use the message text file.
DSM_RC_NOT_ADSM_AUTHORIZED (0927)	You must be the TSM-Authorized user to have multithreading and *passwordaccess* generate.
DSM_RC_NO_INCLEXCL_FILE (2229)	The include-exclude file was not found.
DSM_RC_NO_SYS_OR_INCLEXCL (2230)	Either the dsm.sys or the include-exclude file was not found.

dsmTerminate

The **dsmTerminate** function call ends a session with the Tivoli Storage Manager server and cleans up the Tivoli Storage Manager environment.

Syntax

There are no return codes that are specific for this call.

```
dsInt16_t dsmTerminate (dsUint32_t dsmHandle);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsmUpdateFS

The **dsmUpdateFS** function call updates a file space in Tivoli Storage Manager storage. This ensures that the Tivoli Storage Manager administrator has a current record of your file space.

Syntax

```
dsInt16_t dsmUpdateFS  (dsUint32_t      dsmHandle,
   char         *fs,
   dsmFSUpd     *fsUpdP,
   dsUint32_t    fsUpdAct);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

char *fs (I)
This parameter is a pointer to the file space name.

dsmFSUpd *fsUpdP (I)
This parameter is a pointer to the structure that has the proper fields completed for the update that you want. Complete only those fields that need updating.

dsUint32_t fsUpdAct (I)
A two-byte bit map that indicates which of the above fields to update. The bit masks are:

- DSM_FSUPD_FSTYPE

- DSM_FSUPD_FSINFO

  Note: For Windows operating systems, the drive letter value from **dsmDOSAttrib** is also updated when **FSINFO** is selected.
- DSM_FSUPD_OCCUPANCY
- DSM_FSUPD_CAPACITY
- DSM_FSUPD_BACKSTARTDATE
- DSM_FSUPD_BACKCOMPLETEDATE

See the DSM_FSUPD definitions in Appendix C, “API type definitions source files,” on page 143 for a description of these bit masks.

Return codes

The return code numbers are provided in parentheses ( ).

Table 59. Return codes for dsmUpdateFS

Return code	Explanation
DSM_RC_FS_NOT_REGISTERED (2061)	File space name is not registered.
DSM_RC_WRONG_VERSION_PARM (2065)	Application client’s API version is different from the Tivoli Storage Manager library version.
DSM_RC_FSINFO_TOOLONG (2106)	File space information is too long.

dsmUpdateObj

The **dsmUpdateObj** function call updates the meta information associated with an active backup or archive object already on the server. The application bit data is not affected. To update an object, you must give a specific non-wildcard name. To update an archived object, set the **dsmSendType** to **stArchive**. Only the latest named archive object is updated.

You can only start the **dsmUpdateObj** call in the session state; it cannot be called inside a transaction because it performs its own transaction. And, you can update only one object at a time.

Note: On a UNIX or Linux operating system, if you change the owner field, you cannot query or restore the object unless you are the root user.

Syntax

```
dsInt16_t  dsmUpdateObj
   (dsUint32_t      dsmHandle,
    dsmSendType     sendType,
    void           *sendBuff,
    dsmObjName     *objNameP,
    ObjAttr        *objAttrPtr,  /* objInfo */
    dsUint16_t      objUpdAct);  /* action bit vector */
```

Parameters

The field descriptions are the same as those in **dsmSendObj**, with the following exceptions:

dsmObjName *objNameP (I)
You cannot use a wildcard.

ObjAttr *objAttrPtr (I)
The **objCompressed** field is ignored for this call.

Other differences are:

- **owner**. If you specify a new **owner** field, the owner changes.
- **sizeEstimate**. If you specify a non-zero value it should be the actual amount of data sent, in bytes. The value is stored in the Tivoli Storage Manager meta data for future use.
- **objInfo**. This attribute contains the new information to be placed in the **objInfo** field. Set the **objInfoLength** to the length of the new **obiInfo**.

dsUint16_t objUpdAct
The bit masks and possible actions for **objUpdAct** are:

DSM_BACKUPD_MC
Updates the management class for the object.

DSM_BACKUPD_OBJINFO
Updates **objInfo**, **objInfoLength**, and **sizeEstimate**.

DSM_BACKUPD_OWNER
Updates the owner of the object.

DSM_ARCHUPD_DESCR
Updates the **Description** field. Enter the value for the new description through the **SendBuff** parameter. See the sample program for proper use.

DSM_ARCHUPD_OBJINFO
Updates **objInfo**, **objInfoLength**, and **sizeEstimate**.

DSM_ARCHUPD_OWNER
Updates the owner of the object.

Return codes

The return code numbers are provided in parentheses ( ).

Table 60. Return codes for dsmUpdateObj

Return code	Explanation
DSM_RC_INVALID_ACTION (2232)	Invalid action.
DSM_RC_FS_NOT_REGISTERED (2061)	File space not registered.
DSM_RC_BAD_CALL_SEQUENCE (2041)	Sequence of calls is invalid.
DSM_RC_WILDCHAR_NOTALLOWED (2050)	Wildcard characters are not allowed.
DSM_RC_ABORT_NO_MATCH (2)	Previous query does not match.

dsmUpdateObjEx

The **dsmUpdateObjEx** function call updates the meta information that is associated with an active backup or archive object that is on the server. The application bit data is not affected. To update an object, you must specify a non-wildcard name, or you can specify the object ID to update a specific archived object. You cannot use wildcard characters when specifying the name. To update a backup object, set the **dsmSendType** parameter to **stBackup**. To update an archived object, set the **dsmSendType** parameter to **stArchive**.

You can only start the **dsmUpdateObjEx** call in the session state; it cannot be called inside a transaction because it performs its own transaction. You can update only one object at a time.

Remember: On a UNIX or Linux operating system, if you change the owner field, you cannot query or restore the object unless you are the root user. Only the current active version of a backup object can be updated.

Syntax

```
dsInt16_t  dsmUpdateObjEx
   (dsmUpdateObjExIn_t   *dsmUpdateObjExInP,
    dsmUpdateObjExOut_t  *dsmUpdateObjExOutP);
```

Parameters

dsmUpdateObjExIn_t *dsmUpdateObjExInP

This structure contains the following input parameters:

dsUint16_t stVersion (I)

The current version of the structure that is used.

dsUint32_t dsmHandle (I)

The handle that associates this call with a previous **dsmInitEx** call.

dsmSendType sendType (I)

The type of send that is being performed. The value can be:

stBackup	A backup object that is sent to the server.
stArchive	An archive object that is sent to the server.

dsmObjName *objNameP (I)

A pointer to the structure that contains the filespace name, high-level object name, low-level object name, and object type. You cannot use a wildcard.

ObjAttr *objAttrPtr (I)

Passes object attributes to the application. The values that are updated depend on the flags in the **objUpdAct** field. The **objCompressed** attribute is ignored for this call.

The attributes are:

- **owner** changes the owner if a new name is entered.
- **sizeEstimate** is the actual amount of data that is sent in bytes. The value is stored in the Tivoli Storage Manager meta data for future use.
- **objCompressed** is a Boolean value that states whether or not the object data have already been compressed.
- **objInfo** is an attribute that contains the new information to be placed in the **objInfo** field. Set the **objInfoLength** to the length of the new **objInfo**.
- **mcNameP** contains the name of a management class that overrides the management class that is obtained from **dsmBindMC**.

dsUint32_t objUpdAct

Specifies the bit masks and actions for **objUpdAct** are:

DSM_BACKUPD_MC

Updates the management class for the object.

DSM_BACKUPD_OBJINFO
Updates the information object (**objInfo**), the length of the information object (**objInfoLength**), and the amount of data that is sent (**sizeEstimate**) for the backup object.

DSM_BACKUPD_OWNER
Updates the owner for the backup object.

DSM_ARCHUPD_DESCR
Updates the **Description** field for the archive object. Enter the value for the new description through the **sendBuff** parameter.

DSM_ARCHUPD_OBJINFO
Updates the information object (**objInfo**), the length of the information object (**objInfoLength**), and the amount of data that is sent (**sizeEstimate**) for the archive object.

DSM_ARCHUPD_OWNER
Updates the owner of the archive object.

ObjID archObjId
Specifies the unique object ID for a specific archive object. Because multiple archive objects can have the same name, this parameter identifies a specific one. You can obtain the object ID by using a query archive call.

dsmUpdateObjExOut_t *dsmUpdateObjExOutP
This structure contains the output parameter:

dsUint16_t stVersion (I)
The current version of the structure that is used.

Return codes

The return code numbers are provided in parentheses ( ) in the following table.

Table 61. Return codes for dsmUpdateObjEx

Return code	Explanation
DSM_RC_INVALID_ACTION (2012)	Invalid action.
DSM_RC_FS_NOT_REGISTERED (2061)	File space not registered.
DSM_RC_BAD_CALL_SEQUENCE (2041)	Sequence of calls is invalid.
DSM_RC_WILDCHAR_NOTALLOWED (2050)	Wildcard characters are not allowed.
DSM_RC_ABORT_NO_MATCH (2)	Previous query does not match.

Appendix A. API return codes with explanations

Many of the API return codes describe errors that cause processing to stop. You can send a message to the end user that describes the problem and suggest a course of action. To identify different messages, use these return code values or develop your own numbering system.

For each return code, the following information is provided:

- The return code number. This number corresponds to the number in the `dsmrc.h` header file. See Appendix B, "API return codes source file dsmrc.h," on page 131.
- The severity code. This letter indicates the severity that generated the return code. The possible severity codes and their values include:

Severity code	Type of error	Explanation
S	Severe error	Processing cannot continue.
E	Error	Processing cannot continue.
W	Warning	Processing can continue, but problems might develop later. Use caution.
I	Information	Processing continues. User response is not necessary.

- The symbolic name. This name corresponds to the definition in the dsmrc.h header file. Always use the symbolic name for a return code in your application rather than the return code number.
- The explanation. This field explains the circumstances under which this return code might be generated.
- The system action. This field describes the action that Tivoli Storage Manager takes in response to the return code.
- The user response. This field describes how you should respond to the system action.

The return codes are viewable through the information center at http://publib.boulder.ibm.com/infocenter/tsminfo/v6/topic/com.ibm.itsm.messages.doc/apirtncodes.html, or through the *Tivoli Storage Manager Messages* publication in the section ″Appendix A. API return codes with explanations.″

Appendix B. API return codes source file dsmrc.h

This index is a copy of the dsmrc.h header file that is used in the product, so you can see all possible return codes from the APIs.

See ″Appendix A. API return codes with explanations″ for more information.

```
/***********************************************************************
* Tivoli Storage Manager                                               *
* API Client Component                                                 *
*                                                                      *
* (C) Copyright IBM Corporation 1993,2008                              *
***********************************************************************/

/**********************************************************************/
/* Header File Name:  dsmrc.h                                         */
/*                                                                    */
/* Descriptive-name:  Return codes from Tivoli Storage Manager APIs   */
/**********************************************************************/
#ifndef _H_DSMRC
#define _H_DSMRC


#ifndef DSMAPILIB

#ifndef _H_ANSMACH
typedef int RetCode ;
#endif

#endif


#define DSM_RC_SUCCESSFUL                0 /* successful completion   */
#define DSM_RC_OK                        0 /* successful completion   */

#define DSM_RC_UNSUCCESSFUL             -1 /* unsuccessful completion */

/* dsmEndTxn reason code */
#define DSM_RS_ABORT_SYSTEM_ERROR           1
#define DSM_RS_ABORT_NO_MATCH               2
#define DSM_RS_ABORT_BY_CLIENT              3
#define DSM_RS_ABORT_ACTIVE_NOT_FOUND       4
#define DSM_RS_ABORT_NO_DATA                5
#define DSM_RS_ABORT_BAD_VERIFIER           6
#define DSM_RS_ABORT_NODE_IN_USE            7
#define DSM_RS_ABORT_EXPDATE_TOO_LOW        8
#define DSM_RS_ABORT_DATA_OFFLINE           9
#define DSM_RS_ABORT_EXCLUDED_BY_SIZE       10
#define DSM_RS_ABORT_NO_STO_SPACE_SKIP      11
#define DSM_RS_ABORT_NO_REPOSIT_SPACE       DSM_RS_ABORT_NO_STO_SPACE_SKIP
#define DSM_RS_ABORT_MOUNT_NOT_POSSIBLE     12
#define DSM_RS_ABORT_SIZESTIMATE_EXCEED     13
#define DSM_RS_ABORT_DATA_UNAVAILABLE       14
#define DSM_RS_ABORT_RETRY                  15
#define DSM_RS_ABORT_NO_LOG_SPACE           16
#define DSM_RS_ABORT_NO_DB_SPACE            17
#define DSM_RS_ABORT_NO_MEMORY              18

#define DSM_RS_ABORT_FS_NOT_DEFINED         20
#define DSM_RS_ABORT_NODE_ALREADY_DEFED     21
#define DSM_RS_ABORT_NO_DEFAULT_DOMAIN      22
#define DSM_RS_ABORT_INVALID_NODENAME       23
#define DSM_RS_ABORT_INVALID_POL_BIND       24
```

```
#define DSM_RS_ABORT_DEST_NOT_DEFINED        25
#define DSM_RS_ABORT_WAIT_FOR_SPACE          26
#define DSM_RS_ABORT_NOT_AUTHORIZED          27
#define DSM_RS_ABORT_RULE_ALREADY_DEFED      28
#define DSM_RS_ABORT_NO_STOR_SPACE_STOP      29

#define DSM_RS_ABORT_LICENSE_VIOLATION       30
#define DSM_RS_ABORT_EXTOBJID_ALREADY_EXISTS 31
#define DSM_RS_ABORT_DUPLICATE_OBJECT        32

#define DSM_RS_ABORT_INVALID_OFFSET          33    /* Partial Object Retrieve */
#define DSM_RS_ABORT_INVALID_LENGTH          34    /* Partial Object Retrieve */
#define DSM_RS_ABORT_STRING_ERROR            35
#define DSM_RS_ABORT_NODE_NOT_AUTHORIZED     36
#define DSM_RS_ABORT_RESTART_NOT_POSSIBLE    37
#define DSM_RS_ABORT_RESTORE_IN_PROGRESS     38
#define DSM_RS_ABORT_SYNTAX_ERROR            39

#define DSM_RS_ABORT_DATA_SKIPPED            40
#define DSM_RS_ABORT_EXCEED_MAX_MP           41
#define DSM_RS_ABORT_NO_OBJSET_MATCH         42
#define DSM_RS_ABORT_PVR_ERROR               43
#define DSM_RS_ABORT_BAD_RECOGTOKEN          44
#define DSM_RS_ABORT_MERGE_ERROR             45
#define DSM_RS_ABORT_FSRENAME_ERROR          46
#define DSM_RS_ABORT_INVALID_OPERATION       47
#define DSM_RS_ABORT_STGPOOL_UNDEFINED       48
#define DSM_RS_ABORT_INVALID_DATA_FORMAT     49
#define DSM_RS_ABORT_DATAMOVER_UNDEFINED     50

#define DSM_RS_ABORT_INVALID_MOVER_TYPE      231
#define DSM_RS_ABORT_ITEM_IN_USE             232
#define DSM_RS_ABORT_LOCK_CONFLICT           233
#define DSM_RS_ABORT_SRV_PLUGIN_COMM_ERROR   234
#define DSM_RS_ABORT_SRV_PLUGIN_OS_ERROR     235
#define DSM_RS_ABORT_CRC_FAILED              236
#define DSM_RS_ABORT_INVALID_GROUP_ACTION    237
#define DSM_RS_ABORT_DISK_UNDEFINED          238
#define DSM_RS_ABORT_BAD_DESTINATION         239
#define DSM_RS_ABORT_DATAMOVER_NOT_AVAILABLE 240
#define DSM_RS_ABORT_STGPOOL_COPY_CONT_NO    241
#define DSM_RS_ABORT_RETRY_SINGLE_TXN        242
#define DSM_RS_ABORT_TOC_CREATION_FAIL       243
#define DSM_RS_ABORT_TOC_LOAD_FAIL           244
#define DSM_RS_ABORT_PATH_RESTRICTED         245
#define DSM_RS_ABORT_NO_LANFREE_SCRATCH      246
#define DSM_RS_ABORT_INSERT_NOT_ALLOWED      247
#define DSM_RS_ABORT_DELETE_NOT_ALLOWED      248
#define DSM_RS_ABORT_TXN_LIMIT_EXCEEDED      249
#define DSM_RS_ABORT_OBJECT_ALREADY_HELD     250



/* RETURN CODE */

#define DSM_RC_ABORT_SYSTEM_ERROR            DSM_RS_ABORT_SYSTEM_ERROR
#define DSM_RC_ABORT_NO_MATCH                DSM_RS_ABORT_NO_MATCH
#define DSM_RC_ABORT_BY_CLIENT               DSM_RS_ABORT_BY_CLIENT
#define DSM_RC_ABORT_ACTIVE_NOT_FOUND        DSM_RS_ABORT_ACTIVE_NOT_FOUND
#define DSM_RC_ABORT_NO_DATA                 DSM_RS_ABORT_NO_DATA
#define DSM_RC_ABORT_BAD_VERIFIER            DSM_RS_ABORT_BAD_VERIFIER
#define DSM_RC_ABORT_NODE_IN_USE             DSM_RS_ABORT_NODE_IN_USE
#define DSM_RC_ABORT_EXPDATE_TOO_LOW         DSM_RS_ABORT_EXPDATE_TOO_LOW
#define DSM_RC_ABORT_DATA_OFFLINE            DSM_RS_ABORT_DATA_OFFLINE
#define DSM_RC_ABORT_EXCLUDED_BY_SIZE        DSM_RS_ABORT_EXCLUDED_BY_SIZE
```

```
#define DSM_RC_ABORT_NO_REPOSIT_SPACE        DSM_RS_ABORT_NO_STO_SPACE_SKIP
#define DSM_RC_ABORT_NO_STO_SPACE_SKIP       DSM_RS_ABORT_NO_STO_SPACE_SKIP

#define DSM_RC_ABORT_MOUNT_NOT_POSSIBLE      DSM_RS_ABORT_MOUNT_NOT_POSSIBLE
#define DSM_RC_ABORT_SIZESTIMATE_EXCEED      DSM_RS_ABORT_SIZESTIMATE_EXCEED
#define DSM_RC_ABORT_DATA_UNAVAILABLE        DSM_RS_ABORT_DATA_UNAVAILABLE
#define DSM_RC_ABORT_RETRY                   DSM_RS_ABORT_RETRY
#define DSM_RC_ABORT_NO_LOG_SPACE            DSM_RS_ABORT_NO_LOG_SPACE
#define DSM_RC_ABORT_NO_DB_SPACE             DSM_RS_ABORT_NO_DB_SPACE
#define DSM_RC_ABORT_NO_MEMORY               DSM_RS_ABORT_NO_MEMORY

#define DSM_RC_ABORT_FS_NOT_DEFINED          DSM_RS_ABORT_FS_NOT_DEFINED
#define DSM_RC_ABORT_NODE_ALREADY_DEFED      DSM_RS_ABORT_NODE_ALREADY_DEFED
#define DSM_RC_ABORT_NO_DEFAULT_DOMAIN       DSM_RS_ABORT_NO_DEFAULT_DOMAIN
#define DSM_RC_ABORT_INVALID_NODENAME        DSM_RS_ABORT_INVALID_NODENAME
#define DSM_RC_ABORT_INVALID_POL_BIND        DSM_RS_ABORT_INVALID_POL_BIND
#define DSM_RC_ABORT_DEST_NOT_DEFINED        DSM_RS_ABORT_DEST_NOT_DEFINED
#define DSM_RC_ABORT_WAIT_FOR_SPACE          DSM_RS_ABORT_WAIT_FOR_SPACE
#define DSM_RC_ABORT_NOT_AUTHORIZED          DSM_RS_ABORT_NOT_AUTHORIZED
#define DSM_RC_ABORT_RULE_ALREADY_DEFED      DSM_RS_ABORT_RULE_ALREADY_DEFED
#define DSM_RC_ABORT_NO_STOR_SPACE_STOP      DSM_RS_ABORT_NO_STOR_SPACE_STOP

#define DSM_RC_ABORT_LICENSE_VIOLATION       DSM_RS_ABORT_LICENSE_VIOLATION
#define DSM_RC_ABORT_EXTOBJID_ALREADY_EXISTS DSM_RS_ABORT_EXTOBJID_ALREADY_EXISTS
#define DSM_RC_ABORT_DUPLICATE_OBJECT        DSM_RS_ABORT_DUPLICATE_OBJECT

#define DSM_RC_ABORT_INVALID_OFFSET          DSM_RS_ABORT_INVALID_OFFSET
#define DSM_RC_ABORT_INVALID_LENGTH          DSM_RS_ABORT_INVALID_LENGTH

#define DSM_RC_ABORT_STRING_ERROR            DSM_RS_ABORT_STRING_ERROR
#define DSM_RC_ABORT_NODE_NOT_AUTHORIZED     DSM_RS_ABORT_NODE_NOT_AUTHORIZED
#define DSM_RC_ABORT_RESTART_NOT_POSSIBLE    DSM_RS_ABORT_RESTART_NOT_POSSIBLE
#define DSM_RC_ABORT_RESTORE_IN_PROGRESS     DSM_RS_ABORT_RESTORE_IN_PROGRESS
#define DSM_RC_ABORT_SYNTAX_ERROR            DSM_RS_ABORT_SYNTAX_ERROR

#define DSM_RC_ABORT_DATA_SKIPPED            DSM_RS_ABORT_DATA_SKIPPED
#define DSM_RC_ABORT_EXCEED_MAX_MP           DSM_RS_ABORT_EXCEED_MAX_MP
#define DSM_RC_ABORT_NO_OBJSET_MATCH         DSM_RS_ABORT_NO_OBJSET_MATCH
#define DSM_RC_ABORT_PVR_ERROR               DSM_RS_ABORT_PVR_ERROR
#define DSM_RC_ABORT_BAD_RECOGTOKEN          DSM_RS_ABORT_BAD_RECOGTOKEN
#define DSM_RC_ABORT_MERGE_ERROR             DSM_RS_ABORT_MERGE_ERROR
#define DSM_RC_ABORT_FSRENAME_ERROR          DSM_RS_ABORT_FSRENAME_ERROR
#define DSM_RC_ABORT_INVALID_OPERATION       DSM_RS_ABORT_INVALID_OPERATION
#define DSM_RC_ABORT_STGPOOL_UNDEFINED       DSM_RS_ABORT_STGPOOL_UNDEFINED
#define DSM_RC_ABORT_INVALID_DATA_FORMAT     DSM_RS_ABORT_INVALID_DATA_FORMAT
#define DSM_RC_ABORT_DATAMOVER_UNDEFINED     DSM_RS_ABORT_DATAMOVER_UNDEFINED

#define DSM_RC_ABORT_INVALID_MOVER_TYPE      DSM_RS_ABORT_INVALID_MOVER_TYPE
#define DSM_RC_ABORT_ITEM_IN_USE             DSM_RS_ABORT_ITEM_IN_USE
#define DSM_RC_ABORT_LOCK_CONFLICT           DSM_RS_ABORT_LOCK_CONFLICT
#define DSM_RC_ABORT_SRV_PLUGIN_COMM_ERROR   DSM_RS_ABORT_SRV_PLUGIN_COMM_ERROR
#define DSM_RC_ABORT_SRV_PLUGIN_OS_ERROR     DSM_RS_ABORT_SRV_PLUGIN_OS_ERROR
#define DSM_RC_ABORT_CRC_FAILED              DSM_RS_ABORT_CRC_FAILED
#define DSM_RC_ABORT_INVALID_GROUP_ACTION    DSM_RS_ABORT_INVALID_GROUP_ACTION
#define DSM_RC_ABORT_DISK_UNDEFINED          DSM_RS_ABORT_DISK_UNDEFINED
#define DSM_RC_ABORT_BAD_DESTINATION         DSM_RS_ABORT_BAD_DESTINATION
#define DSM_RC_ABORT_DATAMOVER_NOT_AVAILABLE DSM_RS_ABORT_DATAMOVER_NOT_AVAILABLE
#define DSM_RC_ABORT_STGPOOL_COPY_CONT_NO    DSM_RS_ABORT_STGPOOL_COPY_CONT_NO
#define DSM_RC_ABORT_RETRY_SINGLE_TXN        DSM_RS_ABORT_RETRY_SINGLE_TXN
#define DSM_RC_ABORT_TOC_CREATION_FAIL       DSM_RS_ABORT_TOC_CREATION_FAIL
#define DSM_RC_ABORT_TOC_LOAD_FAIL           DSM_RS_ABORT_TOC_LOAD_FAIL
#define DSM_RC_ABORT_PATH_RESTRICTED         DSM_RS_ABORT_PATH_RESTRICTED
#define DSM_RC_ABORT_NO_LANFREE_SCRATCH      DSM_RS_ABORT_NO_LANFREE_SCRATCH
#define DSM_RC_ABORT_INSERT_NOT_ALLOWED      DSM_RS_ABORT_INSERT_NOT_ALLOWED
#define DSM_RC_ABORT_DELETE_NOT_ALLOWED      DSM_RS_ABORT_DELETE_NOT_ALLOWED
#define DSM_RC_ABORT_TXN_LIMIT_EXCEEDED      DSM_RS_ABORT_TXN_LIMIT_EXCEEDED
#define DSM_RC_ABORT_OBJECT_ALREADY_HELD     DSM_RS_ABORT_OBJECT_ALREADY_HELD
```

```
/* Definitions for server signon reject codes                          */
/* These error codes are in the range (51 to 99) inclusive.            */
#define DSM_RC_REJECT_NO_RESOURCES           51
#define DSM_RC_REJECT_VERIFIER_EXPIRED       52
#define DSM_RC_REJECT_ID_UNKNOWN             53
#define DSM_RC_REJECT_DUPLICATE_ID           54
#define DSM_RC_REJECT_SERVER_DISABLED        55
#define DSM_RC_REJECT_CLOSED_REGISTER        56
#define DSM_RC_REJECT_CLIENT_DOWNLEVEL       57
#define DSM_RC_REJECT_SERVER_DOWNLEVEL       58
#define DSM_RC_REJECT_ID_IN_USE              59
#define DSM_RC_REJECT_ID_LOCKED              61
#define DSM_RC_SIGNONREJECT_LICENSE_MAX      62
#define DSM_RC_REJECT_NO_MEMORY              63
#define DSM_RC_REJECT_NO_DB_SPACE            64
#define DSM_RC_REJECT_NO_LOG_SPACE           65
#define DSM_RC_REJECT_INTERNAL_ERROR         66
#define DSM_RC_SIGNONREJECT_INVALID_CLI      67 /* client type not licensed */
#define DSM_RC_CLIENT_NOT_ARCHRETPROT        68
#define DSM_RC_REJECT_LASTSESS_CANCELED      69
#define DSM_RC_REJECT_UNICODE_NOT_ALLOWED    70
#define DSM_RC_REJECT_NOT_AUTHORIZED         71
#define DSM_RC_REJECT_TOKEN_TIMEOUT          72
#define DSM_RC_REJECT_INVALID_NODE_TYPE      73
#define DSM_RC_REJECT_INVALID_SESSIONINIT    74
#define DSM_RC_REJECT_WRONG_PORT             75
#define DSM_RC_CLIENT_NOT_SPMRETPROT         79
```

```
#define DSM_RC_USER_ABORT          101 /* processing aborted by user        */
#define DSM_RC_NO_MEMORY           102 /* no RAM left to complete request   */
#define DSM_RC_TA_COMM_DOWN       2021 /* no longer used                    */
#define DSM_RC_FILE_NOT_FOUND      104 /* specified file not found          */
#define DSM_RC_PATH_NOT_FOUND      105 /* specified path doesn't exist      */
#define DSM_RC_ACCESS_DENIED       106 /* denied due to improper permission */
#define DSM_RC_NO_HANDLES          107 /* no more file handles available    */
#define DSM_RC_FILE_EXISTS         108 /* file already exists               */
#define DSM_RC_INVALID_PARM        109 /* invalid parameter passed. CRITICAL*/
#define DSM_RC_INVALID_HANDLE      110 /* invalid file handle passed        */
#define DSM_RC_DISK_FULL           111 /* out of disk space                 */
#define DSM_RC_PROTOCOL_VIOLATION  113 /* call protocol violation. CRITICAL */
#define DSM_RC_UNKNOWN_ERROR       114 /* unknown system error. CRITICAL    */
#define DSM_RC_UNEXPECTED_ERROR    115 /* unexpected error. CRITICAL        */
#define DSM_RC_FILE_BEING_EXECUTED 116 /* No write is allowed               */
#define DSM_RC_DIR_NO_SPACE        117 /* directory can't be expanded       */
#define DSM_RC_LOOPED_SYM_LINK     118 /* too many symbolic links were
                                          encountered in translating path.  */
#define DSM_RC_FILE_NAME_TOO_LONG  119 /* file name too long                */
#define DSM_RC_FILE_SPACE_LOCKED   120 /* filespace is locked by the system */
#define DSM_RC_FINISHED            121 /* finished processing               */
#define DSM_RC_UNKNOWN_FORMAT      122 /* unknown format                    */
#define DSM_RC_NO_AUTHORIZATION    123 /* server response when the client has
                                          no authorization to read another
                                          host's owner backup/archive data  */
#define DSM_RC_FILE_SPACE_NOT_FOUND 124/* specified file space not found    */
#define DSM_RC_TXN_ABORTED         125 /* transaction aborted               */
#define DSM_RC_SUBDIR_AS_FILE      126 /* Subdirectory name exists as file  */
#define DSM_RC_PROCESS_NO_SPACE    127 /* process has no more disk space.   */
#define DSM_RC_PATH_TOO_LONG       128 /* a directory path being build became
                                          too long                          */
#define DSM_RC_NOT_COMPRESSED      129 /* file thought to be compressed is
                                          actually not                      */
#define DSM_RC_TOO_MANY_BITS       130 /* file was compressed using more bits
                                          then the expander can handle      */
```

```
#define DSM_RC_SYSTEM_ERROR        131 /* internal system error             */
#define DSM_RC_NO_SERVER_RESOURCES 132 /* server out of resources.          */
#define DSM_RC_FS_NOT_KNOWN        133 /* the file space is not known by the
                                          server                            */
#define DSM_RC_NO_LEADING_DIRSEP   134 /* no leading directory separator    */
#define DSM_RC_WILDCARD_DIR        135 /* wildcard character in directory
                                          path when not allowed             */
#define DSM_RC_COMM_PROTOCOL_ERROR 136 /* communications protocol error     */
#define DSM_RC_AUTH_FAILURE        137 /* authentication failure            */
#define DSM_RC_TA_NOT_VALID        138 /* TA not a root and/or SUID program */
#define DSM_RC_KILLED              139 /* process killed.                   */

#define DSM_RC_RETRY               143 /* retry same operation again        */

#define DSM_RC_WOULD_BLOCK         145 /* operation would cause the system to
                                          block waiting for input.          */
#define DSM_RC_TOO_SMALL           146 /* area for compiled pattern small   */
#define DSM_RC_UNCLOSED            147 /* no closing bracket in pattern     */
#define DSM_RC_NO_STARTING_DELIMITER 148 /* pattern has to start with
                                            directory delimiter             */
#define DSM_RC_NEEDED_DIR_DELIMITER 149 /* a directory delimiter is needed
                                          immediately before and after the
                                          "match directories" metastring
                                          ("...") and one wasn't found      */
#define DSM_RC_UNKNOWN_FILE_DATA_TYPE 150 /* structured file data type is
                                             unknown                        */
#define DSM_RC_BUFFER_OVERFLOW     151 /* data buffer overflow              */


#define DSM_RC_NO_COMPRESS_MEMORY  154 /* Compress/Expand out of memory     */
#define DSM_RC_COMPRESS_GREW       155 /* Compression grew                  */
#define DSM_RC_INV_COMM_METHOD     156 /* Invalid comm method specified     */
#define DSM_RC_WILL_ABORT          157 /* Transaction will be aborted       */
#define DSM_RC_FS_WRITE_LOCKED     158 /* File space is write locked        */
#define DSM_RC_SKIPPED_BY_USER     159 /* User wanted file skipped in the
                                          case of ABORT_DATA_OFFLINE        */
#define DSM_RC_TA_NOT_FOUND        160 /* TA not found in it's directory    */
#define DSM_RC_TA_ACCESS_DENIED    161 /* Access to TA is denied            */
#define DSM_RC_FS_NOT_READY        162 /* File space not ready              */
#define DSM_RC_FS_IS_BAD           163 /* File space is bad                 */
#define DSM_RC_FIO_ERROR           164 /* File input/output error           */
#define DSM_RC_WRITE_FAILURE       165 /* Error writing to file             */
#define DSM_RC_OVER_FILE_SIZE_LIMIT 166 /* File over system/user limit      */
#define DSM_RC_CANNOT_MAKE         167 /* Could not create file/directory,
                                          could be a bad name               */
#define DSM_RC_NO_PASS_FILE        168 /* password file needed and user is
                                          not root                          */
#define DSM_RC_VERFILE_OLD         169 /* password stored locally doesn't
                                          match the one at the host         */
#define DSM_RC_INPUT_ERROR         173 /* unable to read keyboard input     */
#define DSM_RC_REJECT_PLATFORM_MISMATCH 174 /* Platform name doesn't match
                                             up with what the server says
                                             is the platform for the client */
#define DSM_RC_TL_NOT_FILE_OWNER   175 /* User trying to backup a file is not
                                          the file's owner.                 */
#define DSM_RC_COMPRESSED_DATA_CORRUPTED 176 /* Compressed data is corrupted*/
#define DSM_RC_UNMATCHED_QUOTE     177 /* missing starting or ending quote  */

/*---------------------------------------------------------------------------*/
/* Return codes 180-199 are reserved for Policy Set handling                 */
/*---------------------------------------------------------------------------*/
#define DSM_RC_PS_MULTBCG          181 /* Multiple backup copy groups in 1 MC*/
#define DSM_RC_PS_MULTACG          182 /* Multiple arch.  copy groups in 1 MC*/
#define DSM_RC_PS_NODFLTMC         183 /* Default MC name not in policy set  */
#define DSM_RC_TL_NOBCG            184 /* Backup req, no backup copy group   */
#define DSM_RC_TL_EXCLUDED         185 /* Backup req, excl. by in/ex filter  */
#define DSM_RC_TL_NOACG            186 /* Archive req, no archive copy group */
```

```
#define DSM_RC_PS_INVALID_ARCHMC   187 /* Invalid MC name in archive override*/
#define DSM_RC_NO_PS_DATA          188 /* No policy set data on the server   */
#define DSM_RC_PS_INVALID_DIRMC    189 /* Invalid directory MC specified in
                                          the options file.                  */
#define DSM_RC_PS_NO_CG_IN_DIR_MC  190 /* No backup copy group in directory MC.
                                          Must specify an MC using DirMC
                                          option.                            */

#define DSM_RC_WIN32_UNSUPPORTED_FILE_TYPE 280 /* File is not of
                                                  Win32 type FILE_TYPE_DISK  */

/*---------------------------------------------------------------------------*/
/* Return codes for the Trusted Communication Agent                         */
/*---------------------------------------------------------------------------*/
#define DSM_RC_TCA_NOT_ROOT        161 /* Access to TA is denied             */
#define DSM_RC_TCA_ATTACH_SHR_MEM_ERR  200 /* Error attaching shared memory  */
#define DSM_RC_TCA_SHR_MEM_BLOCK_ERR   200 /* Shared memory block error      */
#define DSM_RC_TCA_SHR_MEM_IN_USE      200 /* Shared memory block error      */
#define DSM_RC_TCA_SHARED_MEMORY_ERROR 200 /* Shared memory block error      */
#define DSM_RC_TCA_SEGMENT_MISMATCH    200 /* Shared memory block error      */
#define DSM_RC_TCA_FORK_FAILED     292 /* Error forking off TCA process      */
#define DSM_RC_TCA_DIED            294 /* TCA died unexpectedly              */
#define DSM_RC_TCA_INVALID_REQUEST 295 /* Invalid request sent to TCA        */
#define DSM_RC_TCA_SEMGET_ERROR    297 /* Error getting semaphores           */
#define DSM_RC_TCA_SEM_OP_ERROR    298 /* Error in semaphore set or wait     */
#define DSM_RC_TCA_NOT_ALLOWED     299 /* TCA not allowed (multi thread)     */

/*---------------------------------------------------------------------------*/
/* 400-430  for options                                                      */
/*---------------------------------------------------------------------------*/
#define DSM_RC_INVALID_OPT           400 /* invalid option                   */
#define DSM_RC_NO_HOST_ADDR          405 /* Not enuf info to connect server  */
#define DSM_RC_NO_OPT_FILE           406 /* No default user configuration file*/
#define DSM_RC_MACHINE_SAME          408 /* -MACHINENAME same as real name   */
#define DSM_RC_INVALID_SERVER        409 /* Invalid server name from client  */
#define DSM_RC_INVALID_KEYWORD       410 /* Invalid option keyword           */
#define DSM_RC_PATTERN_TOO_COMPLEX   411 /* Can't match Include/Exclude entry*/
#define DSM_RC_NO_CLOSING_BRACKET    412 /* Missing closing bracket inc/excl */
#define DSM_RC_OPT_CLIENT_NOT_ACCEPTING 417/* Client doesn't accept this option
                                              from the server               */
#define DSM_RC_OPT_CLIENT_DOES_NOT_WANT 418/* Client doesn't want this value
                                              from the server               */
#define DSM_RC_OPT_NO_INCLEXCL_FILE  419   /* inclexcl file not found        */
#define DSM_RC_OPT_OPEN_FAILURE      420   /* can't open file                */
#define DSM_RC_OPT_INV_NODENAME      421/* used for Windows if nodename=local
                                          machine when CLUSTERNODE=YES       */
#define DSM_RC_OPT_NODENAME_INVALID  423/* generic invalid nodename          */
#define DSM_RC_OPT_ERRORLOG_CONFLICT 424/* both logmax & retention specified  */
#define DSM_RC_OPT_SCHEDLOG_CONFLICT 425/* both logmax & retention specified  */
#define DSM_RC_CANNOT_OPEN_TRACEFILE 426/* cannot open trace file             */
#define DSM_RC_CANNOT_OPEN_LOGFILE   427/* cannot open error log file         */


/*---------------------------------------------------------------------------*/
/* 600 to 610 for volume label codes                                         */
/*---------------------------------------------------------------------------*/
#define DSM_RC_DUP_LABEL           600 /* duplicate volume label found       */
#define DSM_RC_NO_LABEL            601 /* drive has no label                 */

/*---------------------------------------------------------------------------*/
/* Return codes for message file processing                                  */
/*---------------------------------------------------------------------------*/
#define DSM_RC_NLS_CANT_OPEN_TXT   610 /* error trying to open msg txt file  */
#define DSM_RC_NLS_CANT_READ_HDR   611 /* error trying to read header        */
#define DSM_RC_NLS_INVALID_CNTL_REC 612 /* invalid control record            */
#define DSM_RC_NLS_INVALID_DATE_FMT 613 /* invalid default date format       */
#define DSM_RC_NLS_INVALID_TIME_FMT 614 /* invalid default time format       */
```

```
#define DSM_RC_NLS_INVALID_NUM_FMT 615 /* invalid default number format      */

/*---------------------------------------------------------------------------*/
/* Return codes 620-630 are reserved for log message return codes            */
/*---------------------------------------------------------------------------*/
#define DSM_RC_LOG_CANT_BE_OPENED  620 /* error trying to open error log     */
#define DSM_RC_LOG_ERROR_WRITING_TO_LOG 621 /* error occurred writing to
                                               log file                      */
#define DSM_RC_LOG_NOT_SPECIFIED   622 /* no error log file was specified    */


/*---------------------------------------------------------------------------*/
/* Return codes 900-999 TSM CLIENT ONLY                                      */
/*---------------------------------------------------------------------------*/
#define DSM_RC_NOT_ADSM_AUTHORIZED  927 /* Must be ADSM authorized to perform*/
                                        /* action : root user or pwd auth    */
#define DSM_RC_REJECT_USERID_UNKNOWN 940 /* userid unknown on server         */
#define DSM_RC_FILE_IS_SYMLINK      959 /* errorlog or trace is a symbolic
                                           link
                                                                         */

#define DSM_RC_DIRECT_STORAGE_AGENT_UNSUPPORTED 961 /* Direct connection to SA not supported */
#define DSM_RC_FS_NAMESPACE_DOWNLEVEL 963 /* Long namespace has been removed from
                                             from the Netware volume */

#define DSM_RC_SERVER_SUPPORTS_FUNC    994 /* the server supports this function  */
#define DSM_RC_SERVER_AND_SA_SUPPORT_FUNC 995 /* Both server and SA support func */
#define DSM_RC_SERVER_DOWNLEVEL_FUNC   996 /* The server is downlevel for func   */
#define DSM_RC_STORAGEAGENT_DOWNLEVEL  997 /* the storage agent is downlevel     */
#define DSM_RC_SERVER_AND_SA_DOWNLEVEL 998 /* both server and SA downlevel       */


/* TCP/IP error codes */
#define DSM_RC_TCPIP_FAILURE       -50 /* TCP/IP communications failure      */
#define DSM_RC_CONN_TIMEDOUT       -51 /* TCP/IP connection attempt timedout */
#define DSM_RC_CONN_REFUSED        -52 /* TCP/IP connection refused by host  */
#define DSM_RC_BAD_HOST_NAME       -53 /* TCP/IP invalid host name specified */
#define DSM_RC_NETWORK_UNREACHABLE -54 /* TCP/IP host name unreachable       */
#define DSM_RC_WINSOCK_MISSING     -55 /* TCP/IP WINSOCK.DLL missing         */
#define DSM_RC_TCPIP_DLL_LOADFAILURE -56 /* Error from LoadLibrary           */
#define DSM_RC_TCPIP_LOADFAILURE   -57 /* Error from GetProcAddress   */
#define DSM_RC_TCPIP_USER_ABORT    -58 /* User aborted while in TCP/IP layer */

/*---------------------------------------------------------------------------*/
/* Return codes (-71)-(-90) are reserved for CommTSM error codes             */
/*---------------------------------------------------------------------------*/
#define DSM_RC_TSM_FAILURE         -71 /* TSM communications failure         */
#define DSM_RC_TSM_ABORT           -72 /* Session aborted abnormally         */

/*comm3270 error codes - no longer used*/
#define DSM_RC_COMM_TIMEOUT       2021   /* no longer used                    */
#define DSM_RC_EMULATOR_INACTIVE  2021   /* no longer used                    */
#define DSM_RC_BAD_HOST_ID        2021   /* no longer used                    */
#define DSM_RC_HOST_SESS_BUSY     2021   /* no longer used                    */
#define DSM_RC_3270_CONNECT_FAILURE 2021 /* no longer used                    */
#define DSM_RC_NO_ACS3ELKE_DLL    2021   /* no longer used                    */
#define DSM_RC_EMULATOR_ERROR     2021   /* no longer used                    */
#define DSM_RC_EMULATOR_BACKLEVEL 2021   /* no longer used                    */
#define DSM_RC_CKSUM_FAILURE      2021   /* no longer used                    */


/* The following Return codes are for EHLLAPI for Windows                    */
#define DSM_RC_3270COMMError_DLL        2021   /* no longer used             */
#define DSM_RC_3270COMMError_GetProc    2021   /* no longer used             */
#define DSM_RC_EHLLAPIError_DLL         2021   /* no longer used             */
#define DSM_RC_EHLLAPIError_GetProc     2021   /* no longer used             */
#define DSM_RC_EHLLAPIError_HostConnect 2021   /* no longer used             */
```

```
#define DSM_RC_EHLLAPIError_AllocBuff   2021   /* no longer used             */
#define DSM_RC_EHLLAPIError_SendKey     2021   /* no longer used             */
#define DSM_RC_EHLLAPIError_PacketChk   2021   /* no longer used             */
#define DSM_RC_EHLLAPIError_ChkSum      2021   /* no longer used             */
#define DSM_RC_EHLLAPIError_HostTimeOut 2021   /* no longer used             */
#define DSM_RC_EHLLAPIError_Send        2021   /* no longer used             */
#define DSM_RC_EHLLAPIError_Recv        2021   /* no longer used             */
#define DSM_RC_EHLLAPIError_General     2021   /* no longer used             */
#define DSM_RC_PC3270_MISSING_DLL       2021   /* no longer used             */
#define DSM_RC_3270COMM_MISSING_DLL     2021   /* no longer used             */


/* NETBIOS error codes */
#define DSM_RC_NETB_ERROR         -151 /* Could not add node to LAN          */
#define DSM_RC_NETB_NO_DLL        -152 /* The ACSNETB.DLL could not be loaded*/
#define DSM_RC_NETB_LAN_ERR       -155 /* LAN error detected                 */
#define DSM_RC_NETB_NAME_ERR      -158 /* Netbios error on Add Name          */
#define DSM_RC_NETB_TIMEOUT       -159 /* Netbios send timeout               */
#define DSM_RC_NETB_NOTINST       -160 /* Netbios not installed - DOS        */
#define DSM_RC_NETB_REBOOT        -161 /* Netbios config err - reboot DOS    */

/* Named Pipe error codes */
#define DSM_RC_NP_ERROR                        -190

/* CPIC error codes */
#define DSM_RC_CPIC_ALLOCATE_FAILURE           2021 /* no longer used      */
#define DSM_RC_CPIC_TYPE_MISMATCH              2021 /* no longer used      */
#define DSM_RC_CPIC_PIP_NOT_SPECIFY_ERR        2021 /* no longer used      */
#define DSM_RC_CPIC_SECURITY_NOT_VALID         2021 /* no longer used      */
#define DSM_RC_CPIC_SYNC_LVL_NO_SUPPORT        2021 /* no longer used      */
#define DSM_RC_CPIC_TPN_NOT_RECOGNIZED         2021 /* no longer used      */
#define DSM_RC_CPIC_TP_ERROR                   2021 /* no longer used      */
#define DSM_RC_CPIC_PARAMETER_ERROR            2021 /* no longer used      */
#define DSM_RC_CPIC_PROD_SPECIFIC_ERR          2021 /* no longer used      */
#define DSM_RC_CPIC_PROGRAM_ERROR              2021 /* no longer used      */
#define DSM_RC_CPIC_RESOURCE_ERROR             2021 /* no longer used      */
#define DSM_RC_CPIC_DEALLOCATE_ERROR           2021 /* no longer used      */
#define DSM_RC_CPIC_SVC_ERROR                  2021 /* no longer used      */
#define DSM_RC_CPIC_PROGRAM_STATE_CHECK        2021 /* no longer used      */
#define DSM_RC_CPIC_PROGRAM_PARAM_CHECK        2021 /* no longer used      */
#define DSM_RC_CPIC_UNSUCCESSFUL               2021 /* no longer used      */
#define DSM_RC_UNKNOWN_CPIC_PROBLEM            2021 /* no longer used      */
#define DSM_RC_CPIC_MISSING_LU                 2021 /* no longer used      */
#define DSM_RC_CPIC_MISSING_TP                 2021 /* no longer used      */
#define DSM_RC_CPIC_SNA6000_LOAD_FAIL          2021 /* no longer used      */
#define DSM_RC_CPIC_STARTUP_FAILURE            2021 /* no longer used      */

/*-----------------------------------------------------------------------------*/
/* Return codes -300 to -307 are reserved for IPX/SPX communications         */
/*-----------------------------------------------------------------------------*/
#define DSM_RC_TLI_ERROR                       2021 /* no longer used      */
#define DSM_RC_IPXSPX_FAILURE                  2021 /* no longer used      */
#define DSM_RC_TLI_DLL_MISSING                 2021 /* no longer used      */
#define DSM_RC_DLL_LOADFAILURE                 2021 /* no longer used      */
#define DSM_RC_DLL_FUNCTION_LOADFAILURE        2021 /* no longer used      */
#define DSM_RC_IPXCONN_REFUSED                 2021 /* no longer used      */
#define DSM_RC_IPXCONN_TIMEDOUT                2021 /* no longer used      */
#define DSM_RC_IPXADDR_UNREACHABLE             2021 /* no longer used      */
#define DSM_RC_CPIC_MISSING_DLL                2021 /* no longer used      */
#define DSM_RC_CPIC_DLL_LOADFAILURE            2021 /* no longer used      */
#define DSM_RC_CPIC_FUNC_LOADFAILURE           2021 /* no longer used      */

/*=== Shared Memory Protocol error codes   ===*/
#define DSM_RC_SHM_TCPIP_FAILURE               -450
#define DSM_RC_SHM_FAILURE                     -451
#define DSM_RC_SHM_NOTAUTH                     -452
```

```
#define DSM_RC_NULL_OBJNAME        2000 /* Object name pointer is NULL      */
#define DSM_RC_NULL_DATABLKPTR     2001 /* dataBlkPtr is NULL               */
#define DSM_RC_NULL_MSG            2002 /* msg parm in dsmRCMsg is NULL     */

#define DSM_RC_NULL_OBJATTRPTR     2004 /* Object Attr Pointer is NULL      */

#define DSM_RC_NO_SESS_BLK         2006 /* no server session info           */
#define DSM_RC_NO_POLICY_BLK       2007 /* no policy hdr     info           */
#define DSM_RC_ZERO_BUFLEN         2008 /* bufferLen is zero for dataBlkPtr */
#define DSM_RC_NULL_BUFPTR         2009 /* bufferPtr is NULL for dataBlkPtr */

#define DSM_RC_INVALID_OBJTYPE     2010 /* invalid object type              */
#define DSM_RC_INVALID_VOTE        2011 /* invalid vote                     */
#define DSM_RC_INVALID_ACTION      2012 /* invalid action                   */
#define DSM_RC_INVALID_DS_HANDLE   2014 /* invalid ADSM handle              */
#define DSM_RC_INVALID_REPOS       2015 /* invalid value for repository     */
#define DSM_RC_INVALID_FSNAME      2016 /* fs should start with dir delim   */
#define DSM_RC_INVALID_OBJNAME     2017 /* invalid full path name           */
#define DSM_RC_INVALID_LLNAME      2018 /* ll should start with dir delim   */
#define DSM_RC_INVALID_OBJOWNER    2019 /* invalid object owner name        */
#define DSM_RC_INVALID_ACTYPE      2020 /* invalid action type              */
#define DSM_RC_INVALID_RETCODE     2021 /* dsmRC in dsmRCMsg is invalid     */
#define DSM_RC_INVALID_SENDTYPE    2022 /* invalid send type                */
#define DSM_RC_INVALID_PARAMETER   2023 /* invalid parameter                */
#define DSM_RC_INVALID_OBJSTATE    2024 /* active, inactive, or any match?  */
#define DSM_RC_INVALID_MCNAME      2025 /* Mgmt class name not found        */
#define DSM_RC_INVALID_DRIVE_CHAR  2026 /* Drive letter is not alphabet     */
#define DSM_RC_NULL_FSNAME         2027 /* Filespace name is NULL           */
#define DSM_RC_INVALID_HLNAME      2028 /* hl should start with dir delim   */

#define DSM_RC_NUMOBJ_EXCEED       2029 /* BeginGetData num objs exceeded   */

#define DSM_RC_NEWPW_REQD          2030 /* new password is required         */
#define DSM_RC_OLDPW_REQD          2031 /* old password is required         */
#define DSM_RC_NO_OWNER_REQD       2032 /* owner not allowed. Allow default */
#define DSM_RC_NO_NODE_REQD        2033 /* node not allowed w/ pw=generate  */
#define DSM_RC_KEY_MISSING         2034 /* key file can't be found          */
#define DSM_RC_KEY_BAD             2035 /* content of key file is bad       */

#define DSM_RC_BAD_CALL_SEQUENCE   2041 /* Sequence of DSM calls not allowed*/
#define DSM_RC_INVALID_TSMBUFFER   2042 /* invalid value for tsmbuffhandle or dataPtr */
#define DSM_RC_TOO_MANY_BYTES      2043 /* too many bytes copied to buffer  */
#define DSM_RC_MUST_RELEASE_BUFFER 2044 /* cant exit app needs to release buffers */
#define DSM_RC_BUFF_ARRAY_ERROR    2045 /* internal buff array error        */
#define DSM_RC_INVALID_DATABLK     2046 /* using tsmbuff datablk should be null */
#define DSM_RC_ENCR_NOT_ALLOWED    2047 /* when using tsmbuffers encription not allowed */
#define DSM_RC_OBJ_COMPRESSED      2048 /* Can't restore using tsmBuff on compressed object */
#define DSM_RC_OBJ_ENCRYPTED       2049 /* Cant restore using tsmbuff an encr obj */
#define DSM_RC_WILDCHAR_NOTALLOWED 2050 /* Wild card not allowed for hl,ll  */
#define DSM_RC_POR_NOT_ALLOWED     2051 /* Can't use partial object restore with tsmBuffers */
#define DSM_RC_NO_ENCRYPTION_KEY   2052 /* Encryption key not found*/
#define DSM_RC_ENCR_CONFLICT       2053 /* mutually exclusive options */

#define DSM_RC_FSNAME_NOTFOUND     2060 /* Filespace name not found         */
#define DSM_RC_FS_NOT_REGISTERED   2061 /* Filespace name not registered    */
#define DSM_RC_FS_ALREADY_REGED    2062 /* Filespace already registered     */
#define DSM_RC_OBJID_NOTFOUND      2063 /* No object id to restore          */
#define DSM_RC_WRONG_VERSION       2064 /* Wrong level of code              */
#define DSM_RC_WRONG_VERSION_PARM  2065 /* Wrong level of parameter struct  */

#define DSM_RC_NEEDTO_ENDTXN       2070 /* Need to call dsmEndTxn           */

#define DSM_RC_OBJ_EXCLUDED        2080 /* Object is excluded by MC         */
#define DSM_RC_OBJ_NOBCG           2081 /* Object has no backup copy group  */
#define DSM_RC_OBJ_NOACG           2082 /* Object has no archive copy group */

#define DSM_RC_APISYSTEM_ERROR     2090 /* API internal error               */
```

```
#define DSM_RC_DESC_TOOLONG             2100 /* description is too long          */
#define DSM_RC_OBJINFO_TOOLONG          2101 /* object attr objinfo too long     */
#define DSM_RC_HL_TOOLONG               2102 /* High level qualifier is too long */
#define DSM_RC_PASSWD_TOOLONG           2103 /* password is too long             */
#define DSM_RC_FILESPACE_TOOLONG        2104 /* filespace name is too long       */
#define DSM_RC_LL_TOOLONG               2105 /* Low level qualifier is too long  */
#define DSM_RC_FSINFO_TOOLONG           2106 /* filespace length is too big      */
#define DSM_RC_SENDDATA_WITH_ZERO_SIZE  2107 /* send data w/ zero est            */

/*=== new return codes for dsmaccess ===*/
#define DSM_RC_INVALID_ACCESS_TYPE 2110 /* invalid access type              */
#define DSM_RC_QUERY_COMM_FAILURE  2111 /* communication error during query */
#define DSM_RC_NO_FILES_BACKUP     2112 /* No backed up files for this fs   */
#define DSM_RC_NO_FILES_ARCHIVE    2113 /* No archived files for this fs    */
#define DSM_RC_INVALID_SETACCESS   2114 /* invalid set access format        */

/*=== new return codes for dsmaccess ===*/
#define DSM_RC_STRING_TOO_LONG     2120 /* String parameter too long        */

#define DSM_RC_MORE_DATA           2200 /* There are more data to restore   */

#define DSM_RC_BUFF_TOO_SMALL      2210 /* DataBlk buffer too small for qry */

#define DSM_RC_NO_API_CONFIGFILE   2228 /*specified API confg file not found*/
#define DSM_RC_NO_INCLEXCL_FILE    2229 /* specified inclexcl file not found*/
#define DSM_RC_NO_SYS_OR_INCLEXCL  2230 /* either dsm.sys or inclexcl file
                                           specified in dsm.sys not found   */
#define DSM_RC_REJECT_NO_POR_SUPPORT 2231 /* server doesn't have POR support*/

#define DSM_RC_NEED_ROOT           2300 /* API caller must be root          */
#define DSM_RC_NEEDTO_CALL_BINDMC  2301 /* dsmBindMC must be called first   */
#define DSM_RC_CHECK_REASON_CODE   2302 /* check reason code from dsmEndTxn */

/*=== return codes 2400 - 2410 used by lic file see agentrc.h ===*/

/*=== return codes 2410 - 2430 used by Oracle agent see agentrc.h ===*/

#define DSM_RC_ENC_WRONG_KEY       4580 /* the key provided is incorrect    */
#define DSM_RC_ENC_NOT_AUTHORIZED  4582 /* user is not allowed to decrypt   */
#define DSM_RC_ENC_TYPE_UNKNOWN    4584 /* encryption type unknown          */

/*=============================================================================
   Return codes (4600)-(4624) are reserved for clustering
==============================================================================*/
#define DSM_RC_CLUSTER_INFO_LIBRARY_NOT_LOADED        4600
#define DSM_RC_CLUSTER_LIBRARY_INVALID                4601
#define DSM_RC_CLUSTER_LIBRARY_NOT_LOADED             4602
#define DSM_RC_CLUSTER_NOT_MEMBER_OF_CLUSTER          4603
#define DSM_RC_CLUSTER_NOT_ENABLED                    4604
#define DSM_RC_CLUSTER_NOT_SUPPORTED                  4605
#define DSM_RC_CLUSTER_UNKNOWN_ERROR                  4606


/*=============================================================================
   Return codes (5701)-(5749) are reserved for proxy
==============================================================================*/
#define DSM_RC_PROXY_REJECT_NO_RESOURCES              5702
#define DSM_RC_PROXY_REJECT_DUPLICATE_ID              5705
#define DSM_RC_PROXY_REJECT_ID_IN_USE                 5710
#define DSM_RC_PROXY_REJECT_INTERNAL_ERROR            5717
#define DSM_RC_PROXY_REJECT_NOT_AUTHORIZED            5722
#define DSM_RC_PROXY_INVALID_FROMNODE                 5746
#define DSM_RC_PROXY_INVALID_SERVERFREE               5747
#define DSM_RC_PROXY_INVALID_CLUSTER                  5748
#define DSM_RC_PROXY_INVALID_FUNCTION                 5749
```

```
/*=============================================================================
   Return codes 5801 - 5849 are reserved for cryptography/security
=============================================================================*/

#define DSM_RC_CRYPTO_ICC_ERROR                       5801
#define DSM_RC_CRYPTO_ICC_CANNOT_LOAD                 5802
#define DSM_RC_SSL_NOT_SUPPORTED                      5803
#define DSM_RC_SSL_INIT_FAILED                        5804
#define DSM_RC_SSL_KEYFILE_OPEN_FAILED                5805
#define DSM_RC_SSL_KEYFILE_BAD_PASSWORD               5806
#define DSM_RC_SSL_BAD_CERTIFICATE                    5807

#endif /* _H_DSMRC */
```

Appendix C. API type definitions source files

This appendix contains structure definitions, type definitions, and constants for the API. The first header files, dsmapitd.h and tsmapitd.h, illustrate the definitions that are common to all operating systems.

The second header file, dsmapips.h, provides an example of definitions that are specific to a particular operating system; in this example, the Windows platform.

The third header file, release.h, includes the version and release information.

```
/***********************************************************************
* Tivoli Storage Manager                                               *
* API Client Component                                                 *
*                                                                      *
* (C) Copyright IBM Corporation 1993,2008                              *
***********************************************************************/
/***********************************************************************
* Header File Name: dsmapitd.h
*
* Environment:     ************************************************
*                  ** This is a platform-independent source file **
*
*                  ************************************************
*
* Design Notes:    This file contains basic data types and constants
*                  includable by all client source files. The constants
*                  within this file should be set properly for the
*                  particular machine and operating system on which the
*                  client software is to be run.
*
*                  Platform specific definitions are included in dsmapips.h
*
* Descriptive-name: Definitions for Tivoli Storage manager API constants
*---------------------------------------------------------------------*/
#ifndef _H_DSMAPITD
#define _H_DSMAPITD
#include "dsmapips.h"      /* Platform specific definitions*/
#include "release.h"
/*=== set the structure alignment to pack the structures ===*/
#if (_OPSYS_TYPE == DS_WINNT) && !defined(_WIN64)
#pragma pack(1)
#endif
#ifdef _MAC
/*=============================================================================
  choices are:
http://developer.apple.com/documentation/DeveloperTools/Conceptual/PowerPCRuntime/
Data/chapter_2_section_3.html
#pragma option align=<mode>
where <mode> is power, mac68k, natural, or packed.
=============================================================================*/
#pragma options align=packed
#endif
typedef char osChar_t;
/*<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>*/
/*                   D E F I N E S                                        */
/*<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>*/
/*-------------------------------------------------------------------------+
|  API Version, Release, and Level to use in dsmApiVersion on dsmInit()    |
+-------------------------------------------------------------------------*/
#define DSM_API_VERSION     COMMON_VERSION
#define DSM_API_RELEASE     COMMON_RELEASE
```

```
#define DSM_API_LEVEL       COMMON_LEVEL
#define DSM_API_SUBLEVEL    COMMON_SUBLEVEL
/*-------------------------------------------------------------------------+
| Maximum field lengths                                                    |
+-------------------------------------------------------------------------*/
#define DSM_MAX_CG_DEST_LENGTH        30        /* copy group destination */
#define DSM_MAX_CG_NAME_LENGTH        30        /* copy group name        */
#define DSM_MAX_DESCR_LENGTH          255       /* archive description    */
#define DSM_MAX_DOMAIN_LENGTH         30        /* policy domain name     */
#define DSM_MAX_FSINFO_LENGTH         500       /* filespace info         */
#define DSM_MAX_USER_FSINFO_LENGTH    480       /* max user filespace info*/
#define DSM_MAX_FSNAME_LENGTH         1024      /* filespace name         */
#define DSM_MAX_FSTYPE_LENGTH         32        /* filespace type         */
#define DSM_MAX_HL_LENGTH             1024      /* object high level name */
#define DSM_MAX_ID_LENGTH             64        /* session node name      */
#define DSM_MAX_LL_LENGTH             256       /* object low level name  */
#define DSM_MAX_MC_NAME_LENGTH        30        /* management class name  */
#define DSM_MAX_OBJINFO_LENGTH        255       /* object info            */
#define DSM_MAX_OWNER_LENGTH          64        /* object owner name      */
#define DSM_MAX_PLATFORM_LENGTH       16        /* application type       */
#define DSM_MAX_PS_NAME_LENGTH        30        /* policy set name        */
#define DSM_MAX_SERVERTYPE_LENGTH     32        /* server platform type   */
#define DSM_MAX_VERIFIER_LENGTH       64        /* password               */
#define DSM_PATH_MAX                  1024      /* API config file path   */
#define DSM_NAME_MAX                  255       /* API config file name   */
#define DSM_MAX_NODE_LENGTH           64        /* node/machine name      */
#define DSM_MAX_RC_MSG_LENGTH         1024      /* msg parm for dsmRCMsg  */
#define DSM_MAX_SERVER_ADDRESS        1024     /* server address */
#define DSM_MAX_MC_DESCR_LENGTH       DSM_MAX_DESCR_LENGTH /* mgmt class  */
#define DSM_MAX_SERVERNAME_LENGTH     DSM_MAX_ID_LENGTH /* server name    */
#define DSM_MAX_GET_OBJ               4080     /* max objs on BeginGetData */
#define DSM_MAX_PARTIAL_GET_OBJ       1300 /* max partial objs on BeginGetData*/
/*-------------------------------------------------------------------------+
| Minimum field lengths                                                    |
+-------------------------------------------------------------------------*/
#define DSM_MIN_COMPRESS_SIZE  2048 /* minimum number of bytes an object  */
                                    /* needs before compression is allowed*/
/*-------------------------------------------------------------------------+
|  Values for mtFlag in dsmSetup call                                      |
+-------------------------------------------------------------------------*/
#define DSM_MULTITHREAD      bTrue
#define DSM_SINGLETHREAD     bFalse
/*-------------------------------------------------------------------------+
|  Values for object type in dsmObjName structure                          |
|  Note: These values must be kept in sync with dsmcomm.h                 |
+-------------------------------------------------------------------------*/
#define DSM_OBJ_FILE                  0x01 /*object has attrib info & data*/
#define DSM_OBJ_DIRECTORY             0x02 /*obj has only attribute info  */
#define DSM_OBJ_RESERVED1             0x04 /* for future use              */
#define DSM_OBJ_RESERVED2             0x05 /* for future use              */
#define DSM_OBJ_RESERVED3             0x06 /* for future use              */
#define DSM_OBJ_WILDCARD              0xFE /* Any object type             */
#define DSM_OBJ_ANY_TYPE              0xFF /* for future use              */
/*-------------------------------------------------------------------------+
| Type definition for compressedState in QryResp                           |
+-------------------------------------------------------------------------*/
#define DSM_OBJ_COMPRESSED_UNKNOWN    0
#define DSM_OBJ_COMPRESSED_YES        1
#define DSM_OBJ_COMPRESSED_NO         2
/*---------------------------------------------------------------------+
| Definitions for "group type" field in tsmGrouphandlerIn_t           |
+---------------------------------------------------------------------*/
#define  DSM_GROUPTYPE_NONE         0x00   /* Not a group member      */
#define  DSM_GROUPTYPE_RESERVED1    0x01   /* for future use          */
#define  DSM_GROUPTYPE_PEER         0x02   /* Peer group              */
#define  DSM_GROUPTYPE_RESERVED2    0x03   /* for future use          */
/*---------------------------------------------------------------------+
| Definitions for "member type" field in tsmGrouphandlerIn_t           |
+---------------------------------------------------------------------*/
#define  DSM_MEMBERTYPE_LEADER      0x01   /* group leader  */
#define  DSM_MEMBERTYPE_MEMBER      0x02   /* group member  */
/*---------------------------------------------------------------------+
| Definitions for "operation type" field in tsmGrouphandlerIn_t         |
+---------------------------------------------------------------------*/
#define DSM_GROUP_ACTION_BEGIN         0x01
#define DSM_GROUP_ACTION_OPEN          0x02 /* create new group             */
#define DSM_GROUP_ACTION_CLOSE         0x03 /* commit and save an open group */
#define DSM_GROUP_ACTION_ADD           0x04 /* Append to a group */
#define DSM_GROUP_ACTION_ASSIGNTO      0x05 /* Assign to a another group */
#define DSM_GROUP_ACTION_REMOVE        0x06 /* remove a member from a group */
/*-------------------------------------------------------------------------+
| Values for copySer in DetailCG structures for Query Mgmt Class response  |
+-------------------------------------------------------------------------*/
#define Copy_Serial_Static          1   /*Copy Serialization Static       */
#define Copy_Serial_Shared_Static   2   /*Copy Serialization Shared Static*/
#define Copy_Serial_Shared_Dynamic  3   /*Copy Serialization Shared Dynamic*/
#define Copy_Serial_Dynamic         4   /*Copy Serialization Dynamic      */
*-------------------------------------------------------------------------+
| Values for copyMode in DetailCG structures for Query Mgmt Class response |
+-------------------------------------------------------------------------*/
#define Copy_Mode_Modified          1   /*Copy Mode Modified              */
#define Copy_Mode_Absolute          2   /*Copy Mode Absolute              */
/*-------------------------------------------------------------------------+
|  Values for objState in qryBackupData structure                          |
+-------------------------------------------------------------------------*/
#define DSM_ACTIVE           0x01      /* query only active objects      */
#define DSM_INACTIVE         0x02      /* query only inactive objects    */
#define DSM_ANY_MATCH        0xFF      /* query all backup objects       */
/*-------------------------------------------------------------------------+
| Boundary values for dsmDate.year field in qryArchiveData structure      |
+-------------------------------------------------------------------------*/
#define DATE_MINUS_INFINITE         0x0000      /* lowest boundary        */
#define DATE_PLUS_INFINITE          0xFFFF      /* highest upper boundary */
/*-------------------------------------------------------------------------+
| Bits masks for update action parameter on dsmUpdateFS()                  |
+-------------------------------------------------------------------------*/
#define DSM_FSUPD_FSTYPE              ((unsigned) 0x00000002)
#define DSM_FSUPD_FSINFO              ((unsigned) 0x00000004)
#define DSM_FSUPD_BACKSTARTDATE       ((unsigned) 0x00000008)
#define DSM_FSUPD_BACKCOMPLETEDATE    ((unsigned) 0x00000010)
#define DSM_FSUPD_OCCUPANCY           ((unsigned) 0x00000020)
#define DSM_FSUPD_CAPACITY            ((unsigned) 0x00000040)
#define DSM_FSUPD_RESERVED1           ((unsigned) 0x00000100)
/*-------------------------------------------------------------------------+
| Bits mask  for backup update action parameter on dsmUpdateObj()          |
+-------------------------------------------------------------------------*/
#define DSM_BACKUPD_OWNER              ((unsigned) 0x00000001)
#define DSM_BACKUPD_OBJINFO            ((unsigned) 0x00000002)
#define DSM_BACKUPD_MC                 ((unsigned) 0x00000004)
#define DSM_ARCHUPD_OWNER              ((unsigned) 0x00000001)
#define DSM_ARCHUPD_OBJINFO            ((unsigned) 0x00000002)
#define DSM_ARCHUPD_DESCR              ((unsigned) 0x00000004)
/*-------------------------------------------------------------------------+
|  Values for repository parameter on dsmDeleteFS()                        |
+-------------------------------------------------------------------------*/
#define DSM_ARCHIVE_REP       0x0A      /* archive repository             */
#define DSM_BACKUP_REP        0x0B      /* backup repository              */
#define DSM_REPOS_ALL         0x01      /* all respository types          */
/*-------------------------------------------------------------------------+
|  Values for vote parameter on dsmEndTxn()                                |
+-------------------------------------------------------------------------*/
#define DSM_VOTE_COMMIT  1             /* commit current transaction      */
#define DSM_VOTE_ABORT   2             /* roll back current transaction   */
/*-------------------------------------------------------------------------+
|  Values for various flags returned in ApiSessInfo structure.             |
+-----------------------------------------------------------------------*/
/* Client compression field codes */
#dfine COMPRESS_YES    1    /* client must compress data             */
#define COMPRESS_NO     2    /* client must NOT compress data         */
#define COMPRESS_CD     3    /* client determined                     */
/* Archive delete permission codes. */
#define ARCHDEL_YES     1    /* archive delete allowed                */
#define ARCHDEL_NO      2    /* archive delete NOT allowed            */
/* Backup delete permission codes. */
#define BACKDEL_YES     1    /* backup delete allowed                 */
#define BACKDEL_NO      2    /* backup delete NOT allowed             */
/*-------------------------------------------------------------------+
  Values for various flags returned in optStruct structure.          |
-------------------------------------------------------------------*/
#define DSM_PASSWD_GENERATE  1
#define DSM_PASSWD_PROMPT    0
#define DSM_COMM_TCP        1    /* tcpip         */
#define DSM_COMM_NAMEDPIPE  2    /* Named pipes   */
#define DSM_COMM_SHM        3    /* Shared Memory */
/* obsolete commmethods */
#define DSM_COMM_PVM_IUCV   12
#define DSM_COMM_3270       12
#define DSM_COMM_IUCV       12
#define DSM_COMM_PWSCS      12
#define DSM_COMM_SNA_LU6_2  12
#define DSM_COMM_IPXSPX     12    /* For IPX/SPX support */
#define DSM_COMM_NETBIOS    12    /* NETBIOS */
#define DSM_COMM_400COMM    12
#define DSM_COMM_CLIO       12    /* CLIO/S */
/*-------------------------------------------------------------------+
|  Values for userNameAuthorities in dsmInitEx for future use          |
+-------------------------------------------------------------------*/
#define DSM_USERAUTH_NONE     ((dsInt16_t)0x0000)
#define DSM_USERAUTH_ACCESS   ((dsInt16_t)0x0001)
#define DSM_USERAUTH_OWNER    ((dsInt16_t)0x0002)
#define DSM_USERAUTH_POLICY   ((dsInt16_t)0x0004)
#define DSM_USERAUTH_SYSTEM   ((dsInt16_t)0x0008)
/*-------------------------------------------------------------------+
|  Values for encryptionType on dsmEndSendObjEx, queryResp             |
+-------------------------------------------------------------------*/
#define DSM_ENCRYPT_NO              ((dsUint8_t)0x00)
#define DSM_ENCRYPT_USER            ((dsUint8_t)0x01)
#define DSM_ENCRYPT_CLIENTENCRKEY   ((dsUint8_t)0x02)
#define DSM_ENCRYPT_DES_56BIT       ((dsUint8_t)0x04)
#define DSM_ENCRYPT_AES_128BIT      ((dsUint8_t)0x08)
/*-------------------------------------------------------------------+
| Definitions for mediaClass field.                                  |
+-------------------------------------------------------------------*/
/*
 *  The following constants define a hierarchy of media access classes.
 *  Lower numbers indicate media which can supply faster access to data.
 */
/* Fixed: represents the class of on-line, fixed media (such as
          hard disks). */
#define  MEDIA_FIXED          0x10
/* Library: represents the class of mountable media accessible
            through a mechanical mounting device. */
#define  MEDIA_LIBRARY        0x20
/* future use */
#define  MEDIA_NETWORK        0x30
/* future use */
#define  MEDIA_SHELF          0x40
/* future use */
#define  MEDIA_OFFSITE        0x50
/* future use */
#define  MEDIA_UNAVAILABLE    0xF0
```

```
/*--------------------------------------------------------------------------+
| Type definition for partial object data for dsmBeginGetData()             |
+--------------------------------------------------------------------------*/
typedef struct
{
    dsUint16_t     stVersion;          /* Structure version                 */
    dsStruct64_t   partialObjOffset;   /* offset into object to begin reading*/
    dsStruct64_t   partialObjLength;   /* amount of object to read          */
} PartialObjData ;               /* partial object data                    */
#define PartialObjDataVersion 1  /*                                        */
/*--------------------------------------------------------------------------+
|  Type definition for date structure                                       |
+--------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t    year;                  /* year, 16-bit integer (e.g., 1990) */
   dsUint8_t     month;                 /* month, 8-bit integer (1 - 12)     */
   dsUint8_t     day;                   /* day. 8-bit integer (1 - 31)       */
   dsUint8_t     hour;                  /* hour, 8-bit integer (0 - 23)      */
   dsUint8_t     minute;                /* minute, 8-bit integer (0 - 59)    */
   dsUint8_t     second;                /* second, b-bit integer (0 - 59)    */
}dsDate ;
/*--------------------------------------------------------------------------+
|  Type definition for Object ID on dsmGetObj() and in dsmGetList structure|
+--------------------------------------------------------------------------*/
typedef dsStruct64_t  ObjID ;
/*--------------------------------------------------------------------------+
| Type definition for dsmQueryBuff on dsmBeginQuery()                       |
+--------------------------------------------------------------------------*/
typedef void dsmQueryBuff ;
/*--------------------------------------------------------------------------+
| Type definition for dsmGetType parameter on dsmBeginGetData()             |
+--------------------------------------------------------------------------*/
typedef enum
{
        gtBackup = 0x00,                     /* Backup processing type    */
        gtArchive                            /* Archive processing type   */
} dsmGetType ;
/*--------------------------------------------------------------------------+
|  Type definition for dsmQueryType parameter on dsmBeginQuery()            |
+--------------------------------------------------------------------------*/
typedef enum
{
   qtArchive = 0x00,                  /* Archive query type                 */
   qtBackup,                          /* Backup query type                  */
   qtBackupActive,                    /* Fast query for active backup files */
   qtFilespace,                       /* Filespace query type               */
   qtMC,                              /* Mgmt. class query type             */
   qtReserved1,                       /* future use                         */
   qtReserved2,                       /* future use                         */
   qtReserved3,                       /* future use                         */
   qtReserved4,                       /* future use                         */
   qtBackupGroups,                    /* group leaders in a specific fs     */
   qtOpenGroups,                      /* Open groups in a specific fs       */
   qtReserved5,                       /* future use                         */
   qtProxyNodeAuth,                   /* nodes that his node can proxy to   */
   qtProxyNodePeer                    /* Peer nodes with the same target    */
}dsmQueryType ;
/*--------------------------------------------------------------------------+
|  Type definition sendType parameter on dsmBindMC() and dsmSendObj()       |
+--------------------------------------------------------------------------*/
typedef enum
{
   stBackup = 0x00,                         /* Backup processing type    */
   stArchive,                               /* Archive processing type   */
   stBackupMountWait,           /* Backup processing with mountwait on   */
   stArchiveMountWait           /* Archive processing with mountwait on  */
```

```
}dsmSendType ;
/*-------------------------------------------------------------------------+
|  Type definition for delType parameter on dsmDeleteObj()                 |
+-------------------------------------------------------------------------*/
typedef enum
{
   dtArchive = 0x00,                               /* Archive delete type */
   dtBackup,                            /* Backup delete (deactivate) type */
   dtBackupID                          /* Backup delete (remove)     type */
}dsmDelType ;
/*-------------------------------------------------------------------------+
|  Type definition sendType parameter on dsmSetAccess()                    |
+-------------------------------------------------------------------------*/
typedef enum
{
   atBackup = 0x00,                         /* Backup processing type    */
   atArchive                               /* Archive processing type   */
}dsmAccessType;
/*-------------------------------------------------------------------------+
|  Type definition for API Version on dsmInit() and dsmQueryApiVersion()   |
+-------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t version;       /* API version                      */
   dsUint16_t release;       /* API release                      */
   dsUint16_t level;         /* API level                        */
}dsmApiVersion;
/*-------------------------------------------------------------------------+
|  Type definition for API Version on dsmInit() and dsmQueryApiVersion()   |
+-------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t stVersion;     /* Structure version                */
   dsUint16_t version;       /* API version                      */
   dsUint16_t release;       /* API release                      */
   dsUint16_t level;         /* API level                        */
   dsUint16_t subLevel;      /* API sub level                    */
   dsmBool_t  unicode;       /* API unicode?                     */
}dsmApiVersionEx;
#define apiVersionExVer       2
/*-------------------------------------------------------------------------+
|  Type definition for object name used on BindMC, Send, Delete, Query     |
+-------------------------------------------------------------------------*/
typedef struct S_dsmObjName
{
   char       fs[DSM_MAX_FSNAME_LENGTH + 1] ;               /* Filespace name */
   char       hl[DSM_MAX_HL_LENGTH + 1] ;                  /* High level name */
   char       ll[DSM_MAX_LL_LENGTH + 1] ;                   /* Low level name */
   dsUint8_t  objType;          /* for object type values, see defines above */
}dsmObjName;
/*-------------------------------------------------------------------------+
|  Type definition for Backup delete info on dsmDeleteObj()                |
+-------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t       stVersion ;                    /* structure version      */
   dsmObjName       *objNameP ;                    /* object name            */
   dsUint32_t       copyGroup ;                    /* copy group             */
}delBack ;
#define  delBackVersion    1
/*-------------------------------------------------------------------------+
|  Type definition for Archive delete info on dsmDeleteObj()               |
+-------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t       stVersion ;                    /* structure version   */
   dsStruct64_t       objId ;                        /* object ID         */
```

```
}delArch ;
#define  delArchVersion  1
/*--------------------------------------------------------------------------+
|  Type definition for Backup ID delete info on dsmDeleteObj()              |
+--------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t       stVersion ;                     /* structure version   */
   dsStruct64_t       objId ;                         /* object ID         */
}delBackID;
#define  delBackIDVersion  1
/*--------------------------------------------------------------------------+
|  Type definition for delete info on dsmDeleteObj()                        |
+--------------------------------------------------------------------------*/
typedef union
{
   delBack   backInfo ;
   delArch   archInfo ;
   delBackID backIDInfo ;
}dsmDelInfo ;
/*--------------------------------------------------------------------------+
|  Type definition for Object Attribute parameter on dsmSendObj()           |
+--------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t   stVersion;                               /* Structure version */
   char         owner[DSM_MAX_OWNER_LENGTH + 1];      /* object owner */
   dsStruct64_t sizeEstimate;     /* Size estimate in bytes of the object */
   dsmBool_t    objCompressed;         /* Is object already compressed? */
   dsUint16_t   objInfoLength;            /* length of object-dependent info */
   char         *objInfo;      /* object-dependent info */
   char         *mcNameP;                 /* mgmnt class name for override  */
}ObjAttr;
#define ObjAttrVersion  2
/*--------------------------------------------------------------------------+
| Type definition for mcBindKey returned on dsmBindMC()                     |
+--------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t  stVersion;                    /* structure version          */
   char        mcName[DSM_MAX_MC_NAME_LENGTH + 1];
                                            /* Name of mc bound to object. */
   dsmBool_t   backup_cg_exists;                             /* True/false */
   dsmBool_t   archive_cg_exists;                            /* True/false */
   char        backup_copy_dest[DSM_MAX_CG_DEST_LENGTH + 1];
                                                 /* Backup copy dest. name */
   char        archive_copy_dest[DSM_MAX_CG_DEST_LENGTH + 1];
                                                     /* Arch copy dest.name */
}mcBindKey;
#define mcBindKeyVersion  1
/*--------------------------------------------------------------------------+
| Type definition for object list on dsmBeginGetData()                      |
+--------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t       stVersion ;          /* structure version            */
   dsUint32_t       numObjId ;           /* number of object IDs in the list */
   ObjID           *objId ;              /* list of object IDs to restore*/
   PartialObjData  *partialObjData;      /*list of partial obj data info */
}dsmGetList ;
#define dsmGetListVersion     2  /* default if not using Partial Obj data */
#define dsmGetListPORVersion  3  /* version if using Partial Obj data     */
/*--------------------------------------------------------------------------+
|  Type definition for DataBlk used to Get or Send data                     |
+--------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t stVersion ;                /* structure version            */
   dsUint32_t bufferLen;                 /* Length of buffer passed below */
   dsUint32_t numBytes;                  /* Actual number of bytes read from */
                                         /* or written to the buffer */
   char       *bufferPtr;                /* Data buffer */
   dsUint32_t numBytesCompressed;        /* on send actual bytes compressed */
}DataBlk;
#define DataBlkVersion  2
/*-------------------------------------------------------------------------+
|  Type definition for Mgmt Class queryBuffer on dsmBeginQuery()           |
+-------------------------------------------------------------------------*/
typedef struct S_qryMCData
{
   dsUint16_t   stVersion;                              /* structure version */
   char         *mcName;                       /* Mgmt class name */
                         /* single name to get one or empty string to get all*/
   dsBool_t     mcDetail;                            /* Want details or not? */
}qryMCData;
#define qryMCDataVersion  1
/*=== values for RETINIT ===*/
#define ARCH_RETINIT_CREATE 0
#define ARCH_RETINIT_EVENT  1
/*-------------------------------------------------------------------------+
|  Type definition for Archive Copy Group details on Query MC response     |
+-------------------------------------------------------------------------*/
typedef struct S_archDetailCG
{
   char         cgName[DSM_MAX_CG_NAME_LENGTH + 1];       /* Copy group name */
   dsUint16_t   frequency;                       /* Copy (archive) frequency */
   dsUint16_t   retainVers;                                /* Retain version */
   dsUint8_t    copySer;       /* for copy serialization values, see defines */
   dsUint8_t    copyMode;         /* for copy mode values, see defines above */
   char         destName[DSM_MAX_CG_DEST_LENGTH + 1];      /* Copy dest name */
   dsBool_t     bLanFreeDest;           /* Destination has lan free path?    */
   dsBool_t     reserved;                              /* Not currently used */
   dsUint8_t    retainInit;              /* possible values see above        */
   dsUint16_t   retainMin;                /* if retInit is EVENT num of days */
}archDetailCG;
/*-------------------------------------------------------------------------+
|  Type definition for Backup Copy Group details on Query MC response      |
+-------------------------------------------------------------------------*/
typedef struct S_backupDetailCG
{
   char         cgName[DSM_MAX_CG_NAME_LENGTH + 1];       /* Copy group name */
   dsUint16_t   frequency;                               /* Backup frequency */
   dsUint16_t   verDataExst;                         /* Versions data exists */
   dsUint16_t   verDataDltd;                        /* Versions data deleted */
   dsUint16_t   retXtraVers;                        /* Retain extra versions */
   dsUint16_t   retOnlyVers;                         /* Retain only versions */
   dsUint8_t    copySer;       /* for copy serialization values, see defines */
   dsUint8_t    copyMode;         /* for copy mode values, see defines above */
   char         destName[DSM_MAX_CG_DEST_LENGTH + 1];      /* Copy dest name */
   dsBool_t     bLanFreeDest;           /* Destination has lan free path?    */
   dsBool_t     reserved;                              /* Not currently used */
}backupDetailCG;
/*-------------------------------------------------------------------------+
|  Type definition for Query Mgmt Class detail response on dsmGetNextQObj()|
+-------------------------------------------------------------------------*/
typedef struct S_qryRespMCDetailData
{
   dsUint16_t      stVersion;                       /* structure version */
   char           mcName[DSM_MAX_MC_NAME_LENGTH + 1];        /* mc name */
   char           mcDesc[DSM_MAX_MC_DESCR_LENGTH + 1]; /*mc description */
   archDetailCG    archDet;                  /* Archive copy group detail */
   backupDetailCG  backupDet;                 /* Backup copy group detail */
}qryRespMCDetailData;
#define qryRespMCDetailDataVersion  3
```

```
/*-------------------------------------------------------------------------+
| Type definition for Query Mgmt Class summary response on dsmGetNextQObj()|
+-------------------------------------------------------------------------*/
typedef struct S_qryRespMCData
{
   dsUint16_t   stVersion;                              /* structure version */
   char         mcName[DSM_MAX_MC_NAME_LENGTH + 1];               /* mc name */
   char         mcDesc[DSM_MAX_MC_DESCR_LENGTH + 1];       /* mc description */
}qryRespMCData;
#define qryRespMCDataVersion  1
/*-------------------------------------------------------------------------+
|  Type definition for Archive queryBuffer on dsmBeginQuery()              |
+-------------------------------------------------------------------------*/
typedef struct S_qryArchiveData
{
   dsUint16_t   stVersion;                          /* structure version */
   dsmObjName   *objName;                     /* Full dsm name of object */
   char         *owner;                                    /* owner name */
                       /* for maximum date boundaries, see defines above */
   dsmDate      insDateLowerBound;      /* low bound archive insert date */
   dsmDate      insDateUpperBound;       /* hi bound archive insert date */
   dsmDate      expDateLowerBound;          /* low bound expiration date */
   dsmDate      expDateUpperBound;           /* hi bound expiration date */
   char         *descr;                /* archive description */
} qryArchiveData;
#define qryArchiveDataVersion  1
/*=== values for retentionInitiated field ===*/
#define DSM_ARCH_RETINIT_UNKNOWN 0 /* ret init is unknown (down-level srv)  */
#define DSM_ARCH_RETINIT_STARTED 1 /* retention clock is started            */
#define DSM_ARCH_RETINIT_PENDING 2 /* retention clock is not started        */
/*=== Values for objHeld ===*/
#define DSM_ARCH_HELD_UNKNOWN 0    /* unknown hold status (down-level srv)  */
#define DSM_ARCH_HELD_FALSE   1    /* object is NOT in a delete hold state  */
#define DSM_ARCH_HELD_TRUE    2    /* object is in a delete hold state      */
/*-------------------------------------------------------------------------+
| Type definition for Query Archive response on dsmGetNextQObj()           |
+-------------------------------------------------------------------------*/
typedef struct S_qryRespArchiveData
{
   dsUint16_t      stVersion;                           /* structure version */
   dsmObjName      objName;                      /* Filespace name qualifier */
   dsUint32_t      copyGroup;                           /* copy group number */
   char            mcName[DSM_MAX_MC_NAME_LENGTH + 1];            /* mc name */
   char            owner[DSM_MAX_OWNER_LENGTH + 1];            /* owner name */
   dsStruct64_t    objId;                                  /* Unique copy id */
   dsStruct64_t    reserved;                       /* backward compatability */
   dsUint8_t       mediaClass;                         /* media access class */
   dsmDate         insDate;                        /* archive insertion date */
   dsmDate         expDate;                    /* expiration date for object */
   char            descr[DSM_MAX_DESCR_LENGTH + 1];   /* archive description */
   dsUint16_t      objInfolen;             /* length of object-dependent info*/
   char            objInfo[DSM_MAX_OBJINFO_LENGTH];  /*object-dependent info */
   dsUint160_t     restoreOrderExt;                         /* restore order */
   dsStruct64_t    sizeEstimate;             /* size estimate stored by user*/
   dsUint8_t       compressType;                          /* Compression flag*/
   dsUint8_t       retentionInitiated;   /* object waiting on retention event*/
   dsUint8_t       objHeld;  /*object is on retention "hold" see values above*/
   dsUint8_t       encryptionType;                     /* type of encryption */
}qryRespArchiveData;
#define qryRespArchiveDataVersion  5
/*-------------------------------------------------------------------------+
|  Type definition for Archive sendBuff parameter on dsmSendObj()          |
+-------------------------------------------------------------------------*/
typedef struct S_sndArchiveData
{
   dsUint16_t   stVersion;                   /* structure version */
   char         *descr;                    /* archive description */
```

```
}sndArchiveData;
#define sndArchiveDataVersion  1
/*-------------------------------------------------------------------------+
|  Type definition for Backup queryBuffer on dsmBeginQuery()              |
+-------------------------------------------------------------------------*/
typedef struct S_qryBackupData
{
   dsUint16_t  stVersion;         /* structure version */
   dsmObjName  *objName;          /* full dsm name of object */
   char        *owner;            /* owner name */
   dsUint8_t   objState;          /* object state selector */
   dsmDate     pitDate;           /* Date value for point in time restore */
                                  /* for possible values, see defines above */
}qryBackupData;
#define qryBackupDataVersion  2
typedef struct
{
  dsUint8_t    reserved1;
  dsStruct64_t reserved2;
} reservedInfo_t;                 /* for future use */
/*-------------------------------------------------------------------------+
| Type definition for Query Backup response on dsmGetNextQObj()           |
+-------------------------------------------------------------------------*/
typedef struct S_qryRespBackupData
{
   dsUint16_t      stVersion;                           /* structure version */
   dsmObjName      objName;                       /* full dsm name of object */
   dsUint32_t      copyGroup;                           /* copy group number */
   char            mcName[DSM_MAX_MC_NAME_LENGTH + 1];            /* mc name */
   char            owner[DSM_MAX_OWNER_LENGTH + 1];            /* owner name */
   dsStruct64_t    objId;                              /* Unique object id */
   dsStruct64_t    reserved;                     /* backward compatability */
   dsUint8_t       mediaClass;                       /* media access class */
   dsUint8_t       objState;                     /* Obj state, active, etc. */
   dsmDate         insDate;                        /* backup insertion date */
   dsmDate         expDate;                   /* expiration date for object */
   dsUint16_t      objInfolen;            /* length of object-dependent info*/
   char            objInfo[DSM_MAX_OBJINFO_LENGTH];  /*object-dependent info */
   dsUint160_t     restoreOrderExt;                         /* restore order */
   dsStruct64_t    sizeEstimate;            /* size estimate stored by user */
   dsStruct64_t    baseObjId;
   dsUint16_t      baseObjInfolen;            /* length of base object-dependent
 info*/
   dsUint8_t       baseObjInfo[DSM_MAX_OBJINFO_LENGTH];  /* base object-dependent
 info */
   dsUint160_t     baseRestoreOrder;                         /* restore order */
   dsUint32_t      fsID;
   dsUint8_t       compressType;
   dsmBool_t       isGroupLeader;
   dsmBool_t       isOpenGroup;
   dsUint8_t       reserved1;               /* for future use */
   dsmBool_t       reserved2;               /* for future use */
   dsUint16_t      reserved3;               /* for future use */
   reservedInfo_t  *reserved4;              /* for future use */
   dsUint8_t       encryptionType;                     /* type of encryption */
}qryRespBackupData;
#define qryRespBackupDataVersion  6
/*-------------------------------------------------------------------------+
|  Type definition for Active Backup queryBuffer on dsmBeginQuery()
|
|  Notes:  For the active backup query, only the fs (filespace) and objType
|          fields of objName need be set.  objType can only be set to
|          DSM_OBJ_FILE or DSM_OBJ_DIRECTORY.  DSM_OBJ_ANY_TYPE will not
|          find a match on the query.
+-------------------------------------------------------------------------*/
typedef struct S_qryABackupData
{
   dsUint16_t      stVersion;                          /* structure version */
   dsmObjName      *objName;                    /* Only fs and objtype used */
}qryABackupData;
#define qryABackupDataVersion  1
/*-------------------------------------------------------------------------+
| Type definition for Query Active Backup response on dsmGetNextQObj()     |
+-------------------------------------------------------------------------*/
typedef struct S_qryARespBackupData
{
   dsUint16_t  stVersion;                          /* structure version */
   dsmObjName  objName;                     /* full dsm name of object */
   dsUint32_t  copyGroup;                          /* copy group number */
   char        mcName[DSM_MAX_MC_NAME_LENGTH + 1];/*management class name*/
   char        owner[DSM_MAX_OWNER_LENGTH + 1];            /* owner name */
   dsmDate     insDate;                       /* backup insertion date */
   dsUint16_t  objInfolen;             /* length of object-dependent info*/
   char        objInfo[DSM_MAX_OBJINFO_LENGTH];  /*object-dependent info */
}qryARespBackupData;
#define qryARespBackupDataVersion  1
/*-------------------------------------------------------------------------+
|  Type definition for Backup queryBuffer on dsmBeginQuery()               |
+-------------------------------------------------------------------------*/
typedef struct qryBackupGroups
{
   dsUint16_t   stVersion;         /* structure version */
   dsUint8_t    groupType;
   char         *fsName;
   char         *owner;
   dsStruct64_t groupLeaderObjId;
   dsUint8_t    objType;
}qryBackupGroups;
#define qryBackupGroupsVersion 1
/*-------------------------------------------------------------------------+
|  Type definition for proxynode queryBuffer on dsmBeginQuery()            |
+-------------------------------------------------------------------------*/
typedef struct qryProxyNodeData
{
   dsUint16_t  stVersion;                  /* structure version */
   char        *targetNodeName;            /* target node name  */
}qryProxyNodeData;
#define qryProxyNodeDataVersion  1
/*-------------------------------------------------------------------------+
| Type definition for qryRespProxyNodeData  parameter used on dsmGetNextQObj()|
+-------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t      stVersion ;                         /* structure version */
   char            targetNodeName[DSM_MAX_ID_LENGTH+1]; /* target node name  */
   char            peerNodeName[DSM_MAX_ID_LENGTH+1];   /* Peer node name    */
   char            hlAddress[DSM_MAX_ID_LENGTH+1];      /* peer hlAddress    */
   char            llAddress[DSM_MAX_ID_LENGTH+1];      /* peer hlAddress    */
}qryRespProxyNodeData;
#define qryRespProxyNodeDataVersion  1
/*-------------------------------------------------------------------------+
|  Type definition for WINNT and OS/2 Filespace attributes                 |
+-------------------------------------------------------------------------*/
typedef struct
{
   char         driveLetter ;          /* drive letter for filespace    */
   dsUint16_t   fsInfoLength;          /* fsInfo length used            */
   char         fsInfo[DSM_MAX_FSINFO_LENGTH];/*caller-determined data  */
}dsmDosFSAttrib ;
/*-------------------------------------------------------------------------+
|  Type definition for UNIX Filespace attributes                           |
+-------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t   fsInfoLength;          /* fsInfo length used            */
   char         fsInfo[DSM_MAX_FSINFO_LENGTH];/*caller-determined data  */
}dsmUnixFSAttrib ;
/*-------------------------------------------------------------------------+
|  Type definition for NetWare Filespace attributes                        |
+-------------------------------------------------------------------------*/
typedef dsmUnixFSAttrib dsmNetwareFSAttrib;
/*-------------------------------------------------------------------------+
|  Type definition for Filespace attributes on all Filespace calls         |
+-------------------------------------------------------------------------*/
typedef union
{
   dsmNetwareFSAttrib  netwareFSAttr;
   dsmUnixFSAttrib     unixFSAttr ;
   dsmDosFSAttrib      dosFSAttr ;
}dsmFSAttr ;
/*-------------------------------------------------------------------------+
|  Type definition for fsUpd parameter on dsmUpdateFS()
+-------------------------------------------------------------------------*/
typedef struct S_dsmFSUpd
{
   dsUint16_t      stVersion ;             /* structure version              */
   char            *fsType ;               /* filespace type                 */
   dsStruct64_t    occupancy ;             /* occupancy estimate             */
   dsStruct64_t    capacity  ;             /* capacity estimate              */
   dsmFSAttr       fsAttr ;                /* platform specific attributes   */
}dsmFSUpd ;
#define dsmFSUpdVersion  1
/*-------------------------------------------------------------------------+
|  Type definition for Filespace queryBuffer on dsmBeginQuery()            |
+-------------------------------------------------------------------------*/
typedef struct S_qryFSData
{
   dsUint16_t  stVersion;                 /* structure version */
   char        *fsName;                   /* File space name */
}qryFSData;
#define qryFSDataVersion  1
/*-------------------------------------------------------------------------+
| Type definition for Query Filespace response on dsmGetNextQObj()         |
+-------------------------------------------------------------------------*/
typedef struct S_qryRespFSData
{
   dsUint16_t     stVersion;                              /* structure version */
   char           fsName[DSM_MAX_FSNAME_LENGTH + 1];         /* Filespace name */
   char           fsType[DSM_MAX_FSTYPE_LENGTH + 1] ;        /* Filespace type */
   dsStruct64_t   occupancy;                       /* Occupancy est. in bytes. */
   dsStruct64_t   capacity;                         /* Capacity est. in bytes. */
   dsmFSAttr      fsAttr ;                    /* platform specific attributes  */
   dsmDate        backStartDate;              /* start backup date             */
   dsmDate        backCompleteDate;           /* end backup Date               */
   dsmDate        reserved1;                   /* For future use               */
}qryRespFSData;
#define qryRespFSDataVersion 3
/*-------------------------------------------------------------------------+
|  Type definition for regFilespace parameter on dsmRegisterFS()
+-------------------------------------------------------------------------*/
typedef struct S_regFSData
{
   dsUint16_t      stVersion;                             /* structure version */
   char            *fsName;                        /* Filespace name */
   char            *fsType;                        /* Filespace type */
   dsStruct64_t    occupancy;                      /* Occupancy est. in bytes. */
   dsStruct64_t    capacity;                        /* Capacity est. in bytes. */
   dsmFSAttr       fsAttr ;                    /* platform specific attributes */
}regFSData;
#define regFSDataVersion  1
/*-------------------------------------------------------------------------+
|  Type definition for session info response on dsmQuerySessionInfo()      |
+-------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t     stVersion;             /* Structure version            */
      /*------------------------------------------------------------------*/
      /*           Server information                                     */
      /*------------------------------------------------------------------*/
   char           serverHost[DSM_MAX_SERVERNAME_LENGTH+1];
                                      /* Network host name of DSM server  */
   dsUint16_t     serverPort;            /* Server comm port on host     */
   dsmDate        serverDate;            /* Server's date/time           */
   char           serverType[DSM_MAX_SERVERTYPE_LENGTH+1];
                                     /* Server's execution platform       */
   dsUint16_t     serverVer;             /* Server's version number      */
   dsUint16_t     serverRel;             /* Server's release number          */
   dsUint16_t     serverLev;             /* Server's level number            */
   dsUint16_t     serverSubLev;          /* Server's sublevel number         */
      /*------------------------------------------------------------------*/
      /*           Client Defaults                                        */
      /*------------------------------------------------------------------*/
   char           nodeType[DSM_MAX_PLATFORM_LENGTH+1]; /*node/application type*/
   char           fsdelim;               /* File space delimiter             */
   char           hldelim;               /* Delimiter betw highlev & lowlev  */
   dsUint8_t      compression;           /* Compression flag                 */
   dsUint8_t      archDel;               /* Archive delete permission        */
   dsUint8_t      backDel;               /* Backup  delete permission        */
   dsUint32_t     maxBytesPerTxn;        /* for future use                   */
   dsUint16_t     maxObjPerTxn;          /* The max objects allowed in a txn */
      /*------------------------------------------------------------------*/
      /*           Session Information                                    */
      /*------------------------------------------------------------------*/
   char       id[DSM_MAX_ID_LENGTH+1];    /* Sign-in id node name         */
   char       owner[DSM_MAX_OWNER_LENGTH+1]; /* Sign-in owner             */
                                      /*   (for multi-user platforms)     */
   char       confFile[DSM_PATH_MAX + DSM_NAME_MAX +1];
                                     /* len is platform dep              */
                                     /* dsInit name of appl config file  */
   dsUint8_t  opNoTrace;              /* dsInit option - NoTrace = 1      */
      /*------------------------------------------------------------------*/
      /*           Policy Data                                            */
      /*------------------------------------------------------------------*/
   char           domainName[DSM_MAX_DOMAIN_LENGTH+1]; /* Domain name     */
   char           policySetName[DSM_MAX_PS_NAME_LENGTH+1];
                                      /* Active policy set name           */
   dsmDate        polActDate;            /* Policy set activation date       */
   char           dfltMCName[DSM_MAX_MC_NAME_LENGTH+1];/* Default Mgmt Class  */
   dsUint16_t     gpBackRetn;            /* Grace-period backup retention    */
   dsUint16_t     gpArchRetn;            /* Grace-period archive retention   */
   char           adsmServerName[DSM_MAX_SERVERNAME_LENGTH+1]; /* adsm server name */
   dsmBool_t      archiveRetentionProtection; /* is server Retention protection enabled */
}ApiSessInfo;
#define ApiSessInfoVersion  3
/*-------------------------------------------------------------------------+
| Type definition for Query options response on dsmQueryCliOptions()       |
|      and dsmQuerySessOptions()                                           |
+-------------------------------------------------------------------------*/
typedef struct
{
   char        dsmiDir[DSM_PATH_MAX + DSM_NAME_MAX +1];
   char        dsmiConfig[DSM_PATH_MAX + DSM_NAME_MAX +1];
   char        serverName[DSM_MAX_SERVERNAME_LENGTH+1];
   dsInt16_t   commMethod;
   char        serverAddress[DSM_MAX_SERVER_ADDRESS];
   char        nodeName[DSM_MAX_NODE_LENGTH+1];
   dsmBool_t   compression;
   dsmBool_t   compressalways;
   dsmBool_t   passwordAccess;
}optStruct ;
/*--------------------------------------------------------------------------+
| Type definition for LogType used in logInfo                               |
+--------------------------------------------------------------------------*/
typedef enum
{
   logServer = 0x00,             /* log msg only to server    */
   logLocal,                     /* log msg only to local error log */
   logBoth                       /* log msg to server and to local error log */
}dsmLogType ;
/*--------------------------------------------------------------------------+
| Type definition for logInfo parameter used on dsmLogEvent()               |
+--------------------------------------------------------------------------*/
typedef struct
{
   char        *message;   /* text of message to be logged */
   dsmLogType  logType;    /* log type : local, server, both */
}logInfo;
/*--------------------------------------------------------------------------+
| Type definition for qryRespAccessData  parameter used on dsmQueryAccess()|
+--------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t        stVersion ;                    /* structure version      */
   char              node[DSM_MAX_ID_LENGTH+1];     /* node name              */
   char              owner[DSM_MAX_OWNER_LENGTH+1]; /* owner                  */
   dsmObjName        objName ;                      /* object name            */
   dsmAccessType     accessType;                    /* archive or backup     */
   dsUint32_t        ruleNumber ;                   /* Access rule id         */
}qryRespAccessData;
#define qryRespAccessDataVersion  1
/*--------------------------------------------------------------------------+
|  Type definition for envSetUp parameter on dsmSetUp()
+--------------------------------------------------------------------------*/
typedef struct S_envSetUp
{
   dsUint16_t      stVersion;                      /* structure version */
   char            dsmiDir[DSM_PATH_MAX + DSM_NAME_MAX +1];
   char            dsmiConfig[DSM_PATH_MAX + DSM_NAME_MAX +1];
   char            dsmiLog[DSM_PATH_MAX + DSM_NAME_MAX +1];
   char          **argv;  /* for executables name argv[0] */
   char            logName[DSM_NAME_MAX +1];
   dsmBool_t       reserved1;         /* for future use */
   dsmBool_t       reserved2;                  /* for future use */
}envSetUp;
#define envSetUpVersion  4
/*--------------------------------------------------------------------------+
|  Type definition for dsmInitExIn_t
+--------------------------------------------------------------------------*/
typedef struct dsmInitExIn_t
{
   dsUint16_t         stVersion;                      /* structure version */
   dsmApiVersionEx    *apiVersionExP;
   char               *clientNodeNameP;
   char               *clientOwnerNameP;
   char               *clientPasswordP;
   char               *userNameP;
   char               *userPasswordP;
   char               *applicationTypeP;
   char               *configfile;
   char               *options;
   char               dirDelimiter;
   dsmBool_t          useUnicode;
   dsmBool_t          bCrossPlatform;
   dsmBool_t          bService;
   dsmBool_t          bEncryptKeyEnabled;
   char              *encryptionPasswordP;
   dsmBool_t          useTsmBuffers;
   dsUint8_t          numTsmBuffers;
}dsmInitExIn_t;
#define dsmInitExInVersion 4
/*-------------------------------------------------------------------------+
|  Type definition for dsmInitExOut_t
+-------------------------------------------------------------------------*/
typedef struct dsmInitExOut_t
{
   dsUint16_t         stVersion;                    /* structure version */
   dsInt16_t          userNameAuthorities;
   dsInt16_t          infoRC;        /* error return code if encountered */
   char               adsmServerName[DSM_MAX_SERVERNAME_LENGTH+1];
   dsUint16_t         serverVer;     /* Server's version number          */
   dsUint16_t         serverRel;     /* Server's release number          */
   dsUint16_t         serverLev;     /* Server's level number            */
   dsUint16_t         serverSubLev;  /* Server's sublevel number         */
}dsmInitExOut_t;
#define dsmInitExOutVersion 2
/*-------------------------------------------------------------------------+
| Type definition for LogType used in logInfo                              |
+-------------------------------------------------------------------------*/
typedef enum
{
   logSevInfo = 0x00,       /* information ANE4991 */
   logSevWarning,           /* warning     ANE4992 */
   logSevError,             /* Error       ANE4993 */
   logSevSevere,            /* severe      ANE4994 */
   logSevLicense,           /* License     ANE4995 */
   logSevTryBuy             /* try Buy     ANE4996 */
}dsmLogSeverity ;
/*-------------------------------------------------------------------------+
|  Type definition for dsmLogExIn_t
+-------------------------------------------------------------------------*/
typedef struct dsmLogExIn_t
{
   dsUint16_t         stVersion;  /* structure version */
   dsmLogSeverity     severity;
   char               appMsgID[8];
   dsmLogType         logType;    /* log type : local, server, both */
   char               *message;   /* text of message to be logged */
   char               appName[DSM_MAX_PLATFORM_LENGTH];
   char               osPlatform[DSM_MAX_PLATFORM_LENGTH];
   char               appVersion[DSM_MAX_PLATFORM_LENGTH];
}dsmLogExIn_t;
#define dsmLogExInVersion 2
/*-------------------------------------------------------------------------+
|  Type definition for dsmlogExOut_t
+-------------------------------------------------------------------------*/
typedef struct dsmLogExOut_t
{
   dsUint16_t         stVersion;  /* structure version */
}dsmLogExOut_t;
#define dsmLogExOutVersion 1
/*-------------------------------------------------------------------------+
|  Type definition for dsmRenameIn_t
+-------------------------------------------------------------------------*/
typedef struct dsmRenameIn_t
{
   dsUint16_t       stVersion;                     /* structure version  */
   dsUint32_t       dsmHandle;                     /* handle for session */
   dsUint8_t        repository;                    /* Backup or Archive  */
   dsmObjName       *objNameP ;                    /* object name */
   char             newHl[DSM_MAX_HL_LENGTH + 1];  /* new High level name */
   char             newLl[DSM_MAX_LL_LENGTH + 1];  /* new Low level name */
   dsmBool_t        merge;                        /* merge into existing name*/
   ObjID            objId;                         /* objId for Archive */
}dsmRenameIn_t;
#define dsmRenameInVersion 1
/*--------------------------------------------------------------------------+
|  Type definition for dsmRenameOut_t
+--------------------------------------------------------------------------*/
typedef struct dsmRenameOut_t
{
   dsUint16_t         stVersion;                      /* structure version */
}dsmRenameOut_t;
#define dsmRenameOutVersion 1
/*--------------------------------------------------------------------------+
|  Type definition for dsmEndSendObjExIn_t
+--------------------------------------------------------------------------*/
typedef struct dsmEndSendObjExIn_t
{
   dsUint16_t       stVersion;                     /* structure version  */
   dsUint32_t       dsmHandle;                     /* handle for session */
}dsmEndSendObjExIn_t;
#define dsmEndSendObjExInVersion 1
/*--------------------------------------------------------------------------+
|  Type definition for dsmEndSendObjExOut_t
+--------------------------------------------------------------------------*/
typedef struct dsmEndSendObjExOut_t
{
   dsUint16_t         stVersion;           /* structure version */
   dsStruct64_t       totalBytesSent;      /* total bytes read from app */
   dsmBool_t          objCompressed;       /* was object compressed */
   dsStruct64_t       totalCompressSize;   /* total size after compress */
   dsStruct64_t       totalLFBytesSent;    /* total bytes sent Lan Free */
   dsUint8_t          encryptionType;      /* type of encryption used   */
}dsmEndSendObjExOut_t;
#define dsmEndSendObjExOutVersion 2
/*--------------------------------------------------------------------------+
|  Type definition for dsmGroupHandlerIn_t
+--------------------------------------------------------------------------*/
typedef struct dsmGroupHandlerIn_t
{
   dsUint16_t      stVersion;        /* structure version  */
   dsUint32_t      dsmHandle;        /* handle for session */
   dsUint8_t       groupType;        /* Type of group                        */
   dsUint8_t       actionType;       /* Type of group operation              */
   dsUint8_t       memberType;       /* Type of member: Leader or member     */
   dsStruct64_t    leaderObjId;      /* OBJID of the groupleader when manipulating
 a member */
   char            *uniqueGroupTagP; /* Unique group identifier              */
   dsmObjName      *objNameP ;       /* group leader object name */
   dsmGetList      memberObjList;    /* list of objects to remove, assign   */
}dsmGroupHandlerIn_t;
#define dsmGroupHandlerInVersion 1
/*--------------------------------------------------------------------------+
|  Type definition for dsmGroupHandlerExOut_t
+--------------------------------------------------------------------------*/
typedef struct dsmGroupHandlerOut_t
{
   dsUint16_t         stVersion;                      /* structure version */
}dsmGroupHandlerOut_t;
#define dsmGroupHandlerOutVersion 1
/*--------------------------------------------------------------------------+
|  Type definition for dsmEndTxnExIn_t
+--------------------------------------------------------------------------*/
typedef struct dsmEndTxnExIn_t
{
   dsUint16_t       stVersion;                     /* structure version  */
   dsUint32_t       dsmHandle;                     /* handle for session */
   dsUint8_t        vote;
}dsmEndTxnExIn_t;
```

```
#define dsmEndTxnExInVersion 1
/*-------------------------------------------------------------------------+
|  Type definition for dsmEndTxnExOut_t
+-------------------------------------------------------------------------*/
typedef struct dsmEndTxnExOut_t
{
   dsUint16_t         stVersion;               /* structure version              */
   dsUint16_t         reason;                  /* reason code                    */
   dsStruct64_t       groupLeaderObjId;        /* groupLeader obj id returned on */
                                               /* DSM_ACTION_OPEN                */
   dsUint8_t          reserved1;               /* future use                     */
   dsUint16_t         reserved2;               /* future use                     */
}dsmEndTxnExOut_t;
#define dsmEndTxnExOutVersion 1
/*-------------------------------------------------------------------------+
|  Type definition for dsmEndGetDataExIn_t
+-------------------------------------------------------------------------*/
typedef struct dsmEndGetDataExIn_t
{
   dsUint16_t     stVersion;     /* structure version  */
   dsUint32_t     dsmHandle;     /* handle for session */
}dsmEndGetDataExIn_t;
#define dsmEndGetDataExInVersion 1
/*-------------------------------------------------------------------------+
|  Type definition for dsmEndGetDataExOut_t
+-------------------------------------------------------------------------*/
typedef struct dsmEndGetDataExOut_t
{
   dsUint16_t     stVersion;         /* structure version             */
   dsUint16_t     reason;            /* reason code                   */
   dsStruct64_t   totalLFBytesRecv;  /* total lan free bytes recieved */
}dsmEndGetDataExOut_t;
#define dsmEndGetDataExOutVersion 1
/*-------------------------------------------------------------------------+
| Type definition for object list on dsmRetentionEvent()                   |
+-------------------------------------------------------------------------*/
typedef struct dsmObjList
{
   dsUint16_t       stVersion;         /* structure version                */
   dsUint32_t       numObjId;          /* number of object IDs in the list */
   ObjID            *objId;            /* list of object IDs to signal     */
}dsmObjList_t ;
#define dsmObjlistVersion 1
/*-------------------------------------------------------------------------+
| Type definition eventType used on dsmRetentionEvent                      |
+-------------------------------------------------------------------------*/
typedef enum
{
   eventRetentionActivate = 0x00, /* signal the server that event has occured */
   eventHoldObj,                  /* suspend delete/expire of the object      */
   eventReleaseObj                /* Resume normal delete/expire processing   */
}dsmEventType_t;
/*-------------------------------------------------------------------------+
|  Type definition for  on dsmRetentionEvent()                             |
+-------------------------------------------------------------------------*/
typedef struct dsmRetentionEventIn_t
{
   dsUint16_t         stVersion;                    /* structure version  */
   dsUint32_t         dsmHandle;                    /* session Handle     */
   dsmEventType_t     eventType;                    /* Event type         */
   dsmObjList_t       objList;                      /* object ID          */
}dsmRetentionEventIn_t;
#define dsmRetentionEventInVersion 1
/*-------------------------------------------------------------------------+
|  Type definition for  on dsmRetentionEvent()              |
+-------------------------------------------------------------------------*/
typedef struct dsmRetentionEventOut_t
```

```
{
   dsUint16_t       stVersion ;                    /* structure version      */
}dsmRetentionEventOut_t;
#define dsmRetentionEventOutVersion 1
/*--------------------------------------------------------------------------+
|  Type definition for  on dsmRequestBuffer()                                   |
+--------------------------------------------------------------------------*/
typedef struct requestBufferIn_t
{
   dsUint16_t         stVersion;                    /* structure version  */
   dsUint32_t         dsmHandle;                    /* session Handle     */
}requestBufferIn_t;
#define requestBufferInVersion 1
/*--------------------------------------------------------------------------+
|  Type definition for  on dsmRequestBuffer()                                   |
+--------------------------------------------------------------------------*/
typedef struct requestBufferOut_t
{
   dsUint16_t       stVersion ;                /* structure version */
   dsUint8_t        tsmBufferHandle;           /* handle to tsm Data buffer */
   char             *dataPtr;                  /* Address to write data to */
   dsUint32_t       bufferLen;                 /* Max length of data to be written */
}requestBufferOut_t;
#define requestBufferOutVersion 1
/*--------------------------------------------------------------------------+
|  Type definition for  on dsmReleaseBuffer()                                |
+--------------------------------------------------------------------------*/
typedef struct releaseBufferIn_t
{
   dsUint16_t         stVersion;                    /* structure version  */
   dsUint32_t         dsmHandle;                    /* session Handle     */
   dsUint8_t          tsmBufferHandle;              /* handle to tsm Data buffer */
   char               *dataPtr;                     /* Address to write data to */
}releaseBufferIn_t;
#define releaseBufferInVersion 1
/*--------------------------------------------------------------------------+
|  Type definition for  on dsmReleaseBuffer()                                |
+--------------------------------------------------------------------------*/
typedef struct releaseBufferOut_t
{
   dsUint16_t       stVersion ;                    /* structure version */
}releaseBufferOut_t;

#define releaseBufferOutVersion 1

/*--------------------------------------------------------------------------+
|  Type definition for  on dsmGetBufferData()                                   |
+--------------------------------------------------------------------------*/
typedef struct getBufferDataIn_t
{
   dsUint16_t         stVersion;                    /* structure version  */
   dsUint32_t         dsmHandle;                    /* session Handle     */
}getBufferDataIn_t;
#define getBufferDataInVersion 1
/*--------------------------------------------------------------------------+
|  Type definition for  on dsmGetBufferData()                                     |
+--------------------------------------------------------------------------*/
typedef struct getBufferDataOut_t
{
   dsUint16_t       stVersion ;            /* structure version */
   dsUint8_t        tsmBufferHandle;       /* handle to tsm Data buffer */
   char             *dataPtr;              /* Address of actual data to read */
   dsUint32_t       numBytes;              /* Actual number of bytes to read from dataPtr*/
}getBufferDataOut_t;
#define getBufferDataOutVersion 1
/*--------------------------------------------------------------------------+
|  Type definition for  on dsmSendBufferData()                                  |
+---------------------------------------------------------------------------*/
typedef struct sendBufferDataIn_t
{
   dsUint16_t         stVersion;             /* structure version  */
   dsUint32_t         dsmHandle;             /* session Handle     */
   dsUint8_t          tsmBufferHandle;       /* handle to tsm Data buffer */
   char               *dataPtr;              /* Address of actual data to send */
   dsUint32_t         numBytes;              /* Actual number of bytes to send from dataPtr*/
}sendBufferDataIn_t;
#define sendBufferDataInVersion 1
/*---------------------------------------------------------------------------+
|  Type definition for  on dsmSendBufferData()                                  |
+---------------------------------------------------------------------------*/
typedef struct sendBufferDataOut_t
{
   dsUint16_t       stVersion ;            /* structure version */
}sendBufferDataOut_t;
#define sendBufferDataOutVersion 1
/*---------------------------------------------------------------------------+
|  Type definition for dsmUpdateObjExIn_t
+---------------------------------------------------------------------------*/
typedef struct dsmUpdateObjExIn_t
{
   dsUint16_t      stVersion;            /* structure version */
   dsUint32_t      dsmHandle;            /* session Handle      */
   dsmSendType     sendType;             /* send type back/arch */
   char            *descrP;              /* archive description */
   dsmObjName      *objNameP;            /* objName             */
   ObjAttr         *objAttrPtr;          /* attribute           */
   dsUint32_t      objUpdAct;            /* update action       */
   ObjID           archObjId;            /* objId for archive   */
}dsmUpdateObjExIn_t;
#define dsmUpdateObjExInVersion 1
/*---------------------------------------------------------------------------+
|  Type definition for dsmUpdateObjExOut_t
+---------------------------------------------------------------------------*/
typedef struct dsmUpdateObjExOut_t
{
    dsUint16_t         stVersion;        /* structure version */
}dsmUpdateObjExOut_t;
#define dsmUpdateObjExOutVersion 1
#if (_OPSYS_TYPE == DS_WINNT) && !defined(_WIN64)
#pragma pack()
#endif
#ifdef _MAC
#pragma options align=reset
#endif
#endif /* _H_DSMAPITD */ </mode></mode>
```

```
/************************************************************************
* Tivoli Storage Manager                                                *
* API Client Component                                                  *
*                                                                       *
* (C) Copyright IBM Corporation 1993,2008                               *
************************************************************************/
/***************************************************************************
* Header File Name: tsmapitd.h
*
* Environment:     *************************************************
*                  ** This is a platform-independent source file **
*
*                  *************************************************
*
* Design Notes:    This file contains basic data types and constants
*                  includable by all client source files. The constants
*                  within this file should be set properly for the
*                  particular machine and operating system on which the
*                  client software is to be run.
```

```
*
*                  Platform specific definitions are included in dsmapips.h
*
* Descriptive-name: Definitions for Tivoli Storage manager API constants
*-------------------------------------------------------------------------*/
#ifndef _H_TSMAPITD
#define _H_TSMAPITD
/*=== set the structure alignment to pack the structures ===*/
#if _OPSYS_TYPE == DS_WINNT
#ifdef _WIN64
#pragma pack(8)
#else
#pragma pack(1)
#endif
#endif
#ifdef _MAC
#pragma options align = packed
#endif
/*==============================================================
   Win32 applications using the tsm interface must use the
   -DUNICODE flag during compilation.
==============================================================*/
#if _OPSYS_TYPE == DS_WINNT && !defined(DSMAPILIB)
#ifndef UNICODE
#error "Win32 applications using the TSM interface MUST be compiled with the -DUNICODE flag"
#endif
#endif
/*==============================================================
   Mac OS X applications using the tsm interface must use the
   -DUNICODE flag during compilation.
==============================================================*/
#if _OPSYS_TYPE == DS_MACOS && !defined(DSMAPILIB)
#ifndef UNICODE
#error "Mac OS X applications using the TSM interface MUST be compiled with the -DUNICODE flag"
#endif
#endif
/*-------------------------------------------------------------------------+
| Type definition for dsmGetType parameter on tsmBeginGetData()            |
+-------------------------------------------------------------------------*/
typedef enum
{
    gtTsmBackup = 0x00,                     /* Backup processing type    */
    gtTsmArchive                            /* Archive processing type   */
} tsmGetType ;
/*-------------------------------------------------------------------------+
|  Type definition for dsmQueryType parameter on tsmBeginQuery()           |
+-------------------------------------------------------------------------*/
typedef enum
{
    qtTsmArchive = 0x00,                     /* Archive query type        */
    qtTsmBackup,                             /* Backup query type         */
    qtTsmBackupActive,                       /* Fast query for active backup files */
    qtTsmFilespace,                          /* Filespace query type      */
    qtTsmMC,                                 /* Mgmt. class query type    */
    qtTsmReserved1,                          /* future use                */
    qtTsmReserved2,                          /* future use                */
    qtTsmReserved3,                          /* future use                */
    qtTsmReserved4,                          /* future use                */
    qtTsmBackupGroups,                       /* All group leaders in a specific filespace */
    qtTsmOpenGroups,                         /* All group members associated with a leader */
    qtTsmReserved5,                          /* future use                */
    qtTsmProxyNodeAuth,                      /* nodes that this node can proxy to   */
    qtTsmProxyNodePeer                       /* peer nodes under this target node  */
} tsmQueryType ;
/*-------------------------------------------------------------------------+
|  Type definition sendType parameter on tsmBindMC() and tsmSendObj()      |
+-------------------------------------------------------------------------*/
```

```
typedef enum
{
    stTsmBackup = 0x00,                         /* Backup processing type    */
    stTsmArchive,                               /* Archive processing type   */
    stTsmBackupMountWait,           /* Backup processing with mountwait on   */
    stTsmArchiveMountWait           /* Archive processing with mountwait on  */
} tsmSendType ;
/*-------------------------------------------------------------------------+
|  Type definition for delType parameter on tsmDeleteObj()                |
+-------------------------------------------------------------------------*/
typedef enum
{
    dtTsmArchive = 0x00,                              /* Archive delete type */
    dtTsmBackup,                           /* Backup delete (deactivate) type */
    dtTsmBackupID                         /* Backup delete (remove)     type */
} tsmDelType ;
/*-------------------------------------------------------------------------+
|  Type definition sendType parameter on tsmSetAccess()                    |
+-------------------------------------------------------------------------*/
typedef enum
{
    atTsmBackup = 0x00,                         /* Backup processing type    */
    atTsmArchive                               /* Archive processing type   */
}tsmAccessType;
/*-------------------------------------------------------------------------+
|  Type definition for Overwrite  parameter on tsmSendObj()                |
+-------------------------------------------------------------------------*/
typedef enum
{
    owIGNORE = 0x00,
    owYES,
    owNO
}tsmOwType;
/*-------------------------------------------------------------------------+
|  Type definition for API Version on tsmInit() and tsmQueryApiVersion()   |
+-------------------------------------------------------------------------*/
typedef struct
{
        dsUint16_t stVersion;     /* Structure version                */
        dsUint16_t version;       /* API version                      */
        dsUint16_t release;       /* API release                      */
        dsUint16_t level;         /* API level                        */
        dsUint16_t subLevel;      /* API sub level                    */
        dsmBool_t  unicode;       /* API unicode?                     */
} tsmApiVersionEx;
#define tsmApiVersionExVer       2
/*-------------------------------------------------------------------------+
|  Type definition for object name used on BindMC, Send, Delete, Query     |
+-------------------------------------------------------------------------*/
typedef struct tsmObjName
{
    dsChar_t   fs[DSM_MAX_FSNAME_LENGTH + 1] ;              /* Filespace name */
    dsChar_t   hl[DSM_MAX_HL_LENGTH + 1] ;                 /* High level name */
    dsChar_t   ll[DSM_MAX_LL_LENGTH + 1] ;                  /* Low level name */
    dsUint8_t  objType;          /* for object type values, see defines above */
    dsChar_t   dirDelimiter;
} tsmObjName;
/*-------------------------------------------------------------------------+
|  Type definition for Backup delete info on dsmDeleteObj()                |
+-------------------------------------------------------------------------*/
typedef struct tsmDelBack
{
    dsUint16_t       stVersion ;                    /* structure version      */
    tsmObjName       *objNameP ;                    /* object name            */
    dsUint32_t       copyGroup ;                    /* copy group             */
} tsmDelBack ;
#define  tsmDelBackVersion    1
```

```
/*--------------------------------------------------------------------------+
|  Type definition for Archive delete info on dsmDeleteObj()                |
+--------------------------------------------------------------------------*/
typedef struct
{
    dsUint16_t       stVersion ;                    /* structure version      */
    dsStruct64_t     objId ;                        /* object ID              */
} tsmDelArch ;
#define  tsmDelArchVersion  1
/*--------------------------------------------------------------------------+
|  Type definition for Backup ID delete info on dsmDeleteObj()              |
+--------------------------------------------------------------------------*/
typedef struct
{
    dsUint16_t       stVersion ;                    /* structure version      */
    dsStruct64_t       objId ;                        /* object ID              */
} tsmDelBackID;
#define  tsmDelBackIDVersion  1
/*--------------------------------------------------------------------------+
|  Type definition for delete info on dsmDeleteObj()                        |
+--------------------------------------------------------------------------*/
typedef union
{
    tsmDelBack   backInfo ;
    tsmDelArch   archInfo ;
    tsmDelBackID backIDInfo;
} tsmDelInfo ;
/*--------------------------------------------------------------------------+
|  Type definition for Object Attribute parameter on dsmSendObj()           |
+--------------------------------------------------------------------------*/
typedef struct tsmObjAttr
{
    dsUint16_t   stVersion;                /* Structure version                 */
    dsChar_t     owner[DSM_MAX_OWNER_LENGTH + 1];      /* object owner          */
    dsStruct64_t sizeEstimate;             /* Size estimate in bytes of the object */
    dsmBool_t    objCompressed;            /* Is object already compressed?     */
    dsUint16_t   objInfoLength;            /* length of object-dependent info   */
    char         *objInfo;                 /* object-dependent info byte buffer */
    dsChar_t     *mcNameP;                 /* mgmnt class name for override     */
    tsmOwType    reserved1;                /* for future use                    */
    tsmOwType    reserved2;                /* for future use                    */
} tsmObjAttr;

#define tsmObjAttrVersion 3
/*--------------------------------------------------------------------------+
| Type definition for mcBindKey returned on dsmBindMC()                     |
+--------------------------------------------------------------------------*/
typedef struct tsmMcBindKey
{
    dsUint16_t  stVersion;                   /* structure version          */
    dsChar_t    mcName[DSM_MAX_MC_NAME_LENGTH + 1];
                                            /* Name of mc bound to object. */
    dsmBool_t   backup_cg_exists;                            /* True/false */
    dsmBool_t   archive_cg_exists;                           /* True/false */
    dsChar_t    backup_copy_dest[DSM_MAX_CG_DEST_LENGTH + 1];
                                                 /* Backup copy dest. name */
    dsChar_t    archive_copy_dest[DSM_MAX_CG_DEST_LENGTH + 1];
                                                     /* Arch copy dest.name */
} tsmMcBindKey;
#define tsmMcBindKeyVersion  1
/*--------------------------------------------------------------------------+
|  Type definition for Mgmt Class queryBuffer on dsmBeginQuery()            |
+--------------------------------------------------------------------------*/
typedef struct tsmQryMCData
{
    dsUint16_t   stVersion;                              /* structure version */
    dsChar_t     *mcName;                       /* Mgmt class name */
```

```
                          /* single name to get one or empty string to get all*/
    dsBool_t     mcDetail;                            /* Want details or not? */
} tsmQryMCData;
#define tsmQryMCDataVersion  1
/*-------------------------------------------------------------------------+
|  Type definition for Archive Copy Group details on Query MC response     |
+-------------------------------------------------------------------------*/
typedef struct tsmArchDetailCG
{
   dsChar_t     cgName[DSM_MAX_CG_NAME_LENGTH + 1];     /* Copy group name */
   dsUint16_t   frequency;                     /* Copy (archive) frequency */
   dsUint16_t   retainVers;                              /* Retain version */
   dsUint8_t    copySer;     /* for copy serialization values, see defines */
   dsUint8_t    copyMode;       /* for copy mode values, see defines above */
   dsChar_t     destName[DSM_MAX_CG_DEST_LENGTH + 1];    /* Copy dest name */
   dsBool_t     bLanFreeDest;         /* Destination has lan free path?    */
   dsBool_t     reserved;                            /* Not currently used */
   dsUint8_t    retainInit;             /* possible values see dsmapitd.h  */
   dsUint16_t   retainMin;              /* if retInit is EVENT num of days */
}tsmArchDetailCG;
/*-------------------------------------------------------------------------+
|  Type definition for Backup Copy Group details on Query MC response      |
+-------------------------------------------------------------------------*/
typedef struct tsmBackupDetailCG
{
   dsChar_t     cgName[DSM_MAX_CG_NAME_LENGTH + 1];       /* Copy group name */
   dsUint16_t   frequency;                               /* Backup frequency */
   dsUint16_t   verDataExst;                         /* Versions data exists */
   dsUint16_t   verDataDltd;                        /* Versions data deleted */
   dsUint16_t   retXtraVers;                        /* Retain extra versions */
   dsUint16_t   retOnlyVers;                         /* Retain only versions */
   dsUint8_t    copySer;       /* for copy serialization values, see defines */
   dsUint8_t    copyMode;         /* for copy mode values, see defines above */
   dsChar_t     destName[DSM_MAX_CG_DEST_LENGTH + 1];      /* Copy dest name */
   dsBool_t     bLanFreeDest;           /* Destination has lan free path?    */
   dsBool_t     reserved;                              /* Not currently used */
}tsmBackupDetailCG;
/*-------------------------------------------------------------------------+
|  Type definition for Query Mgmt Class detail response on dsmGetNextQObj()|
+-------------------------------------------------------------------------*/
typedef struct tsmQryRespMCDetailData
{
    dsUint16_t      stVersion;                       /* structure version */
    dsChar_t        mcName[DSM_MAX_MC_NAME_LENGTH + 1];        /* mc name */
    dsChar_t        mcDesc[DSM_MAX_MC_DESCR_LENGTH + 1]; /*mc description */
    archDetailCG    archDet;                 /* Archive copy group detail */
    backupDetailCG  backupDet;                /* Backup copy group detail */
} tsmQryRespMCDetailData;
#define tsmQryRespMCDetailDataVersion  3
/*-------------------------------------------------------------------------+
| Type definition for Query Mgmt Class summary response on dsmGetNextQObj()|
+-------------------------------------------------------------------------*/
typedef struct tsmQryRespMCData
{
    dsUint16_t   stVersion;                              /* structure version */
    dsChar_t     mcName[DSM_MAX_MC_NAME_LENGTH + 1];               /* mc name */
    dsChar_t     mcDesc[DSM_MAX_MC_DESCR_LENGTH + 1];       /* mc description */
}tsmQryRespMCData;
#define tsmQryRespMCDataVersion  1
/*-------------------------------------------------------------------------+
|  Type definition for Archive queryBuffer on tsmBeginQuery()              |
+-------------------------------------------------------------------------*/
typedef struct tsmQryArchiveData
{
    dsUint16_t   stVersion;                          /* structure version */
    tsmObjName   *objName;                     /* Full dsm name of object */
    dsChar_t     *owner;                                    /* owner name */
```

```
                         /* for maximum date boundaries, see defines above */
    dsmDate      insDateLowerBound;        /* low bound archive insert date */
    dsmDate      insDateUpperBound;         /* hi bound archive insert date */
    dsmDate      expDateLowerBound;            /* low bound expiration date */
    dsmDate      expDateUpperBound;             /* hi bound expiration date */
    dsChar_t     *descr;                 /* archive description */
} tsmQryArchiveData;
#define tsmQryArchiveDataVersion  1
/*--------------------------------------------------------------------------+
| Type definition for Query Archive response on dsmGetNextQObj()             |
+--------------------------------------------------------------------------*/
typedef struct tsmQryRespArchiveData
{
    dsUint16_t      stVersion;                           /* structure version */
    tsmObjName      objName;                      /* Filespace name qualifier */
    dsUint32_t      copyGroup;                           /* copy group number */
    dsChar_t        mcName[DSM_MAX_MC_NAME_LENGTH + 1];            /* mc name */
    dsChar_t        owner[DSM_MAX_OWNER_LENGTH + 1];            /* owner name */
    dsStruct64_t    objId;                                  /* Unique copy id */
    dsStruct64_t    reserved;                       /* backward compatability */
    dsUint8_t       mediaClass;                         /* media access class */
    dsmDate         insDate;                        /* archive insertion date */
    dsmDate         expDate;                    /* expiration date for object */
    dsChar_t        descr[DSM_MAX_DESCR_LENGTH + 1];   /* archive description */
    dsUint16_t      objInfolen;            /* length of object-dependent info */
    dsUint8_t       objInfo[DSM_MAX_OBJINFO_LENGTH];  /*object-dependent info */
    dsUint160_t     restoreOrderExt;                         /* restore order */
    dsStruct64_t    sizeEstimate;             /* size estimate stored by user */
    dsUint8_t       compressType;                         /* Compression flag */
    dsUint8_t       retentionInitiated;   /* object waiting on retention event*/
    dsUint8_t       objHeld; /* object is on "hold" see dsmapitd.h for values */
    dsUint8_t       encryptionType;                     /* type of encryption */
} tsmQryRespArchiveData;
#define tsmQryRespArchiveDataVersion  5
/*--------------------------------------------------------------------------+
|  Type definition for Archive sendBuff parameter on dsmSendObj()           |
+--------------------------------------------------------------------------*/
typedef struct tsmSndArchiveData
{
    dsUint16_t   stVersion;                /* structure version */
    dsChar_t     *descr;                 /* archive description */
} tsmSndArchiveData;
#define tsmSndArchiveDataVersion  1
/*--------------------------------------------------------------------------+
|  Type definition for Backup queryBuffer on dsmBeginQuery()                |
+--------------------------------------------------------------------------*/
typedef struct tsmQryBackupData
{
    dsUint16_t  stVersion;          /* structure version */
    tsmObjName  *objName;           /* full dsm name of object */
    dsChar_t    *owner;             /* owner name */
    dsUint8_t   objState;           /* object state selector */
    dsmDate     pitDate;            /* Date value for point in time restore */
                                    /* for possible values, see defines above */
} tsmQryBackupData;
#define tsmQryBackupDataVersion  2
/*--------------------------------------------------------------------------+
| Type definition for Query Backup response on dsmGetNextQObj()              |
+--------------------------------------------------------------------------*/
typedef struct tsmQryRespBackupData
{
    dsUint16_t      stVersion;                           /* structure version */
    tsmObjName      objName;                       /* full dsm name of object */
    dsUint32_t      copyGroup;                           /* copy group number */
    dsChar_t        mcName[DSM_MAX_MC_NAME_LENGTH + 1];            /* mc name */
    dsChar_t        owner[DSM_MAX_OWNER_LENGTH + 1];            /* owner name */
    dsStruct64_t    objId;                                /* Unique object id */
    dsStruct64_t    reserved;                       /* backward compatability */
    dsUint8_t       mediaClass;                         /* media access class */
    dsUint8_t       objState;                      /* Obj state, active, etc. */
    dsmDate         insDate;                         /* backup insertion date */
    dsmDate         expDate;                    /* expiration date for object */
    dsUint16_t      objInfolen;             /* length of object-dependent info*/
    dsUint8_t       objInfo[DSM_MAX_OBJINFO_LENGTH];  /*object-dependent info */
    dsUint160_t     restoreOrderExt;                         /* restore order */
    dsStruct64_t    sizeEstimate;             /* size estimate stored by user */
    dsStruct64_t    baseObjId;
    dsUint16_t      baseObjInfolen;             /* length of base object-dependent info*/
    dsUint8_t       baseObjInfo[DSM_MAX_OBJINFO_LENGTH];  /* base object-dependent info */
    dsUint160_t     baseRestoreOrder;                         /* restore order */
    dsUint32_t      fsID;
    dsUint8_t       compressType;
    dsmBool_t       isGroupLeader;
    dsmBool_t       isOpenGroup;
    dsUint8_t       reserved1;               /* for future use */
    dsmBool_t       reserved2;               /* for future use */
    dsUint16_t      reserved3;               /* for future use */
    reservedInfo_t  *reserved4;              /* for future use */
    dsUint8_t       encryptionType;                     /* type of encryption */
} tsmQryRespBackupData;
#define tsmQryRespBackupDataVersion  6
/*-------------------------------------------------------------------------+
|  Type definition for Active Backup queryBuffer on dsmBeginQuery()
|
|  Notes:  For the active backup query, only the fs (filespace) and objType
|          fields of objName need be set.  objType can only be set to
|          DSM_OBJ_FILE or DSM_OBJ_DIRECTORY.  DSM_OBJ_ANY_TYPE will not
|          find a match on the query.
+-------------------------------------------------------------------------*/
typedef struct tsmQryABackupData
{
    dsUint16_t      stVersion;                           /* structure version */
    tsmObjName      *objName;                     /* Only fs and objtype used */
} tsmQryABackupData;
#define tsmQryABackupDataVersion  1
/*-------------------------------------------------------------------------+
| Type definition for Query Active Backup response on dsmGetNextQObj()     |
+-------------------------------------------------------------------------*/
typedef struct tsmQryARespBackupData
{
    dsUint16_t  stVersion;                           /* structure version */
    tsmObjName  objName;                       /* full dsm name of object */
    dsUint32_t  copyGroup;                           /* copy group number */
    dsChar_t    mcName[DSM_MAX_MC_NAME_LENGTH + 1];/*management class name*/
    dsChar_t    owner[DSM_MAX_OWNER_LENGTH + 1];            /* owner name */
    dsmDate     insDate;                         /* backup insertion date */
    dsUint16_t  objInfolen;             /* length of object-dependent info*/
    dsUint8_t   objInfo[DSM_MAX_OBJINFO_LENGTH];  /*object-dependent info */
} tsmQryARespBackupData;
#define tsmQryARespBackupDataVersion  1
/*-------------------------------------------------------------------------+
|  Type definition for Backup queryBuffer on dsmBeginQuery()              |
+-------------------------------------------------------------------------*/
typedef struct tsmQryBackupGroups
{
    dsUint16_t   stVersion;         /* structure version */
    dsUint8_t    groupType;
    dsChar_t     *fsName;
    dsChar_t     *owner;
    dsStruct64_t groupLeaderObjId;
    dsUint8_t    objType;
} tsmQryBackupGroups;
#define tsmQryBackupGroupsVersion 1
/*-------------------------------------------------------------------------+
|  Type definition for proxynode queryBuffer on tsmBeginQuery()           |
+-------------------------------------------------------------------------*/
typedef struct tsmQryProxyNodeData
{
   dsUint16_t  stVersion;                   /* structure version */
   dsChar_t    *targetNodeName;             /* target node name      */
}tsmQryProxyNodeData;
#define tsmQryProxyNodeDataVersion  1
/*-------------------------------------------------------------------------+
| Type definition for qryRespProxyNodeData  parameter used on tsmGetNextQObj()|
+-------------------------------------------------------------------------*/
typedef struct tsmQryRespProxyNodeData
{
   dsUint16_t        stVersion ;                           /* structure version */
   dsChar_t          targetNodeName[DSM_MAX_ID_LENGTH+1];  /* target node name  */
   dsChar_t          peerNodeName[DSM_MAX_ID_LENGTH+1];    /* peer node name    */
   dsChar_t          hlAddress[DSM_MAX_ID_LENGTH+1];       /* peer hlAddress    */
   dsChar_t          llAddress[DSM_MAX_ID_LENGTH+1];       /* peer llAddress    */
}tsmQryRespProxyNodeData;
#define tsmQryRespProxyNodeDataVersion  1
/*-------------------------------------------------------------------------+
|  Type definition for WINNT and OS/2 Filespace attributes                 |
+-------------------------------------------------------------------------*/
typedef struct tsmDosFSAttrib
{
    osChar_t     driveLetter ;          /* drive letter for filespace    */
    dsUint16_t   fsInfoLength;          /* fsInfo length used            */
    osChar_t     fsInfo[DSM_MAX_FSINFO_LENGTH];/*caller-determined data  */
} tsmDosFSAttrib ;
/*-------------------------------------------------------------------------+
|  Type definition for UNIX Filespace attributes                           |
+-------------------------------------------------------------------------*/
typedef struct tsmUnixFSAttrib
{
    dsUint16_t   fsInfoLength;          /* fsInfo length used            */
    osChar_t     fsInfo[DSM_MAX_FSINFO_LENGTH];/*caller-determined data  */
} tsmUnixFSAttrib ;
/*-------------------------------------------------------------------------+
|  Type definition for NetWare Filespace attributes                        |
+-------------------------------------------------------------------------*/
typedef tsmUnixFSAttrib tsmNetwareFSAttrib;
/*-------------------------------------------------------------------------+
|  Type definition for Filespace attributes on all Filespace calls         |
+-------------------------------------------------------------------------*/
typedef union
{
    tsmNetwareFSAttrib  netwareFSAttr;
    tsmUnixFSAttrib     unixFSAttr ;
    tsmDosFSAttrib      dosFSAttr ;
} tsmFSAttr ;
/*-------------------------------------------------------------------------+
|  Type definition for fsUpd parameter on dsmUpdateFS()
+-------------------------------------------------------------------------*/
typedef struct   tsmFSUpd
{
    dsUint16_t      stVersion ;            /* structure version              */
    dsChar_t        *fsType ;              /* filespace type                 */
    dsStruct64_t    occupancy ;            /* occupancy estimate             */
    dsStruct64_t    capacity  ;            /* capacity estimate              */
    tsmFSAttr       fsAttr ;               /* platform specific attributes   */
} tsmFSUpd ;
#define tsmFSUpdVersion  1
/*-------------------------------------------------------------------------+
|  Type definition for Filespace queryBuffer on dsmBeginQuery()            |
+-------------------------------------------------------------------------*/
typedef struct tsmQryFSData
{
    dsUint16_t  stVersion;                 /* structure version */
    dsChar_t    *fsName;                   /* File space name */
} tsmQryFSData;
#define tsmQryFSDataVersion  1
/*--------------------------------------------------------------------------+
| Type definition for Query Filespace response on dsmGetNextQObj()          |
+--------------------------------------------------------------------------*/
typedef struct tsmQryRespFSData
{
    dsUint16_t     stVersion;                          /* structure version */
    dsChar_t       fsName[DSM_MAX_FSNAME_LENGTH + 1];     /* Filespace name */
    dsChar_t       fsType[DSM_MAX_FSTYPE_LENGTH + 1] ;     /* Filespace type */
    dsStruct64_t   occupancy;                   /* Occupancy est. in bytes. */
    dsStruct64_t   capacity;                     /* Capacity est. in bytes. */
    tsmFSAttr      fsAttr ;                /* platform specific attributes  */
    dsmDate        backStartDate;           /* start backup date             */
    dsmDate        backCompleteDate;        /* end backup Date               */
    dsmDate        reserved1    ;           /* For future use                */
    dsmBool_t      bIsUnicode;
    dsUint32_t     fsID;
} tsmQryRespFSData;
#define tsmQryRespFSDataVersion 4
/*--------------------------------------------------------------------------+
|  Type definition for regFilespace parameter on dsmRegisterFS()
+--------------------------------------------------------------------------*/
typedef struct tsmRegFSData
{
    dsUint16_t      stVersion;                      /* structure version */
    dsChar_t        *fsName;                        /* Filespace name */
    dsChar_t        *fsType;                        /* Filespace type */
    dsStruct64_t    occupancy;                  /* Occupancy est. in bytes. */
    dsStruct64_t    capacity;                    /* Capacity est. in bytes. */
    tsmFSAttr       fsAttr ;                 /* platform specific attributes */
} tsmRegFSData;
#define tsmRegFSDataVersion  1
/*--------------------------------------------------------------------------+
|  Type definition for session info response on dsmQuerySessionInfo()       |
+--------------------------------------------------------------------------*/
typedef struct
{
  dsUint16_t     stVersion;           /* Structure version                */
     /*-------------------------------------------------------------------*/
     /*           Server information                                      */
     /*-------------------------------------------------------------------*/
  dsChar_t       serverHost[DSM_MAX_SERVERNAME_LENGTH+1];
                                     /* Network host name of DSM server  */
  dsUint16_t     serverPort;            /* Server comm port on host         */
  dsmDate        serverDate;            /* Server's date/time               */
  dsChar_t       serverType[DSM_MAX_SERVERTYPE_LENGTH+1];
                                     /* Server's execution platform      */
  dsUint16_t     serverVer;             /* Server's version number          */
  dsUint16_t     serverRel;             /* Server's release number          */
  dsUint16_t     serverLev;             /* Server's level number            */
  dsUint16_t     serverSubLev;          /* Server's sublevel number         */
     /*-------------------------------------------------------------------*/
     /*           Client Defaults                                         */
     /*-------------------------------------------------------------------*/
  dsChar_t       nodeType[DSM_MAX_PLATFORM_LENGTH+1]; /*node/application type*/
  dsChar_t       fsdelim;               /* File space delimiter             */
  dsChar_t       hldelim;               /* Delimiter betw highlev & lowlev  */
  dsUint8_t      compression;           /* Compression flag                 */
  dsUint8_t      archDel;               /* Archive delete permission        */
  dsUint8_t      backDel;               /* Backup  delete permission        */
  dsUint32_t     maxBytesPerTxn;        /* for future use                   */
  dsUint16_t     maxObjPerTxn;          /* The max objects allowed in a txn */
     /*-------------------------------------------------------------------*/
     /*           Session Information                                     */
     /*-------------------------------------------------------------------*/
  dsChar_t       id[DSM_MAX_ID_LENGTH+1];       /* Sign-in id node name      */
  dsChar_t       owner[DSM_MAX_OWNER_LENGTH+1]; /* Sign-in owner             */
                                    /*   (for multi-user platforms)     */
  dsChar_t       confFile[DSM_PATH_MAX + DSM_NAME_MAX +1];
                                    /* len is platform dep              */
                                    /* dsInit name of appl config file  */
  dsUint8_t      opNoTrace;             /* dsInit option - NoTrace = 1      */
     /*-------------------------------------------------------------------*/
     /*          Policy Data                                             */
     /*-------------------------------------------------------------------*/
  dsChar_t       domainName[DSM_MAX_DOMAIN_LENGTH+1]; /* Domain name         */
  dsChar_t       policySetName[DSM_MAX_PS_NAME_LENGTH+1];
                                    /* Active policy set name           */
  dsmDate        polActDate;            /* Policy set activation date       */
  dsChar_t       dfltMCName[DSM_MAX_MC_NAME_LENGTH+1];/* Default Mgmt Class  */
  dsUint16_t     gpBackRetn;             /* Grace-period backup retention    */
  dsUint16_t     gpArchRetn;             /* Grace-period archive retention   */
  dsChar_t       adsmServerName[DSM_MAX_SERVERNAME_LENGTH+1]; /* adsm server name */
  dsmBool_t      archiveRetentionProtection; /* is server Retention protection enabled */
} tsmApiSessInfo;
#define tsmApiSessInfoVersion  3
/*-------------------------------------------------------------------------+
| Type definition for Query options response on dsmQueryCliOptions()       |
|      and dsmQuerySessOptions()                                           |
+-------------------------------------------------------------------------*/
typedef struct
{
  dsUint16_t  stVersion;
  dsChar_t    dsmiDir[DSM_PATH_MAX + DSM_NAME_MAX +1];
  dsChar_t    dsmiConfig[DSM_PATH_MAX + DSM_NAME_MAX +1];
  dsChar_t    serverName[DSM_MAX_SERVERNAME_LENGTH+1];
  dsInt16_t   commMethod;
  dsChar_t    serverAddress[DSM_MAX_SERVER_ADDRESS];
  dsChar_t    nodeName[DSM_MAX_NODE_LENGTH+1];
  dsmBool_t   compression;
  dsmBool_t   compressalways;
  dsmBool_t   passwordAccess;
  }tsmOptStruct ;
#define tsmOptStructVersion  1
/*-------------------------------------------------------------------------+
| Type definition for qryRespAccessData  parameter used on dsmQueryAccess()|
+-------------------------------------------------------------------------*/
typedef struct
{
    dsUint16_t        stVersion ;                   /* structure version      */
    dsChar_t          node[DSM_MAX_ID_LENGTH+1];     /* node name              */
    dsChar_t          owner[DSM_MAX_OWNER_LENGTH+1]; /* owner                  */
    tsmObjName        objName ;                      /* object name            */
    dsmAccessType     accessType;                    /* archive or backup     */
    dsUint32_t        ruleNumber ;                   /* Access rule id         */
}tsmQryRespAccessData;
#define tsmQryRespAccessDataVersion  1
/*-------------------------------------------------------------------------+
|  Type definition for envSetUp parameter on dsmSetUp()
+-------------------------------------------------------------------------*/
typedef struct tsmEnvSetUp
{
    dsUint16_t      stVersion;                      /* structure version */
    dsChar_t        dsmiDir[DSM_PATH_MAX + DSM_NAME_MAX +1];
    dsChar_t        dsmiConfig[DSM_PATH_MAX + DSM_NAME_MAX +1];
    dsChar_t        dsmiLog[DSM_PATH_MAX + DSM_NAME_MAX +1];
    char            **argv;  /* for executables name argv[0] */
    dsChar_t        logName[DSM_NAME_MAX +1];
    dsmBool_t       reserved1;        /* for future use */
    dsmBool_t       reserved2;                  /* for future use */
} tsmEnvSetUp;
```

```
#define tsmEnvSetUpVersion  4
/*-------------------------------------------------------------------------+
|  Type definition for dsmInitExIn_t
+-------------------------------------------------------------------------*/
typedef struct tsmInitExIn_t
{
    dsUint16_t         stVersion;                      /* structure version */
    tsmApiVersionEx    *apiVersionExP;
    dsChar_t           *clientNodeNameP;
    dsChar_t           *clientOwnerNameP;
    dsChar_t           *clientPasswordP;
    dsChar_t           *userNameP;
    dsChar_t           *userPasswordP;
    dsChar_t           *applicationTypeP;
    dsChar_t           *configfile;
    dsChar_t           *options;
    dsChar_t           dirDelimiter;
    dsmBool_t          useUnicode;
    dsmBool_t          bCrossPlatform;
    dsmBool_t          bService;
    dsmBool_t          bEncryptKeyEnabled;
    dsChar_t           *encryptionPasswordP;
    dsmBool_t          useTsmBuffers;
    dsUint8_t          numTsmBuffers;
} tsmInitExIn_t;
#define tsmInitExInVersion 4
/*-------------------------------------------------------------------------+
|  Type definition for dsmInitExOut_t
+-------------------------------------------------------------------------*/
typedef struct tsmInitExOut_t
{
    dsUint16_t         stVersion;                      /* structure version */
    dsInt16_t          userNameAuthorities;
    dsInt16_t          infoRC;        /* error return code if encountered */
                                      /* adsm server name                 */
    dsChar_t           adsmServerName[DSM_MAX_SERVERNAME_LENGTH+1];
    dsUint16_t         serverVer;     /* Server's version number          */
    dsUint16_t         serverRel;     /* Server's release number          */
    dsUint16_t         serverLev;     /* Server's level number            */
    dsUint16_t         serverSubLev;  /* Server's sublevel number         */
} tsmInitExOut_t;
#define tsmInitExOutVersion 2
/*-------------------------------------------------------------------------+
|  Type definition for dsmLogExIn_t
+-------------------------------------------------------------------------*/
typedef struct tsmLogExIn_t
{
    dsUint16_t         stVersion;  /* structure version */
    dsmLogSeverity     severity;
    dsChar_t           appMsgID[8];
    dsmLogType         logType;    /* log type : local, server, both */
    dsChar_t           *message;   /* text of message to be logged */
    dsChar_t           appName[DSM_MAX_PLATFORM_LENGTH];
    dsChar_t           osPlatform[DSM_MAX_PLATFORM_LENGTH];
    dsChar_t           appVersion[DSM_MAX_PLATFORM_LENGTH];
} tsmLogExIn_t;
#define tsmLogExInVersion 2
/*-------------------------------------------------------------------------+
|  Type definition for dsmlogExOut_t
+-------------------------------------------------------------------------*/
typedef struct tsmLogExOut_t
{
    dsUint16_t         stVersion;  /* structure version */
} tsmLogExOut_t;
#define tsmLogExOutVersion 1
/*-------------------------------------------------------------------------+
|  Type definition for dsmRenameIn_t
```

```
+-------------------------------------------------------------------------*/
typedef struct tsmRenameIn_t
{
    dsUint16_t       stVersion;                     /* structure version  */
    dsUint32_t       tsmHandle;                     /* handle for session */
    dsUint8_t        repository;                    /* Backup or Archive  */
    tsmObjName       *objNameP ;                    /* object name */
    dsChar_t         newHl[DSM_MAX_HL_LENGTH + 1];  /* new High level name */
    dsChar_t         newLl[DSM_MAX_LL_LENGTH + 1];  /* new Low level name */
    dsmBool_t        merge;                        /* merge into existing name*/
    ObjID            objId;                         /* objId for Archive */
} tsmRenameIn_t;
#define tsmRenameInVersion 1
/*-------------------------------------------------------------------------+
|  Type definition for dsmRenameOut_t
+-------------------------------------------------------------------------*/
typedef struct tsmRenameOut_t
{
    dsUint16_t         stVersion;                      /* structure version */
} tsmRenameOut_t;
#define tsmRenameOutVersion 1
/*-------------------------------------------------------------------------+
|  Type definition for tsmEndSendObjExIn_t
+-------------------------------------------------------------------------*/
typedef struct tsmEndSendObjExIn_t
{
    dsUint16_t       stVersion;                     /* structure version  */
    dsUint32_t       tsmHandle;                     /* handle for session */
} tsmEndSendObjExIn_t;
#define tsmEndSendObjExInVersion 1
/*-------------------------------------------------------------------------+
|  Type definition for dsmEndSendObjExOut_t
+-------------------------------------------------------------------------*/
typedef struct tsmEndSendObjExOut_t
{
   dsUint16_t         stVersion;           /* structure version */
   dsStruct64_t       totalBytesSent;      /* total bytes read from app */
   dsmBool_t          objCompressed;       /* was object compressed */
   dsStruct64_t       totalCompressSize;   /* total size after compress */
   dsStruct64_t       totalLFBytesSent;    /* total bytes sent Lan Free */
   dsUint8_t          encryptionType;      /* type of encryption used   */
}tsmEndSendObjExOut_t;
#define tsmEndSendObjExOutVersion 2
/*-------------------------------------------------------------------------+
|  Type definition for tsmGroupHandlerIn_t
+-------------------------------------------------------------------------*/
typedef struct tsmGroupHandlerIn_t
{
    dsUint16_t       stVersion;       /* structure version                       */
    dsUint32_t       tsmHandle;       /* handle for session                      */
    dsUint8_t        groupType;       /* Type of group                           */
    dsUint8_t        actionType;      /* Type of group operation                 */
    dsUint8_t        memberType;      /* Type of member: Leader or member        */
    dsStruct64_t     leaderObjId;     /* OBJID of the groupleader                */
    dsChar_t         *uniqueGroupTagP; /* Unique group identifier                */
    tsmObjName       *objNameP ;      /* group leader object name                */
    dsmGetList       memberObjList;   /* list of objects to remove, assign       */
} tsmGroupHandlerIn_t;
#define tsmGroupHandlerInVersion 1
/*-------------------------------------------------------------------------+
|  Type definition for tsmGroupHandlerExOut_t
+-------------------------------------------------------------------------*/
typedef struct tsmGroupHandlerOut_t
{
    dsUint16_t         stVersion;                      /* structure version */
} tsmGroupHandlerOut_t;
#define tsmGroupHandlerOutVersion 1
```

```
/*-------------------------------------------------------------------------+
|  Type definition for tsmEndTxnExIn_t
+-------------------------------------------------------------------------*/
typedef struct tsmEndTxnExIn_t
{
    dsUint16_t       stVersion;                    /* structure version  */
    dsUint32_t       tsmHandle;                    /* handle for session */
    dsUint8_t        vote;
} tsmEndTxnExIn_t;
#define tsmEndTxnExInVersion 1
/*-------------------------------------------------------------------------+
|  Type definition for tsmEndTxnExOut_t
+-------------------------------------------------------------------------*/
typedef struct tsmEndTxnExOut_t
{
    dsUint16_t         stVersion;              /* structure version              */
    dsUint16_t         reason;                 /* reason code                    */
    dsStruct64_t       groupLeaderObjId;       /* groupLeader obj id returned on */
                                               /* DSM_ACTION_OPEN                */
    dsUint8_t          reserved1;              /* future use                     */
    dsUint16_t         reserved2;              /* future use                     */
} tsmEndTxnExOut_t;
#define tsmEndTxnExOutVersion 1
/*-------------------------------------------------------------------------+
|  Type definition for tsmEndGetDataExIn_t
+-------------------------------------------------------------------------*/
typedef struct tsmEndGetDataExIn_t
{
   dsUint16_t     stVersion;   /* structure version  */
   dsUint32_t     tsmHandle;   /* handle for session */
}tsmEndGetDataExIn_t;
#define tsmEndGetDataExInVersion 1
/*-------------------------------------------------------------------------+
|  Type definition for tsmEndGetDataExOut_t
+-------------------------------------------------------------------------*/
typedef struct tsmEndGetDataExOut_t
{
   dsUint16_t     stVersion;         /* structure version             */
   dsUint16_t     reason;            /* reason code                   */
   dsStruct64_t   totalLFBytesRecv;  /* total lan free bytes recieved */
}tsmEndGetDataExOut_t;
#define tsmEndGetDataExOutVersion 1
/*-------------------------------------------------------------------------+
|  Type definition for  on tsmRetentionEvent()                             |
+-------------------------------------------------------------------------*/
typedef struct tsmRetentionEventIn_t
{
   dsUint16_t         stVersion;                    /* structure version  */
   dsUint32_t         tsmHandle;                    /* session Handle     */
   dsmEventType_t     eventType;                    /* Event type         */
   dsmObjList_t       objList;                      /* object ID          */
}tsmRetentionEventIn_t;
#define tsmRetentionEventInVersion 1
/*-------------------------------------------------------------------------+
|  Type definition for  on tsmRetentionEvent()               |
+-------------------------------------------------------------------------*/
typedef struct tsmRetentionEventOut_t
{
   dsUint16_t       stVersion ;                     /* structure version      */
}tsmRetentionEventOut_t;
#define tsmRetentionEventOutVersion 1
/*-------------------------------------------------------------------------+
|  Type definition for tsmUpdateObjExIn_t
+-------------------------------------------------------------------------*/
typedef struct tsmUpdateObjExIn_t
{
   dsUint16_t      stVersion;            /* structure version   */
   dsUint32_t      tsmHandle;            /* session Handle      */
   tsmSendType     sendType;             /* send type back/arch */
   dsChar_t        *descrP;              /* archive description */
   tsmObjName      *objNameP;            /* objName             */
   tsmObjAttr      *objAttrPtr;          /* attribute           */
   dsUint32_t      objUpdAct;            /* update action       */
   ObjID           archObjId;            /* objId for archive   */
}tsmUpdateObjExIn_t;
#define tsmUpdateObjExInVersion 1
/*-------------------------------------------------------------------------+
|  Type definition for tsmUpdateObjExOut_t
+-------------------------------------------------------------------------*/
typedef struct tsmUpdateObjExOut_t
{
    dsUint16_t         stVersion;        /* structure version */
}tsmUpdateObjExOut_t;
#define tsmUpdateObjExOutVersion 1
#if _OPSYS_TYPE == DS_WINNT
#pragma pack()
#endif
#ifdef _MAC
#pragma options align = reset
#endif
#endif /* _H_TSMAPITD */
```

```
/***********************************************************************
* Tivoli Storage Manager                                               *
* API Client Component                                                 *
*                                                                      *
* (C) Copyright IBM Corporation 1993,2008                              *
***********************************************************************/
/**************************************************************************
* Header File Name: dsmapips.h
*
* Environment:      *********************************************
*                   ** This is a platform-specific source file **
*                   ** versioned for Windows NT                **
*
*                   *********************************************
*
* Design Notes:     This file includes platform dependent definitions
*
* Descriptive-name: Definitions for Tivoli Storage Manager typedefs and LINKAGE
*-------------------------------------------------------------------------*/
#ifndef _H_DSMAPIPS
#define _H_DSMAPIPS
#ifndef _WIN64
#pragma pack(1)
#endif
/*<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>*/
/*                     T Y P E D E F S                                   */
/*<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>*/
/* new typedef file for Version 3   */
#if !defined(DSMAPILIB) || defined (XOPEN_BUILD)
/* support for linkage */
#include <windows.h>
#define DSMLINKAGE WINAPI
#define DS_WINNT    22
#define _OPSYS_TYPE DS_WINNT
   typedef signed   char  dsInt8_t;
   typedef unsigned char  dsUint8_t;
   typedef signed   short dsInt16_t;
   typedef unsigned short dsUint16_t;
   typedef signed   long  dsInt32_t;
   typedef unsigned long  dsUint32_t;
/*=== Character and string types ===*/
#ifdef UNICODE
  typedef wchar_t dsChar_t;
  #define dsTEXT(x)       L##x
#else
  typedef char dsChar_t;
  #define dsTEXT(x)       x
#endif  /* !UNICODE */
/*=== Common typedefs and defines derived from dsChar_t ===*/
typedef dsChar_t           *dsString_t;
  /* added for the extended restore order */
   typedef struct
   {
     dsUint32_t top;
     dsUint32_t hi_hi;
     dsUint32_t hi_lo;
     dsUint32_t lo_hi;
     dsUint32_t lo_lo;
   } dsUint160_t ;
#if defined(_LONG_LONG)
   typedef __int64              dsInt64_t;
   typedef unsigned __int64     dsUint64_t;
   /*=== A "true" unsigned 64-bit integer ===*/
   typedef __int64              dsLongLong_t;
#else
typedef struct tagUINT64_t
   {
      dsUint32_t hi;        /* Most significant 32 bits. */
      dsUint32_t lo;        /* Least significant 32 bits. */
   } dsUint64_t;
#endif
/*--------------------------------------------------------------------------+
| Type definition for bool_t                                                |
+--------------------------------------------------------------------------*/
/*
 *  Had to create a Boolean type that didn't clash with any other predefined
 *  version in any operating system or windowing system.
 */
typedef enum
{
   dsmFalse = 0x00,
   dsmTrue  = 0x01
}dsmBool_t ;
/*===  for backward compatability  ===*/
#define uint8    dsUint8_t
#define int8     dsInt8_t
#define uint16   dsUint16_t
#define int16    dsInt16_t
#define uint32   dsUint32_t
#define int32    dsInt32_t
#define uint64   dsStruct64_t
#define bool_t   dsBool_t
#define dsBool_t dsmBool_t
#define bTrue    dsmTrue
#define bFalse   dsmFalse
typedef struct
{
   dsUint32_t hi;        /* Most significant 32 bits. */
   dsUint32_t lo;        /* Least significant 32 bits. */
}dsStruct64_t ;
#endif /* DSMAPILIB */
#ifndef _WIN64
#pragma pack()
#endif
#endif  /* _H_DSMAPIPS */</windows.h>
```

```
/************************************************************************
* Tivoli Storage Manager                                                *
* Common Source Component                                               *
*                                                                       *
* (C) Copyright IBM Corporation 1993,2008                               *
```

```
*************************************************************************/
/*************************************************************************
* Header File Name: release.h
*
* Environment:     ***********************************************
*                  ** This is a platform-independent source file **
*                  ***********************************************
*
* Design Notes:    This file contains the common information about
*                  the actual version.release.level.sublevel
*
* Descriptive-name: Definitions for Tivoli Storage manager version
*
* Note: This file should contain no LOG or CMVC information. It is
*       shipped with the API code.
*
*-----------------------------------------------------------------------*/
#ifndef _H_RELEASE
#define _H_RELEASE
#define COMMON_VERSION         6
#define COMMON_RELEASE         1
#define COMMON_LEVEL           0
#define COMMON_SUBLEVEL        00
#define COMMON_DRIVER  dsTEXT("")
#define COMMON_VERSIONTXT "6.1.0.00"
#define SHIPYEARTXT  "2008"
#define SHIPYEARTXTW dsTEXT("2008")
#define TSMPRODTXT  "IBM Tivoli Storage Manager"
/*======================================================================
   The following string definitions are used for VERSION information
   and should not be converted to dsTEXT or osTEXT.  They are used
   only at link time.

   These are also used when the Jar file is built on Unix.  See the
   the perl script tools/unx/mzbuild/createReleaseJava
======================================================================*/
#define COMMON_VERSION_STR    "6"
#define COMMON_RELEASE_STR    "1"
#define COMMON_LEVEL_STR      "0"
#define COMMON_SUBLEVEL_STR   "00"
#define COMMON_DRIVER_STR     ""
/*=== product names definitions ===*/
#define COMMON_NAME_DFDSM      1
#define COMMON_NAME_ADSM       2
#define COMMON_NAME_TSM        3
#define COMMON_NAME_ITSM       4
#define COMMON_NAME            COMMON_NAME_ITSM
/*======================================================================
   Internal version, release, and level (build) version.  This
   should be unique for every version+release+ptf of a product.
   This information is recorded in the file attributes and data
   stream for diagnostic purposes.
   NOTE: DO NOT MODIFY THESE VALUES. YOU CAN ONLY ADD NEW ENTRIES ONLY!
======================================================================*/
#define COMMON_BUILD_TSM_510   1
#define COMMON_BUILD_TSM_511   2
#define COMMON_BUILD_TSM_515   3
#define COMMON_BUILD_TSM_516   4
#define COMMON_BUILD_TSM_520   5
#define COMMON_BUILD_TSM_522   6
#define COMMON_BUILD_TSM_517   7
#define COMMON_BUILD_TSM_523   8
#define COMMON_BUILD_TSM_530   9
#define COMMON_BUILD_TSM_524   10
#define COMMON_BUILD_TSM_532   11
#define COMMON_BUILD_TSM_533   12
#define COMMON_BUILD_TSM_525   13
```

```
#define COMMON_BUILD_TSM_534   14
#define COMMON_BUILD_TSM_540   15
#define COMMON_BUILD_TSM_535   16
#define COMMON_BUILD_TSM_541   17
#define COMMON_BUILD_TSM_550   18
#define COMMON_BUILD_TSM_542   19
#define COMMON_BUILD_TSM_551   20
#define COMMON_BUILD_TSM_610   21
#define COMMON_BUILD           COMMON_BUILD_TSM_610
#endif /* _H_RELEASE */
```

Appendix D. API function definitions source file

This appendix contains the dsmapifp.h header file, so you can see the function definitions for the API.

Note: DSMLINKAGE is defined differently for each operating system. See the definitions in the dsmapips.h file for your specific operating system.

```
/*********************************************************************
* Tivoli Storage Manager                                             *
* API Client Component                                               *
*                                                                    *
* (C) Copyright IBM Corporation 1993,2008                            *
*********************************************************************/
/************************************************************************/
/* Header File Name: dsmapifp.h                                         */
/*                                                                      */
/* Descriptive-name: Tivoli Storage Manager API function prototypes     */
/************************************************************************/
#ifndef _H_DSMAPIFP
#define _H_DSMAPIFP
#if defined(__cplusplus)
extern "C" {
#endif
#ifdef DYNALOAD_DSMAPI
/* function will be dynamically loaded */
#include "dsmapidl.h"
#else
/* functions will be implicitly loaded from library */
/*======================================================================*/
/*             P U B L I C   F U N C T I O N S                          */
/*======================================================================*/
extern dsInt16_t DSMLINKAGE  dsmBeginGetData(
       dsUint32_t           dsmHandle,
       dsBool_t             mountWait,
       dsmGetType           getType,
       dsmGetList           *dsmGetObjListP
);
extern dsInt16_t DSMLINKAGE  dsmBeginQuery(
       dsUint32_t           dsmHandle,
       dsmQueryType         queryType,
       dsmQueryBuff         *queryBuffer
);
extern dsInt16_t DSMLINKAGE  dsmBeginTxn(
       dsUint32_t           dsmHandle
);
extern dsInt16_t DSMLINKAGE  dsmBindMC(
       dsUint32_t           dsmHandle,
       dsmObjName           *objNameP,
       dsmSendType          sendType,
       mcBindKey            *mcBindKeyP
);
extern dsInt16_t DSMLINKAGE  dsmChangePW(
       dsUint32_t           dsmHandle,
       char                 *oldPW,
       char                 *newPW
);
extern dsInt16_t DSMLINKAGE dsmCleanUp(
       dsBool_t             mtFlag
);
extern dsInt16_t DSMLINKAGE  dsmDeleteAccess(
       dsUint32_t           dsmHandle,
       dsUint32_t           ruleNum
```

```
);
extern dsInt16_t DSMLINKAGE  dsmDeleteObj(
       dsUint32_t            dsmHandle,
       dsmDelType            delType,
       dsmDelInfo            delInfo
);
extern dsInt16_t DSMLINKAGE  dsmDeleteFS(
       dsUint32_t            dsmHandle,
       char                  *fsName,
       dsUint8_t             repository
);
extern dsInt16_t DSMLINKAGE  dsmEndGetData(
       dsUint32_t            dsmHandle
);
extern dsInt16_t DSMLINKAGE  dsmEndGetDataEx(
       dsmEndGetDataExIn_t  *dsmEndGetDataExInP,
       dsmEndGetDataExOut_t *dsmEndGetDataExOutP
);
extern dsInt16_t DSMLINKAGE  dsmEndGetObj(
       dsUint32_t            dsmHandle
);
extern dsInt16_t DSMLINKAGE  dsmEndQuery(
       dsUint32_t            dsmHandle
);
extern dsInt16_t DSMLINKAGE  dsmEndSendObj(
       dsUint32_t            dsmHandle
);
extern dsInt16_t DSMLINKAGE  dsmEndSendObjEx(
       dsmEndSendObjExIn_t   *dsmEndSendObjExInP,
       dsmEndSendObjExOut_t  *dsmEndSendObjExOutP
);
extern dsInt16_t DSMLINKAGE  dsmEndTxnEx(
       dsmEndTxnExIn_t      *dsmEndTxnExInP,
       dsmEndTxnExOut_t     *dsmEndTxnExOutP
);
extern dsInt16_t DSMLINKAGE  dsmEndTxn(
       dsUint32_t            dsmHandle,
       dsUint8_t             vote,
       dsUint16_t            *reason
);
extern dsInt16_t DSMLINKAGE  dsmGetData(
       dsUint32_t            dsmHandle,
       DataBlk               *dataBlkPtr
);
extern dsInt16_t DSMLINKAGE  dsmGetBufferData(
       getBufferDataIn_t     *dsmGetBufferDataInP,
       getBufferDataOut_t    *dsmGetBufferDataOutP
);
extern dsInt16_t DSMLINKAGE  dsmGetNextQObj(
       dsUint32_t            dsmHandle,
       DataBlk               *dataBlkPtr
) ;
extern dsInt16_t DSMLINKAGE  dsmGetObj(
       dsUint32_t            dsmHandle,
       ObjID                 *objIdP,
       DataBlk               *dataBlkPtr
);
extern dsInt16_t DSMLINKAGE  dsmGroupHandler(
       dsmGroupHandlerIn_t   *dsmGroupHandlerInP,
       dsmGroupHandlerOut_t  *dsmGroupHandlerOutP
);
extern dsInt16_t DSMLINKAGE  dsmInit(
       dsUint32_t            *dsmHandle,
       dsmApiVersion         *dsmApiVersionP,
       char                  *clientNodeNameP,
       char                  *clientOwnerNameP,
       char                  *clientPasswordP,
       char                 *applicationType,
       char                 *configfile,
       char                 *options
);
extern dsInt16_t DSMLINKAGE  dsmInitEx(
       dsUint32_t           *dsmHandleP,
       dsmInitExIn_t        *dsmInitExInP,
       dsmInitExOut_t       *dsmInitExOutP
);
extern dsInt16_t DSMLINKAGE  dsmLogEvent(
       dsUint32_t            dsmHandle,
       logInfo              *lopInfoP
);
extern dsInt16_t DSMLINKAGE  dsmLogEventEx(
       dsUint32_t            dsmHandle,
       dsmLogExIn_t         *dsmLogExInP,
       dsmLogExOut_t        *dsmLogExOutP
);
extern dsInt16_t DSMLINKAGE  dsmQueryAccess(
       dsUint32_t            dsmHandle,
       qryRespAccessData    **accessListP,
       dsUint16_t           *numberOfRules
);
extern void DSMLINKAGE       dsmQueryApiVersion(
       dsmApiVersion        *apiVersionP
);
extern void DSMLINKAGE       dsmQueryApiVersionEx(
       dsmApiVersionEx      *apiVersionP
);
extern dsInt16_t DSMLINKAGE  dsmQueryCliOptions(
       optStruct            *optstructP
);
extern dsInt16_t DSMLINKAGE  dsmQuerySessInfo(
       dsUint32_t            dsmHandle,
       ApiSessInfo          *SessInfoP
);
extern dsInt16_t DSMLINKAGE  dsmQuerySessOptions(
       dsUint32_t            dsmHandle,
       optStruct            *optstructP
);
extern dsInt16_t DSMLINKAGE  dsmRCMsg(
       dsUint32_t            dsmHandle,
       dsInt16_t             dsmRC,
       char                 *msg
);
extern dsInt16_t DSMLINKAGE  dsmRegisterFS(
       dsUint32_t            dsmHandle,
       regFSData            *regFilespaceP
);
extern dsInt16_t DSMLINKAGE   dsmReleaseBuffer(
       releaseBufferIn_t      *dsmReleaseBufferInP,
       releaseBufferOut_t     *dsmReleaseBufferOutP
);
extern dsInt16_t DSMLINKAGE   dsmRenameObj(
       dsmRenameIn_t          *dsmRenameInP,
       dsmRenameOut_t         *dsmRenameOutP
);
extern dsInt16_t DSMLINKAGE  dsmRequestBuffer(
       requestBufferIn_t     *dsmRequestBufferInP,
       requestBufferOut_t    *dsmRequestBufferOutP
);
extern dsInt16_t DSMLINKAGE   dsmRetentionEvent(
       dsmRetentionEventIn_t  *dsmRetentionEventInP,
       dsmRetentionEventOut_t *dsmRetentionEventOutP
);
extern dsInt16_t DSMLINKAGE   dsmSendBufferData(
       sendBufferDataIn_t     *dsmSendBufferDataInP,
       sendBufferDataOut_t    *dsmSendBufferDataOutP
);
extern dsInt16_t DSMLINKAGE   dsmSendData(
       dsUint32_t             dsmHandle,
       DataBlk                *dataBlkPtr
) ;
extern dsInt16_t DSMLINKAGE   dsmSendObj(
       dsUint32_t             dsmHandle,
       dsmSendType            sendType,
       void                   *sendBuff,
       dsmObjName             *objNameP,
       ObjAttr                *objAttrPtr,
       DataBlk                *dataBlkPtr
);
extern dsInt16_t DSMLINKAGE   dsmSetAccess(
       dsUint32_t             dsmHandle,
       dsmAccessType          accessType,
       dsmObjName             *objNameP,
       char                   *node,
       char                   *owner
);
extern dsInt16_t DSMLINKAGE   dsmSetUp(
       dsBool_t               mtFlag,
       envSetUp               *envSetUpP
);
extern dsInt16_t DSMLINKAGE   dsmTerminate(
       dsUint32_t             dsmHandle
);
extern dsInt16_t DSMLINKAGE   dsmUpdateFS(
       dsUint32_t             dsmHandle,
       char                   *fs,
       dsmFSUpd               *fsUpdP,
       dsUint32_t             fsUpdAct
);
extern dsInt16_t DSMLINKAGE   dsmUpdateObj(
       dsUint32_t             dsmHandle,
       dsmSendType            sendType,
       void                   *sendBuff,
       dsmObjName             *objNameP,
       ObjAttr                *objAttrPtr,
       dsUint32_t             objUpdAct
);
extern dsInt16_t DSMLINKAGE   dsmUpdateObjEx(
       dsmUpdateObjExIn_t     *dsmUpdateObjExInP,
       dsmUpdateObjExOut_t    *dsmUpdateObjExOutP
);
#endif /* ifdef DYNALOAD */
#if defined(__cplusplus)
   }
#endif
#endif /* _H_DSMAPIFP */
```

This section contains the function definitions for the API. It is a copy of the tsmapifp.h header file.

Note: DSMLINKAGE is defined differently for each operating system. See the definitions in the tsmapips.h file for your specific operating system.

```
/************************************************************************
* Tivoli Storage Manager                                *
* API Client Component                                  *
*                                                       *
* (C) Copyright IBM Corporation 1993,2008               *
***********************************************************************/
/*************************************************************************/
/* Header File Name: tsmapifp.h                                 */
/*                                                              */
/* Descriptive-name: Tivoli Storage Manager API function prototypes       */
/**************************************************************************/
#ifndef _H_TSMAPIFP
#define _H_TSMAPIFP
#if defined(__cplusplus)
extern "C" {
#endif
#ifdef DYNALOAD_DSMAPI
/* function will be dynamically loaded */
#include "dsmapidl.h"
#else
/* functions will be implicitly loaded from library */
/*========================================================================*/
/*P U B L I C   F U N C T I O N S                          */
/*========================================================================*/
typedef void tsmQueryBuff;
extern dsInt16_t DSMLINKAGE  tsmBeginGetData(
       dsUint32_t            tsmHandle,
       dsBool_t              mountWait,
       tsmGetType            getType,
       dsmGetList            *dsmGetObjListP
);
extern dsInt16_t DSMLINKAGE  tsmBeginQuery(
       dsUint32_t            tsmHandle,
       tsmQueryType          queryType,
       tsmQueryBuff          *queryBuffer
);
extern dsInt16_t DSMLINKAGE  tsmBeginTxn(
       dsUint32_t            tsmHandle
);
extern dsInt16_t DSMLINKAGE  tsmBindMC(
       dsUint32_t            tsmHandle,
       tsmObjName            *objNameP,
       tsmSendType           sendType,
       tsmMcBindKey          *mcBindKeyP
);
extern dsInt16_t DSMLINKAGE  tsmChangePW(
       dsUint32_t            tsmHandle,
       dsChar_t              *oldPW,
       dsChar_t              *newPW
);
extern dsInt16_t DSMLINKAGE tsmCleanUp(
       dsBool_t              mtFlag
);
extern dsInt16_t DSMLINKAGE  tsmDeleteAccess(
       dsUint32_t            tsmHandle,
       dsUint32_t            ruleNum
);
extern dsInt16_t DSMLINKAGE  tsmDeleteObj(
       dsUint32_t            tsmHandle,
       tsmDelType            delType,
       tsmDelInfo            delInfo
);
extern dsInt16_t DSMLINKAGE  tsmDeleteFS(
       dsUint32_t            tsmHandle,
       dsChar_t              *fsName,
       dsUint8_t             repository
);
extern dsInt16_t DSMLINKAGE  tsmEndGetData(
       dsUint32_t            tsmHandle
);
extern dsInt16_t DSMLINKAGE  tsmEndGetDataEx(
       tsmEndGetDataExIn_t      *tsmEndGetDataExInP,
       tsmEndGetDataExOut_t     *tsmEndGetDataExOutP
);
extern dsInt16_t DSMLINKAGE  tsmEndGetObj(
       dsUint32_t            tsmHandle
```

```
);
extern dsInt16_t DSMLINKAGE  tsmEndQuery(
       dsUint32_t            tsmHandle
);
extern dsInt16_t DSMLINKAGE  tsmEndSendObj(
       dsUint32_t            tsmHandle
);
extern dsInt16_t DSMLINKAGE  tsmEndSendObjEx(
       tsmEndSendObjExIn_t      *tsmEndSendObjExInP,
       tsmEndSendObjExOut_t     *tsmEndSendObjExOutP
);
extern dsInt16_t DSMLINKAGE  tsmEndTxn(
       dsUint32_t            tsmHandle,
       dsUint8_t             vote,
       dsUint16_t            *reason
);
extern dsInt16_t DSMLINKAGE  tsmEndTxnEx(
       tsmEndTxnExIn_t      *tsmEndTxnExInP,
       tsmEndTxnExOut_t     *tsmEndTxnExOutP
);
extern dsInt16_t DSMLINKAGE  tsmGetData(
       dsUint32_t            tsmHandle,
       DataBlk*dataBlkPtr
);
extern dsInt16_t DSMLINKAGE  tsmGetBufferData(
       getBufferDataIn_t     *tsmGetBufferDataInP,
       getBufferDataOut_t    *tsmGetBufferDataOutP
);
extern dsInt16_t DSMLINKAGE  tsmGetNextQObj(
       dsUint32_t            tsmHandle,
       DataBlk*dataBlkPtr
) ;
extern dsInt16_t DSMLINKAGE  tsmGetObj(
       dsUint32_t            tsmHandle,
       ObjID                 *objIdP,
       DataBlk               *dataBlkPtr
);
extern dsInt16_t DSMLINKAGE  tsmGroupHandler(
       tsmGroupHandlerIn_t   *tsmGroupHandlerInP,
       tsmGroupHandlerOut_t  *tsmGroupHandlerOutP
);
extern dsInt16_t DSMLINKAGE  tsmInitEx(
       dsUint32_t            *tsmHandleP,
       tsmInitExIn_t         *tsmInitExInP,
       tsmInitExOut_t        *tsmInitExOutP
);
extern dsInt16_t DSMLINKAGE  tsmLogEventEx(
       dsUint32_t            tsmHandle,
       tsmLogExIn_t          *tsmLogExInP,
       tsmLogExOut_t         *tsmLogExOutP
);
extern dsInt16_t DSMLINKAGE  tsmQueryAccess(
       dsUint32_t            tsmHandle,
       tsmQryRespAccessData  **accessListP,
       dsUint16_t            *numberOfRules
);
extern void DSMLINKAGE       tsmQueryApiVersionEx(
       tsmApiVersionEx       *apiVersionP
);
extern dsInt16_t DSMLINKAGE  tsmQueryCliOptions(
       tsmOptStruct          *optstructP
);
extern dsInt16_t DSMLINKAGE  tsmQuerySessInfo(
       dsUint32_t            tsmHandle,
       tsmApiSessInfo        *SessInfoP
);
extern dsInt16_t DSMLINKAGE  tsmQuerySessOptions(
       dsUint32_t            tsmHandle,
       tsmOptStruct          *optstructP
);
extern dsInt16_t DSMLINKAGE  tsmRCMsg(
       dsUint32_t            tsmHandle,
       dsInt16_t             tsmRC,
       dsChar_t              *msg
);
extern dsInt16_t DSMLINKAGE  tsmRegisterFS(
       dsUint32_t            tsmHandle,
       tsmRegFSData          *regFilespaceP
);
extern dsInt16_t DSMLINKAGE  tsmReleaseBuffer(
       releaseBufferIn_t     *tsmReleaseBufferInP,
       releaseBufferOut_t    *tsmReleaseBufferOutP
);
extern dsInt16_t DSMLINKAGE  tsmRenameObj(
       tsmRenameIn_t         *tsmRenameInP,
       tsmRenameOut_t        *tsmRenameOutP
);
extern dsInt16_t DSMLINKAGE  tsmRequestBuffer(
       requestBufferIn_t     *tsmRequestBufferInP,
       requestBufferOut_t    *tsmRequestBufferOutP
);
extern dsInt16_t DSMLINKAGE   tsmRetentionEvent(
       tsmRetentionEventIn_t  *tsmRetentionEventInP,
       tsmRetentionEventOut_t *tsmRetentionEventOutP
);
extern dsInt16_t DSMLINKAGE   tsmSendBufferData(
       sendBufferDataIn_t     *tsmSendBufferDataInP,
       sendBufferDataOut_t    *tsmSendBufferDataOutP
);
extern dsInt16_t DSMLINKAGE  tsmSendData(
       dsUint32_t            tsmHandle,
       DataBlk               *dataBlkPtr
);
extern dsInt16_t DSMLINKAGE  tsmSendObj(
       dsUint32_t            tsmHandle,
       tsmSendType           sendType,
       void                  *sendBuff,
       tsmObjName            *objNameP,
       tsmObjAttr            *objAttrPtr,
       DataBlk               *dataBlkPtr
);
extern dsInt16_t DSMLINKAGE  tsmSetAccess(
       dsUint32_t            tsmHandle,
       tsmAccessType         accessType,
       tsmObjName            *objNameP,
       dsChar_t              *node,
       dsChar_t              *owner
);
extern dsInt16_t DSMLINKAGE tsmSetUp(
       dsBool_t             mtFlag,
       tsmEnvSetUp          *envSetUpP
);
extern dsInt16_t DSMLINKAGE  tsmTerminate(
       dsUint32_t            tsmHandle
);
extern dsInt16_t DSMLINKAGE  tsmUpdateFS(
       dsUint32_t            tsmHandle,
       dsChar_t              *fs,
       tsmFSUpd              *fsUpdP,
       dsUint32_t            fsUpdAct
);
extern dsInt16_t DSMLINKAGE  tsmUpdateObj(
       dsUint32_t            tsmHandle,
       tsmSendType           sendType,
       void                 *sendBuff,
       tsmObjName           *objNameP,
       tsmObjAttr           *objAttrPtr,
       dsUint32_t           objUpdAct

);
extern dsInt16_t DSMLINKAGE tsmUpdateObjEx(
       tsmUpdateObjExIn_t        *tsmUpdateObjExInP,
       tsmUpdateObjExOut_t       *tsmUpdateObjExOutP
);
#if _OPSYS_TYPE == DS_NETWARE
extern void dsmAbort();
#endif
#endif /* ifdef DYNALOAD */
#if defined(__cplusplus)
   }
#endif
#endif /* _H_TSMAPIFP */
```

Appendix E. The X/Open Backup Services API

The X/Open Backup Services API (XBSA) is a set of function definitions, data structures, and return codes that the Open Group developed to present a standardized interface between applications that need to perform backup or archive operations, and the enterprise solutions that provide these services. Tivoli Storage Manager is such a solution.

See http://www.opengroup.org/publications for more information.

TSM supports the Open Group Technical Standard. See the header files included with the Tivoli Storage Manager client package for details on the implementation

The X/Open API contains the following operations:
- Start or end a Tivoli Storage Manager session
- Assign management classes to objects before storing them on a Tivoli Storage Manager server
- Backup or archive objects to a Tivoli Storage Manager server
- Restore or retrieve objects from a Tivoli Storage Manager server
- Query a Tivoli Storage Manager server for information about objects that are stored on the server
- Delete backed-up and archived objects from a Tivoli Storage Manager server

The X/Open API for Tivoli Storage Manager is available on the following platforms:
- AIX
- HP-UX
- Solaris

See the platform README_api_enu for specific information.

When you, as an application developer, install the X/Open API, you receive the following:
- The source code for the three X/Open API header files that your application needs
- The source code for a sample application and the makefile to build it
- The following files that an end user of an application would need:
  - The X/Open API shared library
  - Sample client options files
  - Documentation

For information about installing the X/Open API, see the *Tivoli Storage Manager Installation and Using Guide* for your operating system.

Setting up X/Open API options files

Use the options files to set the conditions and boundaries under which your Tivoli Storage Manager session runs.

The Tivoli Storage Manager administrator, the end user, or you can set the available options. The values of various options permit you to:

- Set up the connection to a Tivoli Storage Manager server
- Control which objects are sent to the server and with what management class they are associated

The same option can appear in more than one options file. When this happens, the file with the highest priority takes precedence. The options files, in order of decreasing priority, include:

1. Administrator options. Options that a Tivoli Storage Manager administrator sets, whether on the client or the server, override any options that are set by you or the end user. For example, the administrator can specify whether or not objects can be compressed before being sent to a Tivoli Storage Manager server. In this case, setting the compression option in the client options file has no effect. The administrator can also decide that the client should decide to permit compression. Setting the compression option in the client options file then determines if objects are compressed before they are stored.
2. The Tivoli Storage Manager options files on the UNIX or Linux platform include the user options file (dsm.opt) and the system options file (dsm.sys). The end user sets up these files when the API is first installed on the user's workstation.

For more information on the options available, see the *Tivoli Storage Manager Installation and Using Guide* for your operating system.

Building the X/Open API sample application

The API package that you receive includes a sample application. This sample application demonstrates the use of the X/Open API function calls in context. Install the sample application and view its source code to understand how you can use the function calls.

The files listed in Table 62 comprise the source files and other files that you need to build the sample application included with the X/Open API.

Table 62. Files available to build X/Open API sample application

File Name	Description
custom.h	Platform custom integer definitions header file
xbsa.h	Header file containing constants, return codes, structure and type definitions, and function prototypes for the Data Movement function group
policy.h	Header file containing structure definitions relating to policy
dsmapitd.h	Header file containing general type definitions
dsmapips.h	Header file containing platform-specific type definitions

Table 62. Files available to build X/Open API sample application (continued)

File Name		Description
xapidata.h xapint64.h xapint64.c xapipref.c xapipref.h xapiqry.c xapiqry.h xapismp.c xapiutil.h xapiutil.c xapicont.c xapidef.h xapidel.c	xapidel.h xapidisp.c xapidisp.h xapilist.c xapilist.h xapiour.c xapiour.h xapirecv.c xapirecv.h xapisend.c xapisend.h xapisess.c xapisess.h	Modules for the command-line-driven sample application
libXApi.xxx		Platform-specific suffix
makexapi.aix makexapi.sol Makexapi.hp		Makefile to build xapismp for AIX Makefile to build xapismp for Solaris
xapismp		X/Open API sample program

When you run the application, remember the following:

- You must run the Signon action before other actions.
- When you enter the object space name or the pathname, prefix them with the correct path delimiter. This is true even if you are specifying the asterisk (*) wildcard character.
- The sample application creates its own data streams when backing up or archiving objects. The object name does not correspond to any file on your workstation. The “Seed string” you enter is used to generate a pattern that can be verified when the object is restored or retrieved.

Follow these steps to compile the sample application and test the installation.

Note: Several steps have slight variations, depending on which UNIX or Linux platform you are using. See the README_api_enu file for specific information.

1. Copy the API library to the /usr/lib directory or create a symbolic link to the file from the /usr/lib directory.
2. Copy the sample application files to the target directory.
3. Copy the header files to the target directory.
4. Copy the makefile to the target directory.
5. Compile the sample.
6. Ensure that your environment variables, especially DSMI_DIR, and options files are set up. See the *Tivoli Storage Manager Installation and User's Guide* for your operating system.
7. Log on as root the first time for password registration.
8. Run xapismp to start the sample application.
9. Follow the instructions that appear on the screen.

Tivoli Storage Manager X/Open API design considerations

Before beginning the design of an X/Open application, you need to have a broad understanding of many aspects of that API.

This section describes how to use the X/Open Application Program Interface. You should be familiar with this section before you design or write an application that uses the X/Open API.

The Tivoli Storage Manager X/Open API supports the functions in XBSA Data Movement function group. These functions include the following:

Functions

BSABeginTxn
BSAChangeToken
BSACreateObject
BSADeleteObject
BSAEndData
BSAEndTxn
BSAGetData
BSAGetEnvironment
BSAGetNextQueryObject
BSAGetObject
BSAInit
BSAMarkObjectInactive
BSAQueryApiVersion
BSAQueryObject
BSASendData
BSATerminate

The X/Open API also supports the BSAResolveLifecycleGroup. See the *X/Open Specification* for detailed information on each function.

Note: The following functions are part of the XBSA Data Movement function group, but are not currently used in the X/Open API. Calls to these functions return the code, BSA_RC_BAD_CALL_SEQUENCE.

BSACreateObjectF
BSAGetObjectF
BSASetEnvironment

The API package that you receive includes a sample application (see "Building the X/Open API sample application" on page 188). Review the source code for the sample application to see examples of the X/Open API functions in context.

X/Open to Tivoli Storage Manager data field mapping

You need to know the mappings of X/Open data fields to Tivoli Storage Manager data fields.

Included here is a mapping between the X/Open data fields and the corresponding Tivoli Storage Manager fields:

X/Open fields	**Tivoli Storage Manager fields**
BSAObjectOwner	Node name
AppObjectOwner	Session owner name
SecurityToken	Password
objectspaceName	Filespace name
Left part of pathname	High-level name
Rightmost part of pathname	Low-level name
LifecycleGroup	Management class
ResourceType	FileSpaceType and objInfo

Maintaining version control in the X/Open API

All APIs have some form of version control, and X/Open is no exception. Ensure that the version of the X/Open API you are using in your application is compatible with the version of the X/Open API library that the end users have installed on their workstations.

The first API call that is issued when using the X/Open API should be **BSAQueryApiVersion**. This call:

- Confirms that the X/Open API library is installed and available on the end user's system.
- Returns the version level of the X/Open API library that the application accesses.

The X/Open API is upwardly compatible. Applications written to older versions or releases of the X/Open API library still operates correctly if the end user is running a newer version.

Determining the release of the X/Open API library is very important because some releases might have different memory requirements and data structure definitions. Downward compatibility might be possible on an individual basis. However, it is not recommended. Downward compatibility, if a requirement, is the responsibility of the application client.

The X/Open API library and the Trusted Communication Agent module (dsmtca) must be at the same level.

The **BSAQueryApiVersion** call returns the version of the X/Open API library that is installed on the end user's workstation. You can then compare the returned value with the version of the X/Open API with which the application client was built.

The version number of the application client's API is entered in the compiled object code as a set of three constants:

```
BSA_API_VERSION
BSA_API_RELEASE
BSA_API_LEVEL
```

These constants are defined in the header file custom.h. The application client's API version should usually be less than, or equal to, the X/Open API library installed on the user's system. Be careful with any other condition.

The **BSAQueryApiVersion** call can be made at any time, whether the API session has been started or not.

Starting or ending a session

Tivoli Storage Manager is a session-based product, and all activities must be performed within a Tivoli Storage Manager session. To start a session, the application starts the **BSAInit** call. This call must be performed prior to any other API call except **BSAQueryApiVersion**.

The **BSAInit** function sets up a session with the Tivoli Storage Manager server as indicated in that are passed in the call or are defined in the options files. Values in the environment pointer field are currently ignored.

Note: The application client only registers new nodes with a Tivoli Storage Manager server if the server has closed registration. If a server has open registration, any nodes that are already registered with the server are accepted by the application. However, if a server has open registration and **BSAInit** tries to register a new node, the return code, BSA_AUTH_FAILURE is generated. Application designers should tell their customers about this requirement so that customers can configure their servers accordingly.

The **ObjectOwner** fields are particularly important to a Tivoli Storage Manager session. The **BSAObjectOwner** is used as the Tivoli Storage Manager node name. The **AppObjectOwner** contains the Tivoli Storage Manager session owner name. The node name and password are used for session authentication with the Tivoli Storage Manager server. The session owner name is used to determine which objects can be accessed during the session.

Two modes for handling passwords, *prompt* and *generate*, are set in the *passwordaccess* option in the client options file. For the prompt mode, the node/owner/password must be supplied in the call to **BSAInit**. For the generate mode, the Tivoli Storage Manager trusted agent decides on the node and owner name. The password is stored in a file.

If the user's dsm.sys file sets *passwordaccess* to *prompt*, then the Tivoli Storage Manager node and password (security token) must be supplied. The session owner can be whatever name you select. An empty string for the session owner ([0]='\0') is used to mean the root owner. The application has control of the object owner values.

If the user's dsm.sys file sets *passwordaccess* to *generate*, then a value is not supplied for **BSAObjOwner** or **AppObjOwner**. These fields must be empty strings. The node name that is used is the machine name, and the session owner is the login user's name. The security token field is ignored in this situation.

If an application passes either node or session owner values when the mode is generate, it receives a return code of TSM_RC_PSWD_GEN. In this case, if your application supports *passwordaccess* set to *generate*, **BSAInit** must be issued again with empty **ObjectOwner** fields. If your application requires *passwordaccess* set to *prompt*, then stop and tell the user to change the option in their dsm.sys file.

You should follow **BSAInit** with a call to **BSAGetEnvironment** to retrieve the actual node and owner that is used for the session. If dsm.sys has *passwordaccess* set to *generate*, these values are *node = hostname*, and *owner = login user*.

When *passwordaccess* is set to *generate*, the root user must start the first Tivoli Storage Manager session. This is necessary to create the file where the password is stored.

End a session with a **BSATerminate** call. This causes the X/Open API to close any connection with the Tivoli Storage Manager server and free all resources associated with this session.

Note: Only one session can be active per call of the API. However, applications on UNIX or Linux platforms can circumvent this restriction by running with multiple processes, with each process owning its own Tivoli Storage Manager session.

Passwordaccess option

If the end user has set *passwordaccess* to *generate* in the client options file, and that user is not the root user, then the Trusted Communication Agent (dsmtca) child process is forced to manage the session with the Tivoli Storage Manager server. The SIGCLD signal is used during ending. If you set *passwordaccess* to *prompt*, then a child process is not used.

Session security

Tivoli Storage Manager, a session-based system, has security components that permit applications to start sessions in a secure manner. These security measures prohibit unauthorized access to the server and help insure system integrity.

Every session that the server starts must complete a sign-on process. This sign-on process requires a password that, when coupled with the node name of the client, insures proper authorization when connecting to the server. The application client is responsible for providing this password to the X/Open API for session initialization.

Passwords have expiration periods associated with them. If a **BSAInit** call fails with the password-expired return code (BSA_RC_TOKEN_EXPIRED), update the password before you can successfully establish the session.

Only the root session owner can change the password. First, make the **BSAInit** call with an empty string in the appObjectOwner field. Then, call **BSAChangeToken** to update the password.

Objects stored in the system also have ownerships associated with them. See "Identifying the object using X/Open API" on page 203 to understand how an application can take advantage of this to support multi-user applications. The application client is responsible for insuring that security and ownership rules are met once a session is started.

Determining the session parameters using X/Open API

After the **BSAInit** function is called to start a session, the application can make a call to **BSAGetEnvironment** to determine the parameters set for the session. The **BSAGetEnvironment** call returns such items as the node, owner, and server names used for the session, and the maximum number of objects that can be created in a single transaction.

The **objectOwner.bsaObjectOwner** field contains the Tivoli Storage Manager node name. This corresponds to the **BSAObjOwner** field when *passwordaccess* is set to *prompt*. When *passwordaccess* is set to *generate*, this field contains the machine name.

The **objectOwner.appObjectOwner** field contains the Tivoli Storage Manager owner name. This corresponds to the **AppObjOwner** field when *passwordaccess* is set to *prompt*. When *passwordaccess* is set to *generate*, this field contains the login name.

The calling application must allocate an array of ADSM_ENV_STRS elements with strings of size BSA_MAX_DESC for the environment output. The application must also allocate an array of character pointers with ADSM_ENV_STRS+1 elements. The extra element is for the NULL termination pointer.

```
char *envP[ADSM_ENV_STRS+1];
char envStrs[ADSM_ENV_STRS] [BSA_MAX_DESC];
for (i=0; i<ADSM_ENV_STRS; i++)
  envP[i] = envStrs[i];

envP[i] = NULL;
rc = BSAGetEnvironment(bsaHandle, &objOwner, envP);
```

Figure 21. An example of BSAGetEnvironment

The format of the output is:

```
envStrs[0] = "TSMSRVR=xxx"
envStrs[1] = "TSMMAXOBJ=xx"
envStrs[1] = "TSMSRVRSTANZA=xx"
```

where:

- TSMSRVR is the Tivoli Storage Manager server name.
- TSMMAXOBJ is the number of objects that can be created within a single transaction.
- TSMSRVRSTANZA is the **adsmServerName** value.

Associating a management class with objects using X/Open API

One of the primary features that Tivoli Storage Manager offers is the use of policies (management classes) to define how objects are stored and managed in Tivoli Storage Manager storage.

A management class is associated with an object when the object is backed up or archived. This management class determines the following:

- How many versions of the object are kept if they are backed up
- How long to keep archive copies
- Where the object is inserted in the storage hierarchy on the server

Management classes have two components: backup copy group and an archive copy group. A copy group is a set of attributes that define the management policies for an object that is backed up or archived. If a backup operation is being performed, the attributes in the backup copy group apply. If an archive is being performed, the attributes in the archive copy group apply.

Because the use of policy is a very important component of Tivoli Storage Manager, the API requires all objects that are sent to the server first be assigned to a management class. There are two ways to do this.

- Use an include-exclude list. The Tivoli Storage Manager product uses an include-exclude list to perform management class binding. The **BSACreateObject** and **BSAResolveLifecycleGroup** calls check the object that is stored against the include-exclude list. When it finds an include statement that matches the name of the object, the management class specified in the statement is assigned to the object. If a management class is not specified, or the object is not explicitly listed in the include-exclude list, the object is assigned to the default management class.
- Override the include-exclude list. The **BSACreateObject** call takes an **ObjectDescriptor** as an input parameter. You can assign a particular management class to an object by placing the name of the management class in the **LGName** field of the **ObjectDescriptor**.

Note: The backup or archive copy group in a particular management class can be empty or NULL. If an object is bound to the NULL backup copy group, then that object cannot be backed up. If an object is bound to the NULL archive copy group, the object cannot be archived.

The transaction model

All data sent to, received from, or deleted from Tivoli Storage Manager storage by the X/Open API is performed within a transaction. This provides a high level of data integrity for the Tivoli Storage Manager product, but it does impose some restrictions that an application client must take into consideration.

Start a transaction with a call to **BSABeginTxn** and end it with a call to **BSAEndTxn**.

A single transaction is an atomic action. Data sent or received within the bounds of a transaction either is all committed at the end of the transaction, or all rolled back if the transaction ends prematurely.

The Tivoli Storage Manager product supports the use of only a single operation type within a transaction. For example, you cannot perform both a send and a get operation within the bounds of a single transaction. The one exception is during a get operation, where you precede the get with a query operation.

Transactions can consist of either single objects or multiple objects. Smaller objects should be sent or received in a multiple object transaction. This greatly improves total system performance, because transaction overhead is decreased. The application client determines whether single or multiple transactions are appropriate.

All objects within a multiple object transaction must be sent to, or received from, the same copy destination. If you must send an object to, or receive it from, a different destination than the previous object, end the current transaction and start a new one. Within the new transaction, you can send or receive the object to the new copy destination.

The Tivoli Storage Manager product limits the number of objects that can be sent or received in a multiple object transaction. You can find this limit by calling **BSAGetEnvironment** and examining the MAXOBJ value.

The application client must keep track of the objects sent or received within a transaction in order to perform retry processing or error processing if the transaction is ended prematurely. Either the server or the client can stop a transaction at any time. The application client must be prepared to handle sudden transaction ends that it did not start.

Querying the Tivoli Storage Manager system

The X/Open API permits an application client to query a Tivoli Storage Manager server for information on the records stored there. You can define a set of criteria that the records on the server must meet in order to be returned by the query. All query operations must be done within the bounds of a transaction.

See “The transaction model.”

A query operation consists of the following steps:

1. Make the **BSABeginTxn** call to start a transaction.

2. Define the parameters of your query. Use the data fields in the QueryDescriptor structure to specify the parameters of your query. Start by setting the **copyType** field to **backup**, **archive**, or any, depending on whether you want to query only backup copies, only archive versions, or both.

   For all queries, you can specify an object name in the **objName** field, or use wildcard characters to identify a group of objects. For backup queries, use the status field to specify only active or inactive copies, or both. For archive queries, specify the description in the description field and set the upper and lower boundaries of the create and expiration times in these fields: **createTimeLB**, **createTimeUB**, **expireTimeLB**, and **expireTimeUB**.
3. Make the **BSAQueryObject** call. To start the query operation,make the **BSAQueryObject** call, passing in the **QueryDescriptor** structure. One of the following three codes is returned:
   - BSA_RC_MORE_DATA. More than one object satisfied the search parameters. The object descriptor for the first object is returned in the ObjectDescriptor field. Go to step 4.
   - BSA_RC_NO_MORE_DATA. Only one object satisfied the search parameters. The object descriptor for the object is returned in the ObjectDescriptor field. Go to step 5.
   - BSA_RC_NO_MATCH. There were no objects that satisfied the search parameters. Go to step 5.
4. Make the **BSAGetNextQueryObject** call. If more than one object satisfied the query parameters, then a **BSAGetNextQueryObject** call must be made to obtain each object after the first. The object descriptor for each object is added to the **ObjectDescriptor** structure.

   After each object is returned, check the return code. If the **BSAGetNextQueryObject** call returns the code BSA_RC_MORE_DATA, make the **BSAGetNextQueryObject** call again. If there is no more data, go to the next step.
5. Make the **BSAEndTxn** call to end the transaction. When all query data has been retrieved or no further query data is needed, the **BSAEndTxn** call must be made to end the transaction and stop the query process. This causes the X/Open API to flush any remaining data from the query stream and release any resources utilized for the query.

Flowchart example for X/Open query operations

You can use a visual example on using X/Open functions for query operations.

Figure 22 on page 197 displays the flowchart for performing query operations.

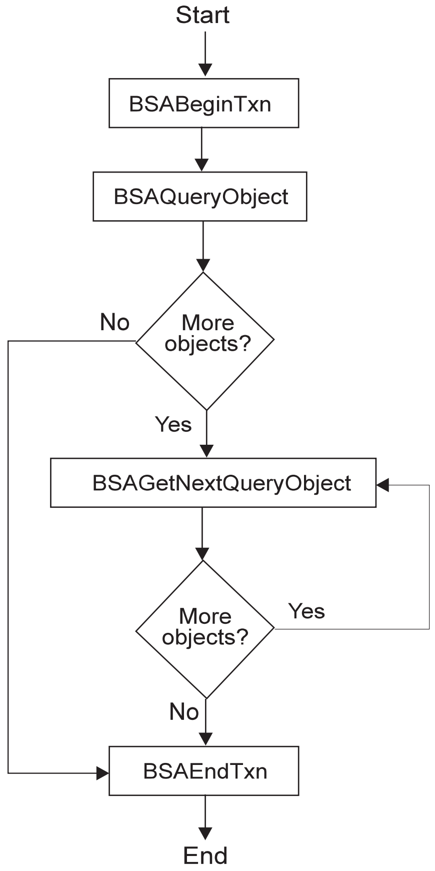


Figure 22. Flowchart for query operations

Sending data to a server using X/Open API

The X/Open API permits application clients to send data to Tivoli Storage Manager server storage. Data can be either backed up or archived. All send operations must be performed within the bounds of a transaction. See "The transaction model" on page 195.

The backup component of the Tivoli Storage Manager system supports multiple versions of named objects that are stored on the server. Any object that is backed up to the server with the same name as an object which is already stored on the server from that client is subject to version control. Objects are considered to be in active or inactive states on the server. The latest copy of an object on the server that has not been deactivated is in the active state. Any other object, whether it is an older version or a deactivated copy, is considered to be inactive. Different management criteria defined by the management class constructs are assigned to active and inactive objects on the server.

The archive component of the Tivoli Storage Manager system stores objects on the server with retention or expiration period controls instead of version control. Each object stored is considered to be unique, even though its name might be the same as an object already archived. This permits an application to archive the same object multiple times, but with different expiration times that are assigned to each copy of the object.

The value of the compression option in the end user's dsm.sys file determines whether Tivoli Storage Manager compresses the object during a send operation.

Some types of data (for example, data that is already compressed) might actually grow larger when processed with the compression algorithm. When this happens, the return code, TSM_RC_ERROR, is generated and added to the Tivoli Storage Manager error log (dsierror.log). If you recognize that this might happen, but want the send operation to continue anyway, tell the end users to specify the following option in their options file before the application runs:

```
COMPRESSAlways Yes
```

A send operation consists of the following steps:

1. Make the **BSABeginTxn** call to start a transaction.
2. Make the **BSAResolveLifecycleGroup** call.

   This call is optional. Use it to associate a particular management class with an object that you are storing on the Tivoli Storage Manager server. If you do not call **BSAResolveLifecycleGroup**, a management class is associated with the object during the call to **BSACreateObject**. For more information, see "Associating a management class with objects using X/Open API" on page 194.
3. Make the **BSACreateObject** call.

   The **BSACreateObject** call takes an **ObjectDescriptor** structure as an input parameter. This structure contains information about the object that is stored, such as the name of the object, and if it is backed up or archived.

   The **ObjectDescriptor.Owner.bsaObjectOwner** value must match the value that is used on the **BSAInit** call. The **ObjectDescriptor.Owner.appObjectOwner** value must also match the one that is used on the **BSAInit** call, unless it was an empty string, signifying that the session was started with the root owner. In this case the object owner can be any value.

   The sizes of the **objInfo** and **desc** fields in the **ObjectDescriptor** structure are set by Tivoli Storage Manager. These sizes are determined by the constants TSM_MAX_OBJINFO and TSM_MAX_DESC in the custom.h header file.

   BSACreateObject can also send the first block of data to the Tivoli Storage Manager server. If the object has more data, go to the next step. If there is no more data, go to step 5.
4. Make the **BSASendData** call.

   Repeat this call as many times as necessary until the entire object has been sent to the Tivoli Storage Manager server.
5. Make the **BSAEndData** call.

   The **BSAEndData** call signifies that there is no more data for a particular object.
6. If you want to send more than one object to the Tivoli Storage Manager server, repeat steps 3 through 5 for each object. Note that all objects sent within the same transaction must be for the same **objectspaceName**.

   Tivoli Storage Manager limits the number of objects that can be sent in one transaction. The limit is determined by the constant MAXOBJ. You can get this value by calling **BSAGetEnvironment**.
7. Make the **BSAEndTxn** call to end the transaction.

Flowchart example for X/Open backup or archive operations

You can use a visual example on using X/Open functions for performing backup or archive operations within a transaction.

Figure 23 displays the flowchart for performing backup or archive operations within a transaction.

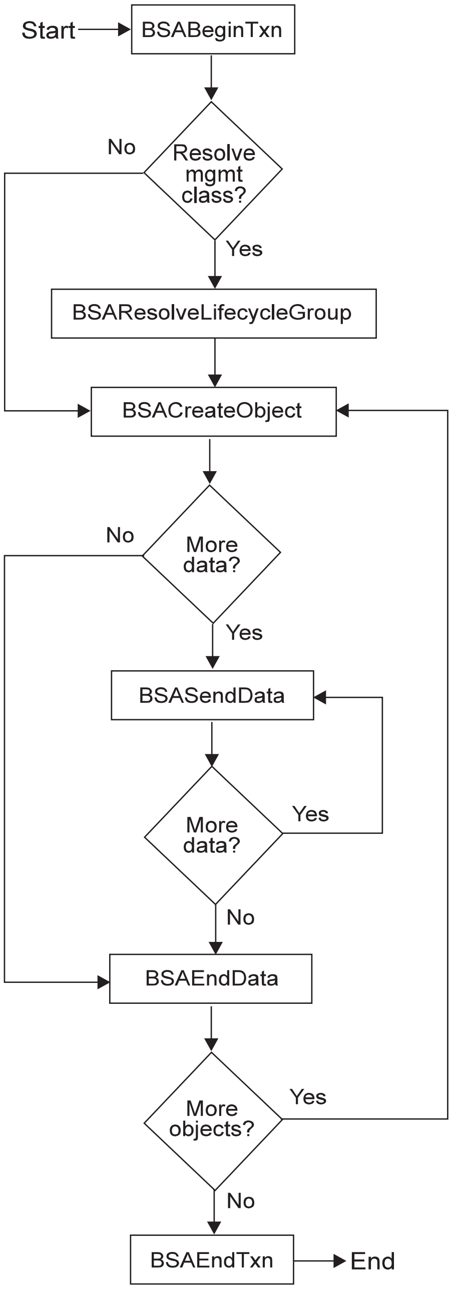


Figure 23. Flowchart for backup and archive operations

The primary feature in this diagram is the loop between the following X/Open API calls from within a transaction:

- **BSACreateObject**
- **BSASendData**

- **BSAEndData**

Receiving data from a server using the X/Open API

The X/Open API permits application clients to receive data from Tivoli Storage Manager storage using the restore and retrieve functions of the product. Restore accesses objects that were previously backed up, and retrieve accesses objects that were previously archived. All restore and retrieve operations must be performed within the bounds of a transaction. See "The transaction model" on page 195.

Note: Only the API can restore or retrieve objects that were backed up or archived with API calls.

Once a session is established with the Tivoli Storage Manager server, use the following procedure to restore or retrieve data:

1. Make the **BSABeginTxn** call to start a transaction.
2. Make the **BSAQueryObject** call to query the Tivoli Storage Manager server for backup or archive data. (This step can be performed outside the transaction.)

   Before beginning a restore or retrieve operation, query the Tivoli Storage Manager server to determine what objects can be received from storage. To issue the query, first fill in the applicable fields in the **QueryDescriptor** structure with the desired search parameters. Then make the **BSAQueryObject** call with the **QueryDescriptor**.

   If the session was started with a NULL owner name, it is not necessary to specify the owner field. If the session was started with an explicit owner name, then only objects that explicitly have that owner name associated with them are returned.

   The query returns all information in an **ObjectDescriptor** structure. Different information is returned depending on whether the object was originally backed up or archived. For example, a query on backup objects returns information on whether an object is active or inactive, while a query on archive objects returns information such as the object descriptions.

   All queries return all information that was originally stored with the object, in addition to the following:

   copyid

   The **copyid** provides an eight-byte number that uniquely identifies this object for this node in Tivoli Storage Manager storage. Use this ID to request a specific object from storage for restore or retrieve processing.

   restoreOrder

   The **restoreOrder** provides a mechanism for receiving objects from Tivoli Storage Manager storage in the most efficient manner possible. Sort the objects to restore on this value to insure that tapes are mounted only once and are read from front to back.

   Keep some or all of the query information for later processing. Keep the **copyid** and **restoreOrder** fields because they are needed for the actual restore operation. Keep any other information that is needed to properly open a data file or identify a destination.
3. Determine the objects to restore or retrieve from the server. Once the backup or archive query has been performed, the application client must determine which objects, if any, are to be restored or retrieved.
4. If more than one object is selected, sort the objects on the restore order field.

Once the objects to restore or retrieve are selected, they must be sorted in ascending order (low to high) by the **restoreOrder** field. This sorting is critical to the performance of the restore operation. Sorting the objects on the **restoreOrder** field means that the data is read from the server in the most efficient order. All data on disk is restored first, followed by data on media classes that require volume mounts (such as tape). The **restoreOrder** field also insures that data on tape is read in order with processing starting at the front of a tape and progressing towards the end.

Properly sorting on the **restoreOrder** field means that duplicate tape mounts and unnecessary tape rewinds do not occur.

5. Make the **BSAGetObject** call.

   The **BSAGetObject** call uses the **copyType** and **copyid** fields of the **ObjectDescriptor** to begin obtaining the first object from the system. The call begins a restore or retrieve operation by identifying the object that is requested from the data stream.

   BSAGetObject obtains the first block of data that is associated with the object. If the object has more data, go to the next step. If the return code is BSA_RC_NO_MORE_DATA, go to step 7.
6. Make the **BSAGetData** call.

   Repeat this call as many times as necessary until the object has been received from the Tivoli Storage Manager server.
7. Make the **BSAEndData** call.

   The **BSAEndData** call signifies that there is no more data for a particular object.
8. If you want to receive more than one object from the Tivoli Storage Manager server, repeat steps 5 through 7 for each object.
9. Make the **BSAEndTxn** call to end the transaction.

   After all data for all requested objects has been received, the **BSAEndTxn** call must be made. You can also use this call to discard any remaining data in the restore stream for objects not yet received.

Flowchart example for X/Open restore or retrieve operations

You can use a visual example on using X/Open functions for restore or retrieve operations.

Figure 24 on page 202 displays the flowchart for performing restore or retrieve operations.

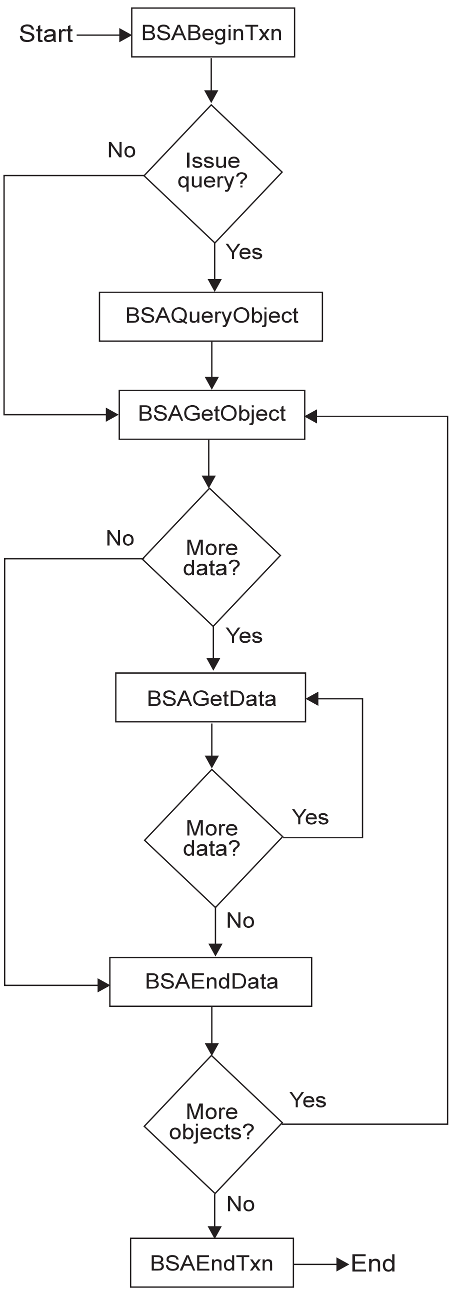


Figure 24. Flowchart for restore and retrieve operations

Deleting objects from the server using X/Open API

The X/Open API applications can make calls to either delete objects that were archived or deactivate objects that were backed up. The former is dependent on the node authorization that is given when a Tivoli Storage Manager administrator registered the node. Administrators can specify whether nodes can delete archive objects.

The **BSADeleteObject** call is used for deleting archive objects, and the BSAMarkObjectInactive call is used for deactivating backup objects.

When deleting an archive object, the object is marked in Tivoli Storage Manager storage for deletion when the system next performs its object expiration cycle. Once an archive object is deleted from the server, it cannot be retrieved.

When a backup object on the Tivoli Storage Manager server is deactivated, the object moves from an active state to an inactive state. These states have different retention policies associated with them that are based on the management class assigned.

Note: A call to **BSAMarkObjectInactive** affects all objects with the same objType and the same name.

A call to **BSADeleteObject** or to **BSAMarkObjectInactive** is always made within the bounds of a transaction. The flowcharts in Figure 25 show how a call to **BSADeleteObject** or **BSAMarkObjectInactive** is preceded by a call to **BSABeginTxn** and followed by a call to **BSAEndTxn**.

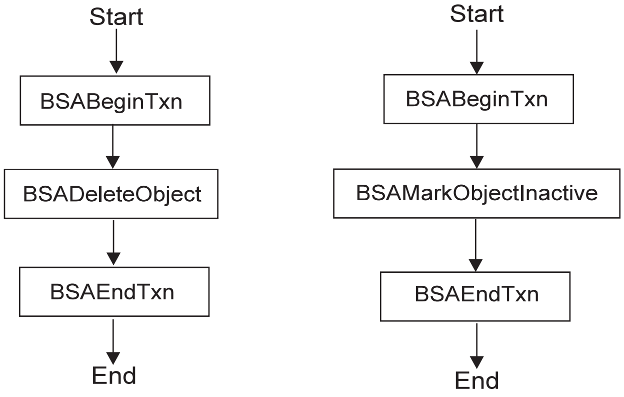


Figure 25. Flowcharts for delete archive (left) and deactivate backup (right) operations

Identifying the object using X/Open API

The Tivoli Storage Manager server can be viewed as an object storage server whose main goal is to efficiently store and retrieve named objects.

The server has two main storage areas:

- The database contains all metadata (name, attributes, and so forth) associated with an object.
- The data storage contains the actual object data. The data storage is actually a storage hierarchy that the system administrator defines. Data is efficiently stored and managed on either online or offline media, depending on cost and access needs.

Each object stored on the server has a name associated with it. The client controls the following key components of the name:

- Object space name
- Pathname
- Object type

When making decisions about naming objects for an application, remember that it might be necessary to externalize the full object names to the end user. Specifically, the end user might need to specify the object in an include or exclude statement when the application is run.

The following example demonstrates what the application client would code on a UNIX or Linux platform:

```
/myobjspace/pathname
```

Object space name

The object space name is a very important component. This name can be the name of a file system, disk drive, or any other high-level qualifier that groups related data together.

Tivoli Storage Manager uses the object space to identify the file system or disk drive the data is located on. Actions can be performed on all entities within an object space with relative ease, such as querying all objects within a specified object space.

The Tivoli Storage Manager server has administrative commands to query the object spaces on a given node in Tivoli Storage Manager storage, and delete them if necessary. All data that the application client stores must have an object space name associated with it. Select the name carefully to group similar data together in the system.

An application client should select different object space names than the file system names a backup-archive client would use, in order to avoid possible interference. The application client should publish its object space names, so that end users can identify the objects for include and exclude statements, if necessary.

Pathnames

Another component of the object name is the pathname. The pathname consists of the directory path the object belongs in, and the actual name of the object in that directory path. When the object space name and pathname are concatenated, they must form a syntactically correct name on the operating system on which the client is running.

It is not necessary for the name to exist as an object on the system or resemble the actual data on the local file system. However, the name must meet the standard naming rules in order to be properly processed for management classes.

Object types in X/Open API

The object type identifies the object as either a file or a directory. A file is an object that contains both attributes and binary data. A directory is an object that contains only attributes.

Tivoli Storage Manager also accepts the value BSAObjectType_DATABASE but treats it as BSAObjectType_FILE.

Setting the owner name

Each object has an owner name associated with it. The rules governing what objects can be accessed depend on what owner name is used when a session is started. This object owner value can be used to control access to the object.

If a session is started with an empty string for the owner, that session owner is treated with session, (or root) authority. This session can perform any action on any object that is owned by this node regardless of the actual owner of that object. The session owner is set during the call to **BSAInit** in the AppObjectOwner field of the ObjectOwner structure.

If a session is started with a specific owner name, the session can only perform actions on objects that have that owner name associated with them. Backups or archives into the system all must have this owner name associated with them. Any queries performed only return values that have this owner name associated with them. The object owner value is set during the **BSACreateObject** call in the Owner field of the ObjectDescriptor structure.

Table 63 summarizes the conditions under which a user has access to an object.

Table 63. Summary of user access to objects

Session owner	Object owner	User access
“ ” (empty string) (root, system owner)	“ ” (empty string)	Yes
“ ” (empty string) (root, system owner)	specific name	Yes
specific name	“ ” (empty string)	No
specific name	same name	Yes
specific name	different name	No

Using X/Open functions with Tivoli Storage Manager

You need to understand how to use specific X/Open functions with Tivoli Storage Manager.

Table 64 describes the following X/Open functions and considerations on how to use them with Tivoli Storage Manager.

Table 64. X/Open function considerations

X/Open function	Consideration
BSAChangeToken	Only the root session owner, or TSM-Authorized session owner can run this function.

Table 64. X/Open function considerations (continued)

X/Open function	**Consideration**
BSACreateObject	Multiple **CreateObject** calls within a single transaction must be for the same **objectspaceName**. Tivoli Storage Manager considers all fields in the ObjectDescriptor to be input fields and does not alter them. The **ObjectDescriptor.resourceType** field is used as the Tivoli Storage Manager file space **fsType** value, and is also stored in the Tivoli Storage Manager **objInfo** area. The **ObjectDescriptor.Owner.bsaObjectOwner** value must match that value that is used on the BSAInit. The **ObjectDescriptor.Owner.appObjectOwner** value must match that which was used on the **BSAInit** if it was not root (blank). If **BSAInit** starts a session with the root owner, then the object owner can be any value. The following fields from the **ObjectDescriptor** are used: • **owner.bsaObjectOwner** • **owner.appObjectOwner** • **objName.objectSpaceName** • **objName.pathName** • **copyType** • **lGName** (management class) • **Size** • **resourceType** • **objectType** (for backup, this can be FILE, DIRECTORY, DATABASE (which is treated like FILE); for archive, this can be FILE or DIRECTORY) • **desc** • **mdobjInfo** For **BSACreateObject**, Tivoli Storage Manager has a limit on the number of objects that can be created in a single transaction. Tivoli Storage Manager returns the value on the **BSAGetEnvironment** call with the TSMMAXOBJ keyword.
BSADeleteObject	With Tivoli Storage Manager, this call is only meaningful for archive objects. For backup objects, use **BSAMarkObjectInactive**.
BSAGetEnvironment	Tivoli Storage Manager returns the server name and maximum objects per transaction. See Figure 21 on page 194 for an example.
BSAGetObject	The following fields from **objectDescriptor** are used: • **copyType** • **copyid** The object name fields are not used. Precede the **BSAGetObject** call with a **BSAQueryObject** call to obtain the *copyid* value.

Table 64. X/Open function considerations (continued)

X/Open function	Consideration
BSAInit	The **BSAObjectOwner** field is used as the Tivoli Storage Manager node name. The **AppObjectOwner** field is used as the Tivoli Storage Manager session owner name. Tivoli Storage Manager has two modes, prompt or generate, to handle passwords. For the prompt mode, the node/owner/password values must be supplied. For the generate mode, the password is saved in a file that the root or TSM-Authorized user must start. Thereafter, node/owner/password values should not be passed on the **BSAInit**. Values passed on the **environmentPtr** are ignored.
BSAQueryObject	The **ObjectDescriptor.Owner.bsaObjectOwner** value must match that which was used on the **BSAInit**. The **ObjectDescriptor.Owner.appObjectOwner** value must match that which was used on the **BSAInit** if it was not root (blank). If **BSAInit** started a session with the root owner, then the object owner can be any value. For **copyType Backup**, the following fields are used: • **owner.bsaObjectOwner** • **owner.appObjectOwner** • **objName.objectSpaceName** • **objName.pathName** • **objectType** (DATABASE is treated as FILE) • **status** For Archive, the following fields are used: • **owner.bsaObjectOwner** • **owner.appObjectOwner** • **objName.objectSpaceName** • **objName.pathName** • **createTime** • **expireTime** • **objectType** • **desc**

Tivoli Storage Manager changes to the XBSA header files

The X/Open API contains the header files custom.h, xbsa.h, and policy.h. Tivoli Storage Manager uses these header files with some changes.

Changes to custom.h

The Tivoli Storage Manager X/Open API supports the following additional constants and return codes in custom.h:

```
/* Constants used
 */
#define TSM_MAX_DESC               100     /* TSM max Desc size             */
#define TSM_MAX_OBJINFO            100     /* TSM max object information size*/
#define TSM_LOWEST_BOUND          0x0000  /* value for LowerBound max       */
#define TSM_HIGHEST_BOUND         0xFFFF  /* value for UpperBound max       */
#define TSM_ENV_STRS              2       /* number of env strings          */
#define ObjectDescriptorVersion     1       /* ver for ObjectDescriptor       */
#define UserDescriptorVersion       1       /* ver for UserDescriptor         */
#define BSAObjectType_DATABASE      4       /* ObjectType for Databases       */

/* Return Codes Used
 */
#define BSA_RC_OK                           0x00
#define BSA_RC_SUCCESS                      0x00

#define TSM_RC_ERROR                       0x60 /* see TSM error log        */
#define TSM_RC_INVALID_NODE                0x61 /* BSAObjOwner not match Init*/
#define TSM_RC_INVALID_COPYTYPE            0x62 /* invalid copyType          */
#define TSM_RC_INVALID_OBJTYPE             0x63 /* invalid objectType        */
#define TSM_RC_INVALID_STATUS              0x64 /* invalid object status     */
#define TSM_RC_INVALID_ST_VER              0x65 /* invalid structure version */
#define TSM_RC_OWNER_TOO_LONG              0x66 /* owner too long            */
#define TSM_RC_PSWD_TOO_LONG               0x67 /* pswd  too long            */
#define TSM_RC_PSWD_GEN                    0x68 /* pswd access = generate    */
```

Changes to policy.h

The Tivoli Storage Manager X/Open API supports the following changes to the function prototypes in policy.h:

```
/* For the function prototypes, int and long have been                    */
/* replaced with typedefs from custom.h.                                  */
/*                                                                        */
/* BSAGetNextQueryObject defined in xbsa.h because it should be part      */
/* of the Data Movement subset.                                           */
```

Changes to xbsa.h

The Tivoli Storage Manager X/Open API supports the following changes to the type definitions in xbsa.h:

```
/* Changed tm typedef to 'struct tm' for AIX compiler                     */
/* ref BSAEvent, ObjectDescriptor, QueryDescriptor, Schedule              */
/*                                                                        */
/* For the function prototypes, int and long have been                    */
/* replaced with typedefs from custom.h.                                  */
/*                                                                        */
/* Included BSAGetNextQueryObject function prototype here since it was    */
/* accidentally omitted from the Data movement subset.                    */
```

Appendix F. Accessibility features for Tivoli Storage Manager

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features of Tivoli Storage Manager are described in this topic.

Accessibility features

The following list includes the major accessibility features in Tivoli Storage Manager:

- Keyboard-only operation
- Interfaces that are commonly used by screen readers
- Keys that are discernible by touch but do not activate just by touching them
- Industry-standard devices for ports and connectors
- The attachment of alternative input and output devices
- User documentation provided in HTML and PDF format. Descriptive text is provided for all documentation images.

The Tivoli Storage Manager Information Center, and its related publications, are accessibility-enabled.

Keyboard navigation

The Tivoli Storage Manager for Windows Console follows Microsoft conventions for all keyboard navigation and access. Drag and Drop support is managed using the Microsoft Windows Accessibility option known as MouseKeys. For more information about MouseKeys and other Windows accessibility options, please refer to the Windows Online Help (keyword: MouseKeys).

Tivoli Storage Manager follows AIX operating system conventions for keyboard navigation and access.

Tivoli Storage Manager follows HP-UX operating-system conventions for keyboard navigation and access.

Tivoli Storage Manager follows Linux operating-system conventions for keyboard navigation and access.

Tivoli Storage Manager follows Macintosh operating-system conventions for keyboard navigation and access.

Tivoli Storage Manager follows Sun Solaris operating-system conventions for keyboard navigation and access.

Tivoli Storage Manager follows z/OS operating-system conventions for keyboard navigation and access.

Vendor software

Tivoli Storage Manager includes certain vendor software that is not covered under the IBM license agreement. IBM makes no representation about the accessibility features of these products. Contact the vendor for the accessibility information about its products.

Related accessibility information

You can view the publications for Tivoli Storage Manager in Adobe® Portable Document Format (PDF) using the Adobe Acrobat Reader. You can access these or any of the other documentation PDFs at the IBM Publications Center at http://www.ibm.com/shop/publications/order/.

IBM and accessibility

For more information about the commitment that IBM has to accessibility, see the IBM Human Ability and Accessibility Center at http://www.ibm.com/able.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information in softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at ″Copyright and trademark information″ at http://www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript®, and the PostScript logo is a registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT®, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Glossary

This glossary includes terms and definitions for IBM Tivoli Storage Manager.

To view glossaries for other IBM products, go to http://www.ibm.com/software/globalization/terminology/.

The following cross-references are used in this glossary:

- *See* refers the reader from a term to a preferred synonym, or from an acronym or abbreviation to the defined full form.
- *See also* refers the reader to a related or contrasting term.

A

absolute mode
In storage management, a backup copy-group mode that specifies that a file is considered for incremental backup even if the file has not changed since the last backup. See also *modified mode*.

access control list (ACL)
In computer security, a list associated with an object that identifies all the subjects that can access the object and their access rights. For example, an access control list is associated with a file that identifies the users who can access that file and their access rights.

access mode
An attribute of a storage pool or a storage volume that specifies whether the server can write to or read from the storage pool or storage volume. The access mode can be read/write, read-only, or unavailable. Volumes in primary storage pools can also have an access mode of destroyed. Volumes in copy storage pools can also have an access mode of offsite.

acknowledgment
The transmission of acknowledgment characters as a positive response to a data transmission.

ACL See *access control list*.

activate
To validate the contents of a policy set and then make it the active policy set.

active-data pool
A named set of storage pool volumes that contain only active versions of client backup data.

active file system
A file system to which space management has been added. With space management, tasks for an active file system include automatic migration, reconciliation, selective migration, and recall. Contrast with *inactive file system*.

active policy set
The activated policy set that contains the policy rules in use by all client nodes that are assigned to the policy domain. See also *policy domain* and *policy set*.

active version
The most recent backup copy of a file stored. The active version of a file cannot be deleted until a backup process detects that the user has either replaced the file with a newer version or has deleted the file from the file server or workstation. Contrast with *inactive version*.

activity log
A log that records normal activity messages that are generated by the server. These messages include information about server and client operations, such as the start time of sessions or device I/O errors.

adaptive subfile backup
A type of backup that sends only changed portions of a file to the server, instead of sending the entire file. Adaptive subfile backup reduces network traffic and increases the speed of the backup.

administrative client
A program that runs on a file server, workstation, or mainframe that administrators use to control and monitor the Tivoli Storage Manager server. Contrast with *backup-archive client*.

administrative command schedule
A database record that describes the planned processing of an administrative

command during a specific time period. See also *client schedule*.

administrative privilege class
See *privilege class*.

administrative session
A period of time during which an administrator user ID communicates with a server to perform administrative tasks. Contrast with *client node session*.

administrator
A user who is registered to the server as an administrator, and who is authorized to perform tasks and issue commands through the assignment of an administrative privilege class.

Advanced Program-to-Program Communication (APPC)
An implementation of the SNA LU 6.2 protocol that allows interconnected systems to communicate and share the processing of programs.

agent node
A client node that has been granted proxy authority to perform operations on behalf of another client node, which is the target node.

aggregate
An object, stored in one or more storage pools, consisting of a group of logical files that are packaged together. See also *logical file* and *physical file*.

aggregate data transfer rate
A performance statistic that indicates the average number of bytes that were transferred per second while processing a given operation.

APPC See *Advanced Program-to-Program Communication*.

application client
A program that is installed on a system to protect an application. The Tivoli Storage Manager server provides backup services to an application client.

archive
To copy programs, data, or files to another storage media, usually for long-term storage or security. Contrast with *retrieve*.

archive copy
A file or group of files that was archived to server storage.

archive copy group
A policy object containing attributes that control the generation, destination, and expiration of archived files.

archive-retention grace period
The number of days that the storage manager retains an archived file when the server is unable to rebind the file to an appropriate management class. See also *bind*.

association
(1) The defined relationship between a client node and a client schedule. An association identifies the name of a schedule, the name of the policy domain to which the schedule belongs, and the name of a client node that performs scheduled operations.

(2) On a configuration manager, the defined relationship between a profile and an object such as a policy domain. Profile associations define the configuration information that is distributed to a managed server when it subscribes to the profile.

audit To check for logical inconsistencies between information that the server has and the actual condition of the system. The storage manager can audit information about items such as volumes, libraries, and licenses. For example, when a storage manager audits a volume, the server checks for inconsistencies between information about backed-up or archived files that are stored in the database and the actual data that are associated with each backup version or archive copy in server storage.

authentication
The process of checking a user's password before permitting user access to the Tivoli Storage Manager server. Authentication can be turned on or off by an administrator with system privilege.

authentication rule
A specification that another user can use to either restore or retrieve files from storage.

authority
The right to access objects, resources, or functions. See also *privilege class*.

authorization rule
A specification that permits another user to either restore or retrieve a user's files from storage.

authorized user
A user who has administrative authority for the Tivoli Storage Manager client on a workstation. This user changes passwords, performs open registrations, and deletes file spaces.

AutoFS
See *automounted file system*.

automatic detection
A feature that detects, reports, and updates the serial number of a drive or library in the database when the path from the local server is defined.

automatic migration
The process that is used to automatically move files from a local file system to storage, based on options and settings that are chosen by a root user on a workstation. See also *threshold migration* and *demand migration*.

automatic reconciliation
The process that is used to reconcile file systems at regular intervals. The intervals are set by a user with root user authority. See also *reconciliation*.

automounted file system (AutoFS)
A file system that is managed by an automounter daemon. The automounter daemon monitors a specified directory path, and automatically mounts the file system to access data.

B

backup-archive client
A program that runs on a workstation or file server and provides a means for users to back up, archive, restore, and retrieve files. Contrast with *administrative client*.

backup copy group
A policy object containing attributes that control the generation, destination, and expiration of backup versions of files. A backup copy group belongs to a management class.

backup-retention grace period
The number of days the storage manager retains a backup version after the server is unable to rebind the file to an appropriate management class.

backup set
A portable, consolidated group of active versions of backup files that are generated for a backup-archive client.

backup set collection
A group of backup sets that are created at the same time and which have the same backup set name, volume names, description, and device classes. The server identifies each backup set in the collection by its node name, backup set name, and file type.

backup version
A file or directory that a client node backed up to server storage. More than one backup version can exist in server storage, but only one backup version is the active version. See also *active version* and *inactive version*.

bindery
A database that consists of three system files for a NetWare server. The files contain user IDs and user restrictions.

bind To associate a file with a management class name. See *rebind*.

C

cache To place a duplicate copy of a file on random access media when the server migrates a file to another storage pool in the hierarchy.

cache file
A snapshot of a logical volume created by Logical Volume Snapshot Agent. Blocks are saved immediately before they are modified during the image backup and their logical extents are saved in the cache files.

CAD See *client acceptor*.

central scheduler
A function that permits an administrator to schedule client operations and administrative commands. The operations can be scheduled to occur periodically or on a specific date. See *client schedule* and *administrative command schedule*.

client A software program or computer that requests services from a server.

client acceptor daemon (CAD)
See *client acceptor*.

client acceptor
An HTTP service that serves the Java applet for the Web client to Web browsers. On Windows systems, the client acceptor is installed and run as a service. On AIX, UNIX, and Linux systems, the client acceptor is run as a daemon, and is also called the *client acceptor daemon* (CAD).

client domain
The set of drives, file systems, or volumes that the user selects to back up or archive data, using the backup-archive client.

client node
A file server or workstation on which the backup-archive client program has been installed, and which has been registered to the server.

client node session
A session in which a client node communicates with a server to perform backup, restore, archive, retrieve, migrate, or recall requests. Contrast with *administrative session*.

client options file
An editable file that identifies the server and communication method, and provides the configuration for backup, archive, hierarchical storage management, and scheduling.

client option set
A group of options that are defined on the server and used on client nodes in conjunction with client options files.

client-polling scheduling mode
A method of operation in which the client queries the server for work. Contrast with *server-prompted scheduling mode*.

client schedule
A database record that describes the planned processing of a client operation during a specific time period. The client operation can be a backup, archive, restore, or retrieve operation, a client operating system command, or a macro. See also *administrative command schedule*.

client/server
Pertaining to the model of interaction in distributed data processing in which a program on one computer sends a request to a program on another computer and awaits a response. The requesting program is called a client; the answering program is called a server.

client system-options file
A file, used on AIX, UNIX, or Linux system clients, containing a set of processing options that identify the servers to be contacted for services. This file also specifies communication methods and options for backup, archive, hierarchical storage management, and scheduling. This file is also called the dsm.sys file. See also *client user-options file*.

client user-options file
A file that contains the set of processing options that the clients on the system use. The set can include options that determine the server that the client contacts, and options that affect backup operations, archive operations, hierarchical storage management operations, and scheduled operations. This file is also called the dsm.opt file. For AIX, UNIX, or Linux systems, see also *client system-options file*.

closed registration
A registration process in which only an administrator can register workstations as client nodes with the server. Contrast with *open registration*.

collocation
The process of keeping all data belonging to a single-client file space, a single client node, or a group of client nodes on a minimal number of sequential-access volumes within a storage pool. Collocation can reduce the number of volumes that must be accessed when a large amount of data must be restored.

collocation group
A user-defined group of client nodes whose data is stored on a minimal number of volumes through the process of collocation.

commit point
A point in time when data is considered consistent.

communication method
The method by which a client and server exchange information. See also *Transmission Control Protocol/Internet Protocol*.

Common Programming Interface for Communications (CPI-C)
A call-level interface that provides a consistent application programming interface (API) for applications that use program-to-program communications. CPI-C uses LU 6.2 architecture to create a set of interprogram services that can establish and end a conversation, send and receive data, exchange control information, and notify a partner program of errors.

communication protocol
A set of defined interfaces that permit computers to communicate with each other.

compression
A function that removes repetitive characters, spaces, or strings of characters from the data being processed and replaces the repetitive characters with control characters. Compression reduces the amount of storage space that is required for the data.

configuration manager
A server that distributes configuration information, such as policies and schedules, to managed servers according to their profiles. Configuration information can include policy and schedules. See also *managed server* and *profile*.

conversation
A connection between two programs over a session that allows them to communicate with each other while processing a transaction.

copy backup
A full backup in which the transaction log files are not deleted so that backup procedures that use incremental or differential backups are not disrupted

copy group
A policy object containing attributes that control how backup versions or archive copies are generated, where backup versions or archive copies are initially located, and when backup versions or archive copies expire. A copy group belongs to a management class. See also *archive copy group*, *backup copy group*, *backup version*, and *management class*.

copy storage pool
A named set of volumes that contain copies of files that reside in primary storage pools. Copy storage pools are used only to back up the data that is stored in primary storage pools. A copy storage pool cannot be a destination for a backup copy group, an archive copy group, or a management class (for space-managed files). See also *primary storage pool* and *destination*.

CPI-C See *Common Programming Interface for Communications*.

D

daemon
A program that runs unattended to perform continuous or periodic functions, such as network control.

damaged file
A physical file in which Tivoli Storage Manager has detected read errors.

data access control mode
A mode that controls whether a command can access a migrated file, see a migrated file as zero-length, or receive an input/output error if it attempts to access a migrated file. See also *execution mode*.

database backup series
One full backup of the database, plus up to 32 incremental backups made since that full backup. Each full backup that is run starts a new database backup series. A number identifies each backup series.

database snapshot
A complete backup of the entire database to media that can be taken off-site. When a database snapshot is created, the current database backup series is not interrupted. A database snapshot cannot have incremental database backups associated with it. See also *database backup series*. Contrast with *full backup*.

data manager server
A server that collects metadata information for client inventory and manages transactions for the storage

agent over the local area network. The data manager server informs the storage agent with applicable library attributes and the target volume identifier.

data mover
A device that moves data on behalf of the server. A network-attached storage (NAS) file server is a data mover.

data storage-management application-program interface (DSMAPI)
A set of functions and semantics that can monitor events on files, and manage and maintain the data in a file. In an HSM environment, a DSMAPI uses events to notify data management applications about operations on files, stores arbitrary attribute information with a file, supports managed regions in a file, and uses DSMAPI access rights to control access to a file object.

default management class
A management class that is assigned to a policy set. This class is used to govern backed up or archived files when a file is not explicitly associated with a specific management class through the include-exclude list.

demand migration
The process that is used to respond to an out-of-space condition on a file system for which hierarchical storage management (HSM) is active. Files are migrated to server storage until space usage drops to the low threshold that was set for the file system. If the high threshold and low threshold are the same, one file is migrated.

desktop client
The group of backup-archive clients that includes clients on Microsoft Windows, Apple, and Novell NetWare operating systems.

destination
A copy group or management class attribute that specifies the primary storage pool to which a client file will be backed up, archived, or migrated.

device class
A named set of characteristics that are applied to a group of storage devices. Each device class has a unique name and represents a device type of disk, file, optical disk, or tape.

device configuration file
(1) For a server, a file that contains information about defined device classes, and, on some servers, defined libraries and drives. The information is a copy of the device configuration information in the database.

(2) For a storage agent, a file that contains the name and password of the storage agent, and information about the server that is managing the SAN-attached libraries and drives that the storage agent uses.

device driver
A program that provides an interface between a specific device and the application program that uses the device.

disaster recovery manager (DRM)
A function that assists in preparing and using a disaster recovery plan file for the server.

disaster recovery plan
A file that is created by the disaster recovery manager (DRM) that contains information about how to recover computer systems if a disaster occurs and scripts that can be run to perform some recovery tasks. The file includes information about the software and hardware that is used by the server, and the location of recovery media.

domain
A grouping of client nodes with one or more policy sets, which manage data or storage resources for the client nodes. See *policy domain* or *client domain*.

DRM See *disaster recovery manager*.

DSMAPI
See *data storage-management application-program interface*.

dynamic serialization
A type of copy serialization in which a file or folder is backed up or archived on the first attempt regardless of whether it changes during a backup or archive.

E

EA See *extended attribute*.

EB See *exabyte*.

EFS See *Encrypted File System*.

Encrypted File System (EFS)
A file system that uses file system-level encryption.

enterprise configuration
A method of setting up servers so that the administrator can distribute the configuration of one of the servers to the other servers, using server-to-server communication. See also *configuration manager*, *managed server*, *profile*, and *subscription*.

enterprise logging
The process of sending events from a Tivoli Storage Manager server to a designated event server. The event server routes the events to designated receivers, such as to a user exit. See also *event*.

error log
A data set or file that is used to record error information about a product or system.

estimated capacity
The available space, in megabytes, of a storage pool.

event (1) An administrative command or a client operation that is scheduled to be run using Tivoli Storage Manager scheduling.

(2) A message that an Tivoli Storage Manager server or client issues. Messages can be logged using Tivoli Storage Manager event logging.

event record
A database record that describes actual status and results for events.

event server
A server to which other servers can send events for logging. The event server routes the events to any receivers that are enabled for the sending server's events.

exabyte (EB)
For processor storage, real and virtual storage, and channel volume, 1 152 921 504 606 846 976 bytes. For disk storage capacity and communications volume, 1 000 000 000 000 000 000 bytes.

exclude
The process of identifying files in an include-exclude list. This process prevents the files from being backed up or migrated whenever a user or schedule enters an incremental or selective backup operation. A file can be excluded from backup and space management, backup only, or space management only.

exclude-include list
See *include-exclude list*.

execution mode
A mode that controls the space-management related behavior of commands that run under the **dsmmode** command.

expiration
The process by which files, data sets, or objects are identified for deletion because their expiration date or retention period has passed.

expiring file
A migrated or premigrated file that has been marked for expiration and removal from storage. If a stub file or an original copy of a premigrated file is deleted from a local file system, or if the original copy of a premigrated file is updated, the corresponding migrated or premigrated file is marked for expiration the next time reconciliation is run.

extend
To increase the portion of available space that can be used to store database or recovery log information.

extended attribute (EA)
Names or value pairs that are associated with files or directories. There are three classes of extended attributes: user attributes, system attributes, and trusted attributes.

external library
A type of library that is provided by Tivoli Storage Manager that permits LAN-free data movement for StorageTek libraries that are managed by Automated Cartridge System Library Software (ACSLS). To activate this function, the Tivoli Storage Manager library type must be EXTERNAL.

F

file access time
On AIX, UNIX, or Linux systems, the time when the file was last accessed.

file age
For migration prioritization purposes, the number of days since a file was last accessed.

file device type
A device type that specifies the use of sequential access files on disk storage as volumes.

file server
A dedicated computer and its peripheral storage devices that are connected to a local area network that stores programs and files that are shared by users on the network.

file space
A logical space in server storage that contains a group of files that have been backed up or archived by a client node, from a single logical partition, file system, or virtual mount point. Client nodes can restore, retrieve, or delete their file spaces from server storage. In server storage, files belonging to a single file space are not necessarily stored together.

file space ID (FSID)
A unique numeric identifier that the server assigns to a file space when it is stored in server storage.

file state
The space management mode of a file that resides in a file system to which space management has been added. A file can be in one of three states: resident, premigrated, or migrated. See also *resident file*, *premigrated file*, and *migrated file*.

file system migrator (FSM)
A kernel extension that intercepts all file system operations and provides any space management support that is required. If no space management support is required, the operation is passed to the operating system, which performs its normal functions. The file system migrator is mounted over a file system when space management is added to the file system.

file system state
The storage management mode of a file system that resides on a workstation on which the hierarchical storage management (HSM) client is installed. A file system can be in one of these states: native, active, inactive, or global inactive.

frequency
A copy group attribute that specifies the minimum interval, in days, between incremental backups.

FSID See *file space ID*.

FSM See *file system migrator*.

full backup
The process of backing up the entire server database. A full backup begins a new database backup series. See also *database backup series* and *incremental backup*. Contrast with *database snapshot*.

fuzzy backup
A backup version of a file that might not accurately reflect what is currently in the file because the file was backed up at the same time as it was being modified.

fuzzy copy
A backup version or archive copy of a file that might not accurately reflect the original contents of the file because it was backed up or archived the file while the file was being modified. See also *backup version* and *archive copy*.

G

General Parallel File System™
A high-performance shared-disk file system that can provide data access from nodes in a cluster environment.

gigabyte (GB)
In decimal notation, 1 073 741 824 when referring to memory capacity; in all other cases, it is defined as 1 000 000 000.

global inactive state
The state of all file systems to which space management has been added when space management is globally deactivated for a client node. When space management is globally deactivated, hierarchical storage management (HSM) cannot perform migration, recall, or reconciliation. However, a root user can update space management settings and add space management to additional file systems. Users can access resident and premigrated files.

Globally Unique Identifier (GUID)
An algorithmically determined number that uniquely identifies an entity within a system.

GPFS™
See *General Parallel File System*.

GPFS node set
A mounted, defined group of GPFS file systems.

group backup
The backup of a group containing a list of files from one or more file space origins.

GUID See *Globally Unique Identifier*.

H

hierarchical storage management (HSM)
A function that automatically distributes and manages data on disk, tape, or both by regarding devices of these types and potentially others as levels in a storage hierarchy that range from fast, expensive devices to slower, cheaper, and possibly removable devices. The objectives are to minimize access time to data and maximize available media capacity.

hierarchical storage management (HSM) client
A client program that works with the Tivoli Storage Manager server to provide hierarchical storage management (HSM) for a system. See also *hierarchical storage management* and *space manager client*.

HSM See *hierarchical storage management*.

HSM client
See *hierarchical storage management client*.

I

ILM See *information lifecycle management*.

image A file system or raw logical volume that is backed up as a single object.

image backup
A backup of a full file system or raw logical volume as a single object.

inactive file system
A file system for which space management has been deactivated. Contrast with *active file system*.

inactive version
A backup version of a file that is either not the most recent backup version, or that is a backup version of a file that no longer exists on the client system. Inactive backup versions are eligible for expiration processing according to the management class assigned to the file. Contrast with *active version*.

include-exclude file
A file containing statements to determine the files to back up and the associated management classes to use for backup or archive. See also *include-exclude list*.

include-exclude list
A list of options that include or exclude selected files for backup. An exclude option identifies files that should not be backed up. An include option identifies files that are exempt from the exclusion rules or assigns a management class to a file or a group of files for backup or archive services.

incremental backup
(1) A copy of all database data that has changed since the most recent successful full backup operation. An incremental backup is also known as a *cumulative backup image* because each incremental backup includes the contents of the previous incremental backup.

(2) The process of backing up information in the database that is new or changed since the last full backup. Contrast with *full backup*. See also *database backup series*.

(3) For Data Protection for Microsoft Exchange Server, a backup in which the transaction logs are backed up and then cleared.

individual mailbox restore
See *mailbox restore*.

information lifecycle management (ILM)
GPFS policy-based file management for storage pools and file sets.

i-node The internal structure that describes the individual files on AIX, UNIX, or Linux systems. An i-node contains the node, type, owner, and location of a file.

i-node number
A number specifying a particular i-node file in the file system.

IP address
A unique address for a device or logical unit on a network that uses the IP standard.

J

job file
A generated file that contains configuration information for a migration job. The file is XML format and can be created and edited in the hierarchical storage management (HSM) client for Windows client graphical user interface.

journal-based backup
A method for backing up Windows clients and AIX clients that exploits the change notification mechanism in a file to improve incremental backup performance by reducing the need to fully scan the file system.

journal daemon
On AIX, UNIX, or Linux systems, a program that tracks change activity for files residing in file systems.

journal service
In Microsoft Windows, a program that tracks change activity for files residing in file systems.

K

kilobyte (KB)
For processor storage, real and virtual storage, and channel volume, 210 or 1 024 bytes. For disk storage capacity and communications volume, 1 000 bytes.

L

LAN See *local area network*.

LAN-free data movement
The movement of client data between a client system and a storage device on a storage area network (SAN), bypassing the local area network. This process is also referred to as *LAN-free data transfer*.

LAN-free data transfer
See *LAN-free data movement*.

leader data
Bytes of data, from the beginning of a migrated file, that are stored in the file's corresponding stub file on the local file system. The amount of leader data that is stored in a stub file depends on the stub size that is specified.

library
(1) A repository for demountable recorded media, such as magnetic disks and magnetic tapes.

(2) A collection of one or more drives, and possibly robotic devices (depending on the library type), which can be used to access storage volumes.

library client
A server that uses server-to-server communication to access a library that is managed by another storage management server. See also *library manager*.

library manager
A server that controls device operations when multiple storage management servers share a storage device. See also *library client*.

local Pertaining to a device, file, or system that is accessed directly from a user's system, without the use of a communication line.

local area network (LAN)
A network that connects several devices in a limited area (such as a single building or campus) and that can be connected to a larger network.

local shadow volumes
Data that is stored on shadow volumes localized to a disk storage subsystem.

LOFS See *loopback virtual file system.*

logical file
A file that is stored in one or more server storage pools, either by itself or as part of an aggregate. See also *aggregate* and *physical file*.

logical occupancy
The space that is used by logical files in a storage pool. This space does not include the unused space created when logical files are deleted from aggregate files, so it might be less than the physical occupancy.

logical unit (LU)
An access point through which a user or application program accesses the Systems Network Architecture (SNA) network to communicate with another user or application program.

logical unit number (LUN)
In the Small Computer System Interface (SCSI) standard, a unique identifier that is used to differentiate devices, each of which is a logical unit (LU).

logical volume
A portion of a physical volume that contains a file system.

logical volume backup
A back up of a file system or logical volume as a single object.

Logical Volume Snapshot Agent (LVSA)
Software that can act as the snapshot provider for creating a snapshot of a logical volume during an online image backup.

loopback virtual file system (LOFS)
A file system that is created by mounting a directory over another local directory, also known as mount-over-mount. A LOFS can also be generated using an automounter.

LU See *logical unit*.

LUN See *logical unit number*.

LVSA See *Logical Volume Snapshot Agent*.

M

MB See *megabyte*.

macro file
A file that contains one or more storage manager administrative commands, which can be run only from an administrative client using the MACRO command. Contrast with *Tivoli Storage Manager command script*.

mailbox restore
A function that restores Microsoft Exchange Server data (from IBM Data Protection for Exchange backups) at the mailbox level or mailbox-item level.

managed object
In Tivoli Storage Manager, a definition in the database of a managed server that was distributed to the managed server by a configuration manager. When a managed server subscribes to a profile, all objects that are associated with that profile become managed objects in the database of the managed server. In general, a managed object cannot be modified locally on the managed server. Objects can include policy, schedules, client option sets, server scripts, administrator registrations, and server and server group definitions.

managed server
A Tivoli Storage Manager server that receives configuration information from a configuration manager using a subscription to one or more profiles. Configuration information can include definitions of objects such as policy and schedules. See also *configuration manager*, *subscription*, and *profile*.

management class
A policy object that users can bind to each file to specify how the server manages the file. The management class can contain a backup copy group, an archive copy group, and space management attributes. See also *copy group*, *space manager client*, *bind*, and *rebind*.

maximum transmission unit
The largest possible unit of data that can be sent on a given physical medium in a single frame. For example, the maximum transmission unit for Ethernet is 1500 bytes.

megabyte (MB)
(1) 1 048 576 bytes (two to the twentieth power) when used in this publication.

(2) For processor storage, real and virtual storage, and channel volume, 2 to the power of 20 or 1 048 576 bits. For disk storage capacity and communications volume, 1 000 000 bits.

metadata
Data that describes the characteristics of data; descriptive data.

migrate
To move data from one storage location to another. In Tivoli Storage Manager products, migrating can mean moving data from a client node to server storage, or moving data from one storage pool to the next storage pool defined in the server storage hierarchy. In both cases the movement is controlled by policy, such as thresholds that are set. See also *migration threshold*.

migrated file
A file that has been copied from a local file system to Tivoli Storage Manager storage. For HSM clients on UNIX or Linux systems, the file is replaced with a stub file on the local file system. On Windows systems, creation of the stub file

is optional. See also *stub file* and *resident file*. For HSM clients on UNIX or Linux systems, contrast with *premigrated file*.

migrate-on-close recall mode
A mode that causes a migrated file to be recalled back to its originating file system temporarily. Contrast with *normal recall mode* and *read-without-recall recall mode*.

migration job
A specification of files to migrate, and actions to perform on the original files after migration. See also *job file*.

migration threshold
High and low capacities for storage pools or file systems, expressed as percentages, at which migration is set to start and stop.

mirroring
The process of writing the same data to multiple locations at the same time. Mirroring data protects against data loss within the recovery log.

mode A copy group attribute that specifies whether to back up a file that has not been modified since the last time the file was backed up. See *modified mode* and *absolute mode*.

modified mode
In storage management, a backup copy-group mode that specifies that a file is considered for incremental backup only if it has changed since the last backup. A file is considered a changed file if the date, size, owner, or permissions of the file have changed. See also *absolute mode*.

mount limit
The maximum number of volumes that can be simultaneously accessed from the same device class. The mount limit determines the maximum number of mount points. See also *mount point*.

mount point
On the Tivoli Storage Manager server, a logical drive through which volumes in a sequential access device class are accessed. For removable-media device types, such as tape, a mount point is a logical drive that is associated with a physical drive. For the file device type, a mount point is a logical drive that is associated with an I/O stream. The number of mount points for a device class is defined by the value of the mount limit attribute for that device class. See also *mount limit*.

mount retention period
The maximum number of minutes that the server retains a mounted sequential-access media volume that is not being used before it dismounts the sequential-access media volume.

mount wait period
The maximum number of minutes that the server waits for a sequential-access volume mount request to be satisfied before canceling the request.

MTU See *maximum transmission unit*.

N

Nagle algorithm
An algorithm that reduces congestion of TCP/IP networks by combining smaller packets and sending them together.

named pipe
A type of interprocess communication that permits message data streams to pass between peer processes, such as between a client and a server.

NAS See *network-attached storage*.

NAS node
A client node that is a network-attached storage (NAS) file server. Data for the NAS node is transferred by a NAS file server that is controlled by the network data management protocol (NDMP). A NAS node is also called a NAS file server node.

native file system
A file system that is locally added to the file server and is not added for space management. The hierarchical storage manager (HSM) client does not provide space management services to the file system.

native format
A format of data that is written to a storage pool directly by the Tivoli Storage Manager server. Contrast with *non-native data format*.

NDMP
See *Network Data Management Protocol*.

NetBIOS
See *Network Basic Input/Output System*.

network-attached storage (NAS) file server
A dedicated storage device with an operating system that is optimized for file-serving functions. A NAS file server can have the characteristics of both a node and a data mover.

Network Basic Input/Output System (NetBIOS)
A standard interface to networks and personal computers that is used on local area networks to provide message, print-server, and file-server functions. Application programs that use NetBIOS do not have to handle the details of LAN data link control (DLC) protocols.

Network Data Management Protocol (NDMP)
A protocol that allows a network storage-management application to control the backup and recovery of an NDMP-compliant file server, without installing vendor-acquired software on that file server.

network data-transfer rate
A rate that is calculated by dividing the total number of bytes that are transferred by the data transfer time. For example, this rate can be the time that is spent transferring data over a network.

node A file server or workstation on which the backup-archive client program has been installed, and which has been registered to the server.

node name
A unique name that is used to identify a workstation, file server, or PC to the server.

node privilege class
A privilege class that gives an administrator the authority to remotely access backup-archive clients for a specific client node or for all clients in a policy domain. See also *privilege class*.

non-native data format
A format of data that is written to a storage pool that differs from the format that the server uses for operations.

normal recall mode
A mode that causes a migrated file to be copied back to its originating file system when it is accessed.

O

offline volume backup
A backup in which the volume is locked so that no other system applications can access it during the backup operation.

online volume backup
A backup in which the volume is available to other system applications during the backup operation.

open registration
A registration process in which users can register their workstations as client nodes with the server. Contrast with *closed registration*.

operator privilege class
A privilege class that gives an administrator the authority to disable or halt the server, enable the server, cancel server processes, and manage removable media. See also *privilege class*.

options file
A file that contains processing options. On Windows and NetWare systems, the file is called dsm.opt. On AIX, UNIX, Linux, and Mac OS X systems, the file is called dsm.sys.

originating file system
The file system from which a file was migrated. When a file is recalled using normal or migrate-on-close recall mode, it is always returned to its originating file system.

orphaned stub file
A file for which no migrated file can be found on the Tivoli Storage Manager server that the client node is contacting for space management services. For example, a stub file can be orphaned when the client system-options file is modified to contact a server that is different than the one to which the file was migrated.

out-of-space protection mode
A mode that controls whether the program intercepts out-of-space conditions. See also *execution mode*.

P

pacing
In SNA, a technique by which the receiving system controls the rate of transmission of the sending system to prevent overrun.

packet In data communication, a sequence of binary digits, including data and control signals, that is transmitted and switched as a composite whole.

page A defined unit of space on a storage medium or within a database volume.

partial-file recall mode
A recall mode that causes the hierarchical storage management (HSM) function to read just a portion of a migrated file from storage, as requested by the application accessing the file.

password generation
A process that creates and stores a new password in an encrypted password file when the old password expires. Automatic generation of a password prevents password prompting. Password generation can be set in the options file (passwordaccess option). See also *options file*.

path An object that defines a one-to-one relationship between a source and a destination. Using the path, the source accesses the destination. Data can flow from the source to the destination, and back. An example of a source is a data mover (such as a network-attached storage [NAS] file server), and an example of a destination is a tape drive.

pattern-matching character
See *wildcard character*.

physical file
A file that is stored in one or more storage pools, consisting of either a single logical file, or a group of logical files that are packaged together as an aggregate. See also *aggregate* and *logical file*.

physical occupancy
The amount of space that is used by physical files in a storage pool. This space includes the unused space that is created when logical files are deleted from aggregates. See also *physical file*, *logical file*, and *logical occupancy*.

plug-in
A self-contained software component that modifies (adds, or changes) the function in a particular system. When a plug-in is added to a system, the foundation of the original system remains intact.

policy domain
A grouping of policy users with one or more policy sets, which manage data or storage resources for the users. The users are client nodes that are associated with the policy domain.

policy privilege class
A privilege class that gives an administrator the authority to manage policy objects, register client nodes, and schedule client operations for client nodes. Authority can be restricted to certain policy domains. See also *privilege class*.

policy set
A group of rules in a policy domain. The rules specify how data or storage resources are automatically managed for client nodes in the policy domain. Rules can be contained in management classes. See also *active policy set* and *management class*.

premigrated file
A file that has been copied to Tivoli Storage Manager storage, but has not been replaced with a stub file on the local file system. An identical copy of the file resides both on the local file system and in Tivoli Storage Manager storage. Premigrated files occur on UNIX and Linux file systems to which space management has been added. Contrast with *migrated file* and *resident file*.

premigrated files database
A database that contains information about each file that has been premigrated to Tivoli Storage Manager storage. The database is stored in a hidden directory named .SpaceMan in each file system to which space management has been added.

premigration
The process of copying files that are eligible for migration to Tivoli Storage Manager storage, but leaving the original file intact on the local file system.

premigration percentage
A space management setting that controls whether the next eligible candidates in a file system are premigrated following threshold or demand migration.

primary storage pool
A named set of volumes that the server uses to store backup versions of files, archive copies of files, and files migrated from client nodes. See also *destination* and *copy storage pool*.

privilege class
A level of authority that is granted to an administrator. The privilege class determines which administrative tasks the administrator can perform. See also *node privilege class*, *operator privilege class*, *policy privilege class*, *storage privilege class*, and *system privilege class*.

profile
A named group of configuration information that can be distributed from a configuration manager when a managed server subscribes. Configuration information can include registered administrator IDs, policies, client schedules, client option sets, administrative schedules, storage manager command scripts, server definitions, and server group definitions. See also *configuration manager* and *managed server*.

Q

quota (1) For HSM on AIX, UNIX, or Linux systems, the limit (in megabytes) on the amount of data that can be migrated and premigrated from a file system to server storage.

(2) For HSM on Windows systems, a user-defined limit to the space that is occupied by recalled files.

R

randomization
The process of distributing schedule start times for different clients within a specified percentage of the schedule's startup window.

raw logical volume
A portion of a physical volume that is comprised of unallocated blocks and has no journaled file system (JFS) definition. A logical volume is read/write accessible only through low-level I/O functions.

read-without-recall recall mode
A mode that causes hierarchical storage management (HSM) to read a migrated file from storage without storing it back on the local file system. The last piece of information read from the file is stored in a buffer in memory on the local file system. Contrast with *normal recall mode* and *migrate-on-close recall mode*.

rebind
To associate a backed-up file with a new management class name. For example, rebinding occurs when the management class associated with a file is deleted. See also *bind*.

recall In Tivoli Storage Manager, to copy a migrated file from server storage back to its originating file system using the space management client. See also *transparent recall*, *selective recall*, and *recall mode*.

recall mode
A mode that is assigned to a migrated file with the dsmattr command that determines how the file is processed when it is recalled. It determines whether the file is stored on the local file system, is migrated back to Tivoli Storage Manager storage when it is closed, or is read from Tivoli Storage Manager storage without storing it on the local file system.

receiver
A server repository that contains a log of server and client messages as events. For example, a receiver can be a file exit, a user exit, or the Tivoli Storage Manager server console and activity log. See also *event*.

reclamation
The process of consolidating the remaining data from many sequential-access volumes onto fewer, new sequential-access volumes.

reclamation threshold
The percentage of space that a sequential-access media volume must have before the server can reclaim the volume. Space becomes reclaimable when files are expired or are deleted.

reconciliation
The process of synchronizing a file system with the Tivoli Storage Manager server, and then removing old and obsolete objects from the Tivoli Storage Manager server.

recovery log
A log of updates that are about to be

written to the database. The log can be used to recover from system and media failures. The recovery log consists of the active log (including the log mirror) and archive logs.

register
To define a client node or administrator ID that can access the server.

registry
A repository that contains access and configuration information for users, systems, and software.

resident file
On a Windows system, a complete file on a local file system that might also be a migrated file because a migrated copy can exist in Tivoli Storage Manager storage. On a UNIX or Linux system, a complete file on a local file system that has not been migrated or premigrated, or that has been recalled from Tivoli Storage Manager storage and modified. Contrast with *stub file* and *premigrated file*. See *migrated file*.

restore
To copy information from its backup location to the active storage location for use. For example, to copy information from server storage to a client workstation.

retention
The amount of time, in days, that inactive backed-up or archived files are kept in the storage pool before they are deleted. Copy group attributes and default retention grace periods for the domain define retention.

retrieve
To copy archived information from the storage pool to the workstation for use. The retrieve operation does not affect the archive version in the storage pool.

roll back
To remove changes that were made to database files since the last commit point.

root user
A system user who operates without restrictions. A root user has the special rights and privileges needed to perform administrative tasks.

S

SAN See *storage area network*.

schedule
A database record that describes client operations or administrative commands to be processed. See *administrative command schedule* and *client schedule*.

scheduling mode
The type of scheduling operation for the server and client node that supports two scheduling modes: client-polling and server-prompted.

scratch volume
A labeled volume that is either blank or contains no valid data, that is not defined, and that is available for use.

script A series of commands, combined in a file, that carry out a particular function when the file is run. Scripts are interpreted as they are run. Contrast with *Tivoli Storage Manager command script*.

Secure Sockets Layer (SSL)
A security protocol that provides communication privacy. With SSL, client/server applications can communicate in a way that is designed to prevent eavesdropping, tampering, and message forgery.

selective backup
The process of backing up certain files or directories from a client domain. The files that are backed up are those that are not excluded in the include-exclude list. The files must meet the requirement for serialization in the backup copy group of the management class that is assigned to each file. Contrast with *incremental backup*.

selective migration
The process of copying user-selected files from a local file system to Tivoli Storage Manager storage and replacing the files with stub files on the local file system. Contrast with *threshold migration* and *demand migration*.

selective recall
The process of copying user-selected files from Tivoli Storage Manager storage to a local file system. Contrast with *transparent recall*.

serialization
The process of handling files that are modified during backup or archive processing. See *dynamic serialization*, *static*

serialization, *shared static serialization*, and *shared dynamic serialization*.

server
A software program or a computer that provides services to other software programs or other computers.

server options file
A file that contains settings that control various server operations. These settings affect such things as communications, devices, and performance.

server-prompted scheduling mode
A client/server communication technique where the server contacts the client node when tasks must be done. Contrast with *client-polling scheduling mode*.

server storage
The primary, copy, and active-data storage pools that are used by the server to store user files such as backup versions, archive copies, and files migrated from space manager client nodes (space-managed files). See also *active-data pool*, *primary storage pool*, *copy storage pool*, *storage pool volume*, and *volume*.

session
A logical or virtual connection between two stations, software programs, or devices on a network that allows the two elements to communicate and exchange data.

session resource usage
The amount of wait time, processor time, and space that is used or retrieved during a client session.

shared dynamic serialization
A value for serialization that specifies that a file must not be backed up or archived if it is being modified during the operation. Tivoli Storage Manager retries the backup or archive operation a number of times; if the file is being modified during each attempt, Tivoli Storage Manager will back up or archive the file on its last try. See also *serialization*. Contrast with *dynamic serialization*, *shared static serialization*, and *static serialization*.

shared library
A library device that is used by multiple storage manager servers.

shared static serialization
A copy-group serialization value that specifies that a file must not be modified during a backup or archive operation. Tivoli Storage Manager attempts to retry the operation a number of times. If the file is in use during each attempt, the file is not backed up or archived. See also *serialization*. Contrast with *dynamic serialization*, *shared dynamic serialization*, and *static serialization*.

snapshot
An image backup type that consists of a point-in-time view of a volume.

space-managed file
A file that is migrated from a client node by the space manager client. The space manager client recalls the file to the client node on demand.

space management
The process of keeping sufficient free storage space available on a local file system for new data by migrating files to server storage. Synonymous with *hierarchical storage management*.

space manager client
A program that runs on a UNIX or Linux system to manage free space on the local file system by migrating files to server storage. The program can recall the files either automatically or selectively. Also called *hierarchical storage management (HSM) client*.

space monitor daemon
A daemon that checks space usage on all file systems for which space management is active, and automatically starts threshold migration when space usage on a file system equals or exceeds its high threshold.

sparse file
A file that is created with a length greater than the data it contains, leaving empty spaces for the future addition of data.

special file
On AIX, UNIX, or Linux systems, a file that defines devices for the system, or temporary files that are created by processes. There are three basic types of special files: first-in, first-out (FIFO); block; and character.

SSL
See *Secure Sockets Layer*.

stabilized file space
A file space that exists on the server but not on the client.

stanza A group of lines in a file that together have a common function or define a part of the system. Each stanza is identified by a name that occurs in the first line of the stanza. Depending on the type of file, a stanza is ended by the next occurrence of a stanza name in the file, or by an explicit end-of-stanza marker. A stanza can also be ended by the end of the file.

startup window
A time period during which a schedule must be initiated.

static serialization
A copy-group serialization value that specifies that a file must not be modified during a backup or archive operation. If the file is in use during the first attempt, the storage manager cannot back up or archive the file. See also *serialization*. Contrast with *dynamic serialization*, *shared dynamic serialization*, and *shared static serialization*.

storage agent
A program that enables the backup and restoration of client data directly to and from storage attached to a storage area network (SAN).

storage area network (SAN)
A dedicated storage network that is tailored to a specific environment, combining servers, systems, storage products, networking products, software, and services.

storage hierarchy
(1) A logical order of primary storage pools, as defined by an administrator. The order is typically based on the speed and capacity of the devices that the storage pools use. The storage hierarchy is defined by identifying the next storage pool in a storage pool definition. See also *storage pool*.

(2) An arrangement of storage devices with different speeds and capacities. The levels of the storage hierarchy include: main storage, such as memory and direct-access storage device (DASD) cache; primary storage (DASD containing user-accessible data); migration level 1 (DASD containing data in a space-saving format); and migration level 2 (tape cartridges containing data in a space-saving format).

storage pool
A named set of storage volumes that are the destination that is used to store client data. A storage pool contains backup versions, archive copies, and files that are migrated from space manager client nodes. A primary storage pool is backed up to a copy storage pool. See also *primary storage pool*, *copy storage pool*, and *active-data pool*.

storage pool volume
A volume that has been assigned to a storage pool. See also *volume*, *active-data pool*, *copy storage pool*, and *primary storage pool*.

storage privilege class
A privilege class that gives an administrator the authority to control how storage resources for the server are allocated and used, such as monitoring the database, the recovery log, and server storage. See also *privilege class*.

stub A shortcut on the Windows file system that is generated by the hierarchical storage management (HSM) client for a migrated file that allows transparent user access. A stub is the sparse file representation of a migrated file, with a reparse point attached.

stub file
A file that replaces the original file on a local file system when the file is migrated to storage. A stub file contains the information that is necessary to recall a migrated file from Tivoli Storage Manager storage. It also contains additional information that can be used to eliminate the need to recall a migrated file.

stub file size
The size of a file that replaces the original file on a local file system when the file is migrated to Tivoli Storage Manager storage. The size that is specified for stub files determines how much leader data can be stored in the stub file. The default for stub file size is the block size defined for a file system minus 1 byte.

subscription
In a Tivoli environment, the process of identifying the subscribers that the profiles are distributed to. For Tivoli Storage Manager, a subscription is the process by which a managed server receives configuration information associated with a particular profile on a configuration manager. See also *managed server*, *configuration manager*, and *profile*.

Systems Network Architecture (SNA)
The description of the logical structure, formats, protocols, and operational sequences for transmitting information through and controlling the configuration and operation of networks.

system privilege class
A privilege class that gives an administrator the authority to issue all server commands. See also *privilege class*.

T

tape library
A set of equipment and facilities that support an installation's tape environment. The tape library can include tape storage racks, mechanisms for automatic tape mounting, a set of tape drives, and a set of related tape volumes mounted on those drives.

tape volume prefix
The high-level-qualifier of the file name or the data set name in the standard tape label.

target node
A client node for which other client nodes (called agent nodes) have been granted proxy authority. The proxy authority allows the agent nodes to perform operations such as backup and restore on behalf of the target node, which owns the data.

TCA See *trusted communications agent*.

TCP/IP
See *Transmission Control Protocol/Internet Protocol*.

threshold migration
The process of moving files from a local file system to Tivoli Storage Manager storage based on the high and low thresholds that are defined for the file system. Contrast with *demand migration*, *selective migration*, and *migration job*.

throughput
In storage management, the total bytes in the workload, excluding overhead, that are backed up or restored, divided by elapsed time.

timeout
A time interval that is allotted for an event to occur or complete before operation is interrupted.

timestamp control mode
A mode that determines whether commands preserve the access time for a file or set it to the current time.

Tivoli Storage Manager command script
A sequence of Tivoli Storage Manager administrative commands that are stored in the database of the Tivoli Storage Manager server. The script can run from any interface to the server. The script can include substitution for command parameters and conditional logic.

Transmission Control Protocol/Internet Protocol (TCP/IP)
An industry-standard, nonproprietary set of communication protocols that provides reliable end-to-end connections between applications over interconnected networks of different types.

transparent recall
The process that is used to automatically recall a file to a workstation or file server when the file is accessed. See also *recall mode*. Contrast with *selective recall*.

trusted communications agent (TCA)
A program that handles the sign-on password protocol when clients use password generation.

U

UCS-2 A 2-byte (16-bit) encoding scheme based on ISO/IEC specification 10646-1. UCS-2 defines three levels of implementation: Level 1-No combining of encoded elements allowed; Level 2-Combining of encoded elements is allowed only for Thai, Indic, Hebrew, and Arabic; Level 3-Any combination of encoded elements are allowed.

UNC See *Universal Naming Convention name*.

Unicode
A character encoding standard that supports the interchange, processing, and display of text that is written in the common languages around the world, plus some classical and historical texts. The Unicode standard has a 16-bit character set defined by ISO 10646.

Unicode-enabled file space
Unicode file space names provide support for multilingual workstations without regard for the current locale.

Unicode transformation format 8
Unicode Transformation Format (UTF), 8-bit encoding form, which is designed for ease of use with existing ASCII-based systems. The CCSID value for data in UTF-8 format is 1208.

Universal Naming Convention (UNC) name
A name that is used to access a drive or directory containing files shared across a network. The UNC name includes the system name and a SharePoint name that represents the shared drive or directory.

Universally Unique Identifier (UUID)
The 128-bit numerical identifier that is used to ensure that two components do not have the same identifier.

UTF-8 See *Unicode transformation format 8*.

UUID See *Universally Unique Identifier*.

V

validate
To check a policy set for conditions that can cause problems if that policy set becomes the active policy set. For example, the validation process checks whether the policy set contains a default management class.

version
A backup copy of a file stored in server storage. The most recent backup copy of a file is the active version. Earlier copies of the same file are inactive versions. The number of versions retained by the server is determined by the copy group attributes in the management class.

virtual file space
A representation of a directory on a network-attached storage (NAS) file system as a path to that directory.

virtual volume
An archive file on a target server that represents a sequential media volume to a source server.

volume
A discrete unit of storage on disk, tape or other data recording medium that supports some form of identifier and parameter list, such as a volume label or input/output control. See also *scratch volume*, and *storage pool volume*.

volume history file
A file that contains information about volumes that have been used by the server for database backups and for export of administrator, node, policy, or server data. The file also has information about sequential-access storage pool volumes that have been added, reused, or deleted. The information is a copy of volume information that is recorded in the server database.

Volume Shadow Copy Service
A set of Microsoft application-programming interfaces (APIs) that you can use to create shadow copy backups of volumes, exact copies of files, including all open files, and so on.

VSS See *Volume Shadow Copy Service*.

VSS Backup
A backup operation that uses Microsoft Volume Shadow Copy Service (VSS) technology. The backup operation produces an online snapshot (point-in-time consistent copy) of Exchange data. This copy can be stored on local shadow volumes or on Tivoli Storage Manager server storage.

VSS Fast Restore
A function that uses a Microsoft Volume Shadow Copy Service (VSS) software provider to restore VSS Backups (IBM Data Protection for Exchange database files and log files) that reside on local shadow volumes.

VSS Instant Restore
A volume-level hardware-assisted Microsoft Volume Shadow Copy Service (VSS) function where target volumes that contain the snapshot are copied back to the original source volumes.

VSS offloaded backup
A backup operation that uses a Microsoft Volume Shadow Copy Service (VSS) hardware provider (installed on an alternate system) to move IBM Data Protection for Exchange data to the Tivoli Storage Manager server. This type of backup operation shifts the backup load from the production system to another system.

VSS Restore
A function that uses a Microsoft Volume Shadow Copy Service (VSS) software provider to restore VSS Backups (IBM Data Protection for Exchange database files and log files) that reside on Tivoli Storage Manager server storage to their original location.

W

wildcard character
A special character such as an asterisk (*) or a question mark (?) that can be used to represent one or more characters. Any character or set of characters can replace the wildcard character.

workstation
A configuration of input/output equipment at which an operator works. A workstation is a terminal or microcomputer at which a user can run applications and that is usually connected to a mainframe or a network.

worldwide name
A 64-bit, unsigned name identifier that is unique.

workload partition (WPAR)
A partition within a single operating system instance.

Index

Program Number: 5608-E02, 5608-E03, 5608-E04, 5608-E05, 5608-E06, 5608-E07,
5608-E10, 5608-E11, 5608-E12, 5608-ILD, 5698-B19, 5698-B21,
5698-B22

Printed in USA

SC23-9793-00