IBM

ADSTAR Distributed Storage Manager

SH26-4002-00

Using the Application Program Interface

Version 2

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page vii.

This book is also available in a softcopy form that can be viewed with the IBM BookManager READ licensed program.

First Edition (November 1996)

This edition applies to Version 2 Release 1 of the ADSTAR Distributed Storage Manager, 5648-020, 5622-112, 5697-078, 5763-SV1, 5733-197, 5686-073, 5655-119, 5765-564, 28H2250, 28H2180, 89G1342, and to any subsequent releases until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality.

You can send us comments about this book electronically:

- IBMLink from U.S.: STARPUBS at SJSVM28
- IBMLink from Canada: STARPUBS at TORIBM
- IBM Mail Exchange: USIB3VVD at IBMMAIL
- Internet: starpubs@sjsvm28.vnet.ibm.com (or starpubs at sjsvm28.vnet.ibm.com)
- Fax (U.S.): 1-800-426-6209

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Notices

Program Interface Information

This reference manual is intended to help the customer add ADSM application programming interface calls to an existing application. This manual documents General-Use Program Interface information provided by ADSTAR Distributed Storage Manager.

General-Use Program Interfaces allow the user to write programs that obtain the services of ADSM.

Trademarks

The following terms, denoted by a double asterisk (**) in this publication, are trademarks of other companies:

AFS	Transarc Corporation
Apple	Apple Computer, Inc.
Attachmate	Attachmate Corp.
Borland	Borland International, Inc.
CompuServe	CompuServe, Inc.
DECstation	Digital Equipment Corp.
DynaText	Electronic Book Technologies, Inc.
EWS-UX/V	NEC Corporation
Extra!	Attachmate Corp.
Hewlett-Packard	Hewlett-Packard Company
HP-UX	Hewlett-Packard Company
Intel	Intel Corp.
IPX/SPX	Novell, Inc.
IRIX	Silicon Graphics, Inc.
Lotus	Lotus Development Corporation
Lotus Notes	Lotus Development Corporation
Macintosh	Apple Computer, Inc.
MacTCP	Apple Computer, Inc.
Microsoft	Microsoft Corp.
Motif	Open Software Foundation, Inc.
NDS	Novell, Inc.
NetWare	Novell, Inc.
NetWare Directory Services	Novell, Inc.
NetWare Loadable Module	Novell, Inc.
NFS	Sun Microsystems, Inc.
NLM	Novell, Inc.
Novell	Novell, Inc.
Open Desktop	The Santa Cruz Operation, Inc.
OpenWindows	Sun Microsystems, Inc.
PC/TCP	FTP Software, Inc.
SCO	The Santa Cruz Operation, Inc.
SINIX	Siemens Nixdorf Information Systems, Inc.
Solaris	Sun Microsystems, Inc.
SPARC	SPARC International, Inc.
Sun	Sun Microsystems, Inc.
Sun Microsystems	Sun Microsystems, Inc.
SunOS	Sun Microsystems, Inc.
Sun-3	Sun Microsystems, Inc.
Sun-4	Sun Microsystems, Inc.
ULTRIX	Digital Equipment Corp.
WATCOM	WATCOM Systems, Inc.
Windows	Microsoft Corp.
Windows NT	Microsoft Corp.
X Windows	Massachusetts Institute of Technology
X/Open	X/Open Company Limited

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Preface

ADSTAR Distributed Storage Manager (ADSM) is a client/server program product that provides storage management services to customers in a multivendor computer environment. It includes the following programs:

- A *server program* that allows systems to act as a backup and archive server and a migration server for distributed workstations and file servers. The server program provides hierarchical storage management.
- An *administrative client program* that allows an ADSM administrator to control and monitor server activities, define storage management policies for backup, archive, and space management services, and set up schedules to perform those services at regular intervals.
- A *backup-archive client program* that allows users to back up and archive files from their workstations or file servers to ADSM storage, and restore and retrieve backup versions of files and archived copies of files to their local file systems.
- A *hierarchical storage management (HSM) client program* that provides space management services. The HSM client program migrates eligible files to ADSM storage to maintain specific levels of free space on local file systems, and automatically recalls migrated files when they are accessed. It also allows users to migrate and recall specific files. This client is not available on all ADSM-supported platforms.
- An *application program interface (API)* that allows you to enhance an existing application with storage management services. When an application is registered as a client node with an ADSM server, users can use the application to back up and archive objects, such as databases, to ADSM storage, and restore and retrieve objects from ADSM storage.

This publication documents an application program interface. It is intended for application programmers who want to develop an application that uses the ADSM storage management functions.

What You Should Know Before Reading This Publication

This publication assumes the following:

- You are familiar with the C Programming language
- You understand the ADSM storage management functions

Style Conventions

Throughout the book, we use the following style conventions in the text:

Figure 1. Style Conventions

Convention	Indicates
bold	An API function call or a file name.
UPPER CASE	An ADSM command or option.
italics	A new term or a reference to another publication.
`monospace text`	A data structure or type, or an example in the text.

Referenced Publications

The publications referenced in this book are listed in Figure 2.

Figure 2. Referenced Publications

Short Title	Publication Title	Order Number
Installing the Clients	*ADSTAR Distributed Storage Manager Installing the Clients*	SH26-4049
Using the UNIX Backup-Archive Clients	*ADSTAR Distributed Storage Manager Using the UNIX Backup-Archive Clients*	SH26-4052
ADSM Messages	*ADSTAR Distributed Storage Manager Messages*	SH35-0133

All of the ADSM publications are available in online readable format on the *ADSM Online Product Library* CD-ROM, order number SK2T-1878. The ADSM library is also available in softcopy on the following CD-ROMs:

AIX Base Collection Kit	(order number SK2T-2066)
AS/400 Base Collection Kit	(order number SK2T-2171)
MVS Base Collection Kit	(order number SK2T-0710)
OS/2 Base Collection Kit	(order number SK2T-2176)
VM Base Collection Kit	(order number SK2T-2067)
VSE Base Collection Kit	(order number SK2T-0060)
IBM SystemView for AIX	(order number SK2T-1451)

In addition, the following book is published by the X/Open Company. It contains the detailed specification on the X/Open Backup Services API. You should refer to it for information on the XBSA functions and data types. You can also contact X/Open at **xospecs@xopen.org**.

Figure 3. X/Open Publications. These are available from X/Open Company.

Short Title	Publication Title	Order Number
X/Open Specification	*X/Open Preliminary Specification: Systems Management: Backup Services API (XBSA)*	P424

Summary of Changes

Major technical changes in this book are indicated by a vertical bar (|) in the margin.

Previous versions of this book were distributed only in ASCII format on CompuServe. This is the first version to be available in hardcopy form.

Version 2, Release 1, June 1995

These are the changes made for ADSM Version 2 Release 1.

Functional Enhancements

- The ADSM API is now available on the following platforms:
  - AT&T UNIX
  - NEC EWS-UX/V
  - OpenEdition MVS
  - SGI IRIX
  - Siemens Nixdorf SINIX
- A new API function, **dsmRCMsg**, has been added.
- Support for *partial object retrieval* has been added. This allows the user to retrieve a portion of an object instead of the entire object. Partial object retrieval is only valid for archived copies.
- The API now uses many of the same files as the backup-archive clients, such as the message text file and the **options.doc** file. Also, the API uses the same installation directory as the backup-archive clients for files referenced by DSMI_DIR.

Changes to this Book

- An appendix describing the return codes in more detail has been added. This appendix was formerly available on CompuServe as a separate book.

Version 2, Release 1, March 1996

These are additional changes made for ADSM Version 2 Release 1.

Functional Enhancements

- A version of the ADSM API that implements the X/Open Backup Services API standard is now available on the following platforms:
  - AIX 3.2.5 and 4.1
  - Solaris 2.3 and 2.4

Changes to this Book

- Information on the ADSM X/Open API has been added as an appendix.

Version 2, Release 1, September 1996

These are additional changes made for ADSM Version 2 Release 1.

Functional Enhancements

- The ADSM API is now available on the following platforms:
  - AS/400

Chapter 1. Introduction to the ADSM API

The ADSTAR Distributed Storage Manager application program interface (API) enables an application client to use the ADSM storage management functions. The API consists of a set of 24 function calls that an application can use to perform the following operations:

- Initialize and terminate an ADSM session
- Assign management classes to objects before storing them on an ADSM server
- Back up or archive objects to an ADSM server
- Restore or retrieve objects from an ADSM server
- Query the server for information on objects stored there
- Manage file spaces

When you, as an application developer, install the ADSM API, you get the following:

- The files that an end user of an application would need:
  - The API shared library
  - The Trusted Agent program
  - Sample client options files
  - Documentation
- The source code for the three API header files that your application needs
- The source code for a sample application and the makefile to build it

For information on installing the API, see *Installing the Clients*.

Getting Information and Support

The IBM Storage Systems Division (SSD) Software Developer's Program provides a range of services to software developers who want to use the ADSM API. Information about the SSD Software Developer's Program is available in:

- IBMSTORAGE forum on CompuServe
- SSD Software Developer's Program Information Package

To obtain the Software Developer's Program Information Package:

1. Call 800-4-IBMSSD (800-442-6773). Outside the U.S.A., call 408-256-0000.
2. Listen for the SSD Software Developer's Program prompt.
3. Request the Software Developer's Program Information Package.

IBM has two programs—*standard* and *premier*—to provide you with technical support and notification of updates.

| See the **register.frm** file for details and application forms. On the AS/400 platform, this
| file is **QANSAPI/QAANSDOC(REGISTER)**. You receive this file when you install the API.

Configuration Files and Options Files

Configuration files and options files allow you to set the conditions and boundaries under which your ADSM session runs. The available options can be set by the ADSM administrator, the end user, or you. The values of various options allow you to do the following:

- Set up the connection to an ADSM server
- Control which objects are sent to the server and what management class they are associated with
- Set the format in which various object attributes appear, such as the date and time

The same option can appear in more than one configuration file. When this happens, the file with the highest priority takes precedence.

The different configuration files, in order of decreasing priority, are as follows:

1. *Administrator options*. Options set by an ADSM administrator, whether on the client or the server, override any options set by you or the end user.

   For example, the administrator can specify whether or not objects can be compressed before being sent to an ADSM server. In this case, setting the COMPRESSION option in the API configuration file has no effect. The administrator can also decide that the choice of allowing compression is to be determined by the client. Setting the COMPRESSION option in the API configuration file then determines whether objects are compressed before being stored.

2. The *API options list* takes effect when it is passed to a **dsmInit** call as a parameter. The list can contain user options such as:

   NODENAME
   SERVERNAME (UNIX only)
   TCPServeraddr (non-UNIX)

   The purpose of the API options list is to allow an application client to make changes to the values of the options in the API configuration file and the client user options file. For example, your application might query the end user for the user's preferred format for displaying dates and times. On the basis of the user's answers, you could construct an API options list with these two options and pass it into the call to **dsmInit**.

   For information on the format of the API options list, see the `options` parameter on page 77.

   You can also set this parameter to NULL, indicating there is no API options list for this ADSM session.

3. On the UNIX platform, the *API configuration file* can include any of the options in the user options file and the following options from the system options file:

   COMPRESSION
   DIRMC
   INCLEXCL

| On the Windows, OS/2, NetWare, and AS/400 platforms, the API configuration file
| can contain the user options.

You set up the options in the API configuration file to have values that you think will be appropriate in the end user's ADSM session. The values take effect when the API configuration file is passed to the new ADSM session as a parameter in the **dsmInit** call.

You can also set this parameter to NULL, indicating there is no API configuration file for this ADSM session.

4. The ADSM options files on the UNIX platform include the *user options file* (**dsm.opt**) and the *system options file* (**dsm.sys**). On the Windows, OS/2, and
| NetWare platforms, the options file consists of **dsm.opt** only. On the AS/400 plat-
| form, the options file consists of ***LIBL/QOPTADSM(APIOPT)** only. These files are set up by the end user when the ADSM API is first installed on the user's workstation. The options in these files can be overridden by the methods mentioned above.

For more detailed information on the options available, see either the *Installing the Clients* book or the separate User's Guide for your platform.

Chapter 2. Using the API Sample Application

| The API package that you receive includes a sample application. This sample application demonstrates the use of the API function calls in context. You should install the sample application and look at its source code to better understand how the function calls can be used.

A function named **rcApiOut** is included in the sample application. This function calls the API function **dsmRCMsg**, which translates a return code into a message. **rcApiOut** then prints the message.

Building the Sample Application

When you build the sample application, the procedure to follow depends on whether
| you are running the ADSM API on a Windows, Windows 32-bit (Windows 95 and
| Windows NT), OS/2, UNIX, NetWare, or AS/400 platform.

Windows platform

For the Windows platform, the files listed in Figure 4 are available to build either of the
| sample applications included in the ADSM API package for Windows.

Figure 4 (Page 1 of 2). Files Available for Building Windows API Sample Applications

File Name	Description
dapismp.prj	Project file used by the Borland C++ IDE to build the dapismp executable
dapismp.mak	Makefile for dapismp executable
dapismp.cfg	Configuration file referred to by dapismp.mak
dapismp.def	Definition file for sample application dapismp.exe
dapibkup.c dapidata.h dapiinit.c dapint64.c dapint64.h dapipref.c dapiproc.c dapipw.c dapiqry.c dapirc.c dapismp.c dapitype.h dapiutil.c dapiutil.h	The components of a general non-Windows application that demonstrates the use of the major ADSM API functions
apidemo.prj	Project file used by the Borland C++ IDE to build the apidemo executable
apidemo.mak	Makefile for apidemo executable

Figure 4 (Page 2 of 2). Files Available for Building Windows API Sample Applications

File Name	Description
apidemo.cfg	Configuration file referred to by apidemo.mak
apidemo.def	Definition file for apidemo.exe
apidemo.c apidemo.h apidemo.rc apidemo.res	A sample Windows application source module Header file for apidemo.exe .RC file for apidemo.exe .RES file for apidemo.exe

All of the build methods implemented by the above files use the Borland C++ compiler. Whether you use the makefile or the project file for the build, either the .mak file or the options in the project file may need to be adjusted to fit your environment. Specifically, the library and/or include directories may need to be changed. The DLL **adsm.dll** is a 16-bit DLL, and the makefile assumes use of the Borland C++ compiler.

Windows 32-Bit (Windows 95 and Windows NT) Platform

For the Windows 95 and Windows NT platforms, the files listed in Figure 5 make up the source files needed to build the sample application included in the ADSM API package for Windows 95 and Windows NT.

Figure 5 (Page 1 of 2). Files Available for Building Windows 95 and Windows NT API Sample Application

File Name	Description
dapismp	API sample program
blkhook.def blkhook.c blkhook.mak	Definition file for blocking hook single-threaded DLL Sample source code for blocking function Borland C++ 4.5 makefile for blocking function
adsm32.dll blkhook.dll	API DLLs
dsmrc.h dsmapitd.h dsmapifp.h dsmapidl.h dsmint64.h	Return codes header file Type definitions header file Function prototype header file Dynamically loaded function prototype header file Int 64 function prototype header file
dapidata.h dapint64.h dapitype.h dapiutil.h	Source code header files for dapismp.exe
adsm32.def	Definition file for single-threaded DLL

Figure 5 (Page 2 of 2). Files Available for Building Windows 95 and Windows NT API Sample Application

File Name	Description
dapibkup.c dapiinit.c dapint64.c dapipref.c dapiproc.c dapipw.c dapiqry.c dapirc.c dapismp.c dapiutil.c dynaload.c	Source code files for dapismp.exe
dapismp.mak dapismp.vc	Borland C++ 4.5 makefile for API sample program Visual C++ makefile for API sample program

In addition, the API\OBJ directory contains the API sample program object files, and the API\SAMPRUN directory is the sample program **dapismp** execution directory.

The source code (**blkhook.c**) and makefile (**blkhook.mak**) for a blocking function are included (along with the sample API program **dapismp**) for applications that use non-threaded GUIs and need to switch out of their message loops when blocked. The blocking hook DLL is called by the API routines in the **dapismp** sample program. The DLLs **adsm32.dll** and **blkhook.dll** are both 32-bit DLLs.

The API sample application **dapismp** may be compiled with the Borland C++ 4.5 compiler, using makefile **dapismp.mak**, or with the Microsoft Visual C++ 4.2 compiler, using makefile **dapismp.vc**. The makefiles may need to be adjusted to fit your environment. Specifically, the library and/or the include directories may need to be changed. To run the sample application, enter *dapismp* from the api\samprun directory.

OS/2 platform

For the OS/2 platform, the files listed in Figure 6 make up the source files and other files needed to build the sample application included in the ADSM API package for OS/2.

Figure 6 (Page 1 of 2). Files Available for Building OS/2 API Sample Application

File Name	Description
dapismp.mak	Makefile for dapismp.exe
dapismp.def	Definition file for sample application dapismp.exe

Figure 6 (Page 2 of 2). Files Available for Building OS/2 API Sample Application

File Name	Description
dapibkup.c dapidata.h dapiinit.c dapint64.c dapint64.h dapipref.c dapiproc.c dapipw.c dapiqry.c dapirc.c dapismp.c dapitype.h dapiutil.c dapiutil.h	The source code for dapismp.exe

The **dapismp.mak** file may need to be adjusted to fit your environment. Specifically, the library and/or include directories may need to be changed. The DLL **adsmos2.dll** is a 16-bit DLL, and the makefile assumes use of the Microsoft C 6.0 compiler.

UNIX platform

For the UNIX platform, the files listed in Figure 7 make up the source files and other files needed to build the sample application included with the ADSM API.

Figure 7 (Page 1 of 2). Files Available for Building UNIX API Sample Application

File Name	Description
dsmrc.h	Return codes header file
dsmapitd.h	Type definitions header file
dsmapifp.h	Function prototype header file
dapibkup.c dapidata.h dapiinit.c dapint64.h dapint64.c dapipref.c dapiproc.c dapipw.c dapiqry.c dapirc.c dapismp.c dapitype.h dapiutil.h dapiutil.c	Modules for the command line driven sample application

Figure 7 (Page 2 of 2). Files Available for Building UNIX API Sample Application

File Name	Description
makeapi.aix	Makefile to build dapismp for AIX
makeapi.hp	Makefile to build dapismp for HP-UX
makeapi.sun	Makefile to build dapismp for SunOS
makeapi.sol	Makefile to build dapismp for Solaris
makeapi.att	Makefile to build dapismp for AT&T
makeapi.nec	Makefile to build dapismp for NEC
makeapi.sgi	Makefile to build dapismp for SGI
makeapi.sinix	Makefile to build dapismp for SINIX
makeapi.oemvs	Makefile to build dapismp for OpenEdition MVS
caller1.c caller2.c	Simple example modules

Follow these steps to compile the sample application and test the installation. Note that several of the steps have slight variations, depending on the UNIX platform you are using.

In the following instructions, the source directory from which the files are copied may not be the same as the directory you are using on your workstation. If that is the case, substitute the directory that you set in the DSMI_DIR environment variable for the one in these instructions.

The target directory in these instructions has the name **/u/developer/testapi**. However, you can use any name for the target directory.

1 Copy the API library to the **/usr/lib** directory:

AIX:	`cp /usr/lpp/adsm/api/bin/libApiDS.a /usr/lib`
HP-UX:	`cp /usr/adsm/api/libApiDS.sl /usr/lib`
SunOS:	`cp /usr/adsm/api/libApiDS.so.1.0.sun41 /usr/lib/libApiDS.so.1.0`
Solaris:	`cp /usr/adsm/api/libApiDS.so.sol2 /usr/lib/libApiDS.so`
AT&T:	`cp /usr/adsm/api/libApiDS.so /usr/lib`
NEC:	`cp /usr/adsm/api/libApiDS.so /usr/lib`
SGI:	`cp /usr/adsm/api/libApiDS.so /usr/lib`
SINIX:	`cp /usr/adsm/api/libApiDS.so /usr/lib`
OpenEdition MVS:	`cp /usr/adsm/api/libApiDS.a /usr/lib`

For some of the platforms, instead of copying the API library, you can create a symbolic link to the file from the **/usr/lib** directory. First change to the **/usr/lib** directory with the command:

```
cd /usr/lib
```

Then enter the following command:

AIX:	`ln -s /usr/lpp/adsm/api/bin/libApiDS.a`
HP-UX:	`ln -s /usr/adsm/api/libApiDS.sl`
AT&T:	`ln -s /usr/adsm/api/libApiDS.so`

NEC:	`ln -s /usr/adsm/api/libApiDS.so`
SGI:	`ln -s /usr/adsm/api/libApiDS.so`
SINIX:	`ln -s /usr/adsm/api/libApiDS.so`
OpenEdition MVS:	`ln -s /usr/adsm/api/libApiDS.a`

2 Copy the sample application files to the target directory:

AIX:	`cp /usr/lpp/adsm/api/bin/dapi* /u/developer/testapi`
HP-UX:	`cp /usr/adsm/api/dapi* /u/developer/testapi`
SunOS:	`cp /usr/adsm/api/dapi* /u/developer/testapi`
Solaris:	`cp /usr/adsm/api/dapi* /u/developer/testapi`
AT&T:	`cp /usr/adsm/api/dapi* /home/developer/testapi`
NEC:	`cp /usr/adsm/api/dapi* /home/developer/testapi`
SGI:	`cp /usr/adsm/api/dapi* /home/developer/testapi`
SINIX:	`cp /usr/adsm/api/dapi* /home/developer/testapi`
OpenEdition MVS:	`cp /usr/adsm/api/dapi* /u/developer/testapi`

3 Copy the header files to the target directory:

AIX:	`cp /usr/lpp/adsm/api/bin/*.h /u/developer/testapi`
HP-UX:	`cp /usr/adsm/api/*.h /u/developer/testapi`
SunOS:	`cp /usr/adsm/api/*.h /u/developer/testapi`
Solaris:	`cp /usr/adsm/api/*.h /u/developer/testapi`
AT&T:	`cp /usr/adsm/api/*.h /home/developer/testapi`
NEC:	`cp /usr/adsm/api/*.h /home/developer/testapi`
SGI:	`cp /usr/adsm/api/*.h /home/developer/testapi`
SINIX:	`cp /usr/adsm/api/*.h /home/developer/testapi`
OpenEdition MVS:	`cp /usr/adsm/api/*.h /u/developer/testapi`

4 Copy the makefile to the target directory:

AIX:	`cp /usr/lpp/adsm/api/bin/makeapi.aix /u/developer/testapi`
HP-UX:	`cp /usr/adsm/api/makeapi.hp /u/developer/testapi`
SunOS:	`cp /usr/adsm/api/makeapi.sun /u/developer/testapi`
Solaris:	`cp /usr/adsm/api/makeapi.sol /u/developer/testapi`
AT&T:	`cp /usr/adsm/api/makeapi.att /home/developer/testapi`
NEC:	`cp /usr/adsm/api/makeapi.nec /home/developer/testapi`
SGI:	`cp /usr/adsm/api/makeapi.sgi /home/developer/testapi`
SINIX:	`cp /usr/adsm/api/makeapi.sinix /home/developer/testapi`
OpenEdition MVS:	`cp /usr/adsm/api/makeapi.oemvs /u/developer/testapi`

5 Compile the sample with the following command:

AIX:	`make -f makeapi.aix`
HP-UX:	`make -f makeapi.hp`
SunOS:	`make -f makeapi.sun`
Solaris:	`make -f makeapi.sol`

AT&T:	`make -f makeapi.att`
NEC:	`make -f makeapi.nec`
SGI:	`make -f makeapi.sgi`
SINIX:	`make -f makeapi.sinix`
OpenEdition MVS:	`make -f makeapi.oemvs`

6 Ensure that your environment variables, especially DSMI_DIR, and options files are set up. See “Configuration Files and Options Files” on page 2 and the *Installing the Clients* book for information.

7 Log on as root the first time for password registration.

8 Run **dapismp** to start the sample application.

NetWare platform

For the NetWare platform, the files listed in Figure 8 make up the source files and other files needed to build the sample application included with the ADSM API.

Figure 8. ADSM files in the Novell NetWare API package for developers

File Name	Description
dsmapifp.h	prototype sdk header file
dsmapitd.h	types sdk header file
dsmrc.h	return code sdk header file
dapibkup.c dapidata.h dapiinit.c dapipref.c dapiproc.c dapipw.c dapiqry.c dapirc.c dapismp.c dapismp.lnk dapismp.wat dapismp.nlm dapitype.h dapinwut.c dapiutil.c dapiutil.h	Source code for a sample application that demonstrates the use of the major ADSM API functions

Follow these steps to compile the sample application and test the installation:

1 Once your makefile and WATCOM compiler are ready, type:

```
wmake /f dapismp.wat
```

You will see compiler messages on the screen until the compile is completed.

2 Copy **dapismp.nlm** to the NetWare server.

3 Before running the sample application, **dapismp.nlm**, make sure that you have the following files on your NetWare server:

dsm.smp
dscameng.txt
dsmapi.nlm
acpwsrvs.nlm
acpwtcps.nlm
acpwsaas.nlm
dapismp.nlm

4 Copy **dsm.smp** to **dsm.opt**.

5 Edit **dsm.opt** and enter values for the COMMMETHOD and NODENAME options. Also enter values for the options relating to the specific communication method that you are using.

6 At the NetWare console enter the following:

```
search add sys:\adsm\api
```

7 To start the sample application, run **dapismp** by entering:

```
load dapismp
```

|
AS/400 platform

|
| For the AS/400 platform, the files listed in Figure 9 make up the source files and other files needed to build the sample application included with the ADSM API.

| *Figure 9 (Page 1 of 2). ADSM files in the AS/400 API package for developers*

File Name	**Description**
CRTAPISMP	Source for a CL Program to create the Sample Application
dsmrc.h	Return codes header file
dsmapitd.h	Type definitions header file
dsmapifp.h	Function prototype header file

Figure 9 (Page 2 of 2). ADSM files in the AS/400 API package for developers

File Name	**Description**
dapibkup.c dapidata.h dapiinit.c dapint64.h dapint64.c dapipref.c dapiproc.c dapipw.c dapiqry.c dapirc.c dapismp.c dapitype.h dapiutil.h dapiutil.c	Source code for a sample application that demonstrates the use of the major ADSM API functions

The **.c** files are members of file **QANSAPI/QCSRC**, and the **.h** files are members of file **QANSAPI/H.**. The CL source is contained in file **QANSAPI/QCLSRC**.

Follow these steps to compile the sample application and test the installation:

1 Compile the CL program QANSAPI/QCLSRC(CRTAPISMP):

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

2 Compile the source code for the Sample Application:

Issue the CRTCMOD command for each member in QANSAPI/QCSRC
For example:

```
CRTCMOD MODULE(QANSAPI/DAPISMP) SRCFILE(QANSAPI/QCSRC) OUTPUT(*PRINT)
```

3 Invoke CRTAPISMP to create the Sample Application and bind it to the API service program.

```
CALL CRTAPISMP ( target_library QANSAPI )
```

4 Ensure that **target_library** is in your library list.

5 Copy **QANSAPI/QAANSDOC(APIOPTSMP)** to **library/QOPTADSM(APIOPT)**, where **library** is in your library list.

6 Edit **library/QOPTADSM(APIOPT)** and enter values for the COMMMETHOD and NODENAME options. Also enter values for the options relating to the specific communication method that you are using.

| **7** To start the sample application, run **dapismp** by entering:

```
call dapismp
```

Running the Sample Application

After you start the sample application by entering **dapismp**, follow the instructions that appear on the screen. Some things to remember when running the application are:

- You must run the Signon action before any other action.
- When entering the File space, High-level, or Low-level names, 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.
- For OpenEdition MVS, the owner of **dapismp** must have root user authority, and the set-user-ID bit must be turned on if **dapismp** is run with PASSWORDACCESS=Generate.

  ```
  chown 0 dapismp
  chmod u+s dapismp
  ```

Chapter 3. Using the Application Program Interface

This section describes, in a task-oriented fashion, how to use the Application Program Interface (API). You should be familiar with this section prior to designing or writing an application that uses the ADSM API.

| The API package that you receive includes a sample application. This chapter demonstrates how the API can be used by showing examples excerpted from the sample application.

Remember that the examples in this section are just program fragments. Their purpose is to show how the API functions might be used in context, not to provide pieces of executable code.

Note: The examples that follow are taken from the UNIX sample application. The analogous program fragments on other platforms might look somewhat different.

Maintaining Version Control in the API

All APIs have some form of version control, and ADSM is no exception. You must be sure the version of the ADSM API that you use in your application is compatible with the version of the API library that the end users have installed on their workstations.

The first API call issued when using the ADSM API should usually be **dsmQueryApiVersion**. This call does the following for the application client:

- Confirms that the ADSM API library is installed and available on the end user's system
- Returns the version level of the API library being accessed by the application

The API is designed to be upward compatible, so that applications written to older versions or releases of the API library will still operate correctly if the end user is running a newer version.

Determining the release of the API library is critical, because some releases might have different memory requirements and data structure definitions. Downward compatibility might be possible on a case-by-case basis, but it is not a design goal to be so. Downward compatibility, if a requirement, is the responsibility of the application client.

The API library and the Trusted Communication Agent module (**dsmapitca**) must be at the same level.

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

The version number of the application client's API is hard-coded in the compiled object code as a set of three constants (see Appendix A, "API Type Definitions Source File" on page 91):

```
DSM_API_VERSION
DSM_API_RELEASE
DSM_API_LEVEL
```

The application client's API version should usually be less than or equal to the API library installed on the user's system. Any other condition should be entered into with care.

The **dsmQueryApiVersion** call can be issued at any time, whether the API session has been initialized or not.

Data structures used by the API also have version control information in them. Those structures that can grow in size have version information as the first field within the structure. This allows the ADSM API to verify the version of the structure being used.

Example

The example in Figure 10 on page 17 first shows the type definition of the structure `dsmApiVersion` from the header file **dsmapitd.h**. The example then defines a global variable called `apiLibVer` and shows how it can be used in a call to **dsmQueryApiVersion** to return the version of the end user's API library. Finally, the returned value is compared to the version number of the application client's API.

Starting and Terminating a Session

ADSM is a session-oriented product, and all activities must be performed within an ADSM session. To start a session, the application invokes the **dsmInit** call. This call must be performed prior to any other API call except **dsmQueryApiVersion**. The **dsmInit** function sets up a session with the ADSM server as indicated in the parameters passed in the call or defined in the options files.

Three parameters that must be passed in to **dsmInit** are the client's node name, owner name, and password. The owner name is case sensitive, but the node name and password are not.

The first time a unique API application client initializes a session with the ADSM server, the client's *application type* is registered with the server. This type can be used as a very high-level identifier to perform validity checking on the type of data the client has stored at the server. For example, if the application client is the *Wonder DB* backup solution on UNIX, the application type might be `UNIX_Wonder_DB`. After initialization is complete, the application can check to see if the application type is correct, and if not, issue an error or modify processing to handle the returned application type.

Note that the application client can only register new nodes with a server if the server has closed registration. If a server has open registration, then any nodes that are already registered with the server will be accepted by the application. However, if a server has open registration and **dsmInit** tries to register a new node, then the return code `DSM_RC_REJECT_ID_UNKNOWN` is generated. Application designers should tell their customers about this requirement so that customers can configure their servers accordingly.

```
typedef struct                   /* from dsmapitd.h */
{
        unsigned short version;       /* API version */
        unsigned short release;       /* API release */
        unsigned short level;         /* API level   */
} dsmApiVersion;

extern  dsmApiVersion                 apiLibVer;

memset(&apiLibVer,0x00,sizeof(dsmApiVersion));     /* Zero out block */
dsmQueryApiVersion(&apiLibVer);        /* Get the end user's version */
printf("\nAPI Library Version = %d.%d.%d\n\n",apiLibVer.version,
                                              apiLibVer.release,
                                              apiLibVer.level);
/* Compare with the application client's version */
if ( (apiLibVer.version != DSM_API_VERSION) ||
     (apiLibVer.release != DSM_API_RELEASE) ||
     (apiLibVer.level   != DSM_API_LEVEL  ))
  {  printf("Application Version = %d.%d.%d\n",DSM_API_VERSION,
                                               DSM_API_RELEASE,
                                               DSM_API_LEVEL);
     printf("Application version is different than end user's API version.\n\n");
  };
```

Figure 10. Example: Getting the Version Level of the API

Once the application type has been registered with the server, it persists with the node name until that node is deleted from the system. Future session initializations from the same node can set this parameter to NULL.

dsmInit also links the ADSM session with the application client's API configuration file and option list. The application client can use the API configuration file and option list to set a number of ADSM options. These values override the values set in the user's configuration files at installation time, although they cannot change the options defined by the ADSM administrator. If the application client does not have its own configuration file and option list, both of these parameters can be set to NULL. (For more information on configuration files, see “Configuration Files and Options Files” on page 2.)

Once a session is initialized, the application can issue a call to **dsmQuerySessInfo** to determine the parameters set for this session. Items such as the policy domain and transaction limits are all returned to the application with this call.

Sessions are terminated by a **dsmTerminate** call. This causes the API to close any connection with the ADSM server and free all resources associated with this session.

Note: Only one session can be active per invocation of the API. However, applications on UNIX and OS/2 platforms can circumvent this restriction by running with multiple processes, each process owning its own ADSM session. Applications on the AS/400 platform can circumvent this restriction by running with multiple jobs, each job owning its own ADSM session. The Windows and NetWare platforms do not have this option.

Application Design Considerations

On the UNIX platform, if the end user has PASSWORDACCESS=Generate, then the Trusted Communication Agent (**dsmapitca**) child process is forked to manage the session with the ADSM server. The SIGCLD signal is used during termination. If PASSWORDACCESS=Prompt, no child process is used.

On OpenEdition MVS, if the end user has PASSWORDACCESS=Generate, then the API calls have to be executed with the effective userid set to 0, because there is no separate Trusted Communication Agent process on this platform. This can be achieved by running the whole program with root user authority:

```
chown 0 dapismp
chmod u+s dapismp
```

For better security control, each API call can be surrounded with seteuid(0) and seteuid(getuid()), thus running only API calls with root user authority. If there is no need for automatic session reconnection on timeouts, this special handling may be applied only to the **dsmInit** call.

If PASSWORDACCESS=Prompt, there is no need for such a mechanism, but the code should handle the case where the call to **dsmInit** fails because the user's password is expired.

Session Security

ADSM, being a session-based system, has security components that allow applications to initialize 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 has to go through a sign-on process. This sign-on process requires the use of 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 ADSM API for session initialization.

Passwords have expiration periods associated with them. Thus, if a **dsmInit** call fails with a password expired return code, the application client must issue the **dsmChangePW** call to update the password. If a password has expired, it must be updated before the session can be successfully established.

Objects stored in the system also have ownerships associated with them. See the section “Identifying the Object” on page 41 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 initialized.

Examples

The example in Figure 11 on page 19 defines a number of global and local variables and then uses them in calls to **dsmInit** and **dsmTerminate**. Note that **dsmInit** takes a pointer to `dsmHandle` for one of its parameters, while **dsmTerminate** takes the `dsmHandle` itself.

The function **rcApiOut** calls the API function **dsmRCMsg**, which translates a return code into a message. **rcApiOut** then prints the message for the user. A version of **rcApiOut** is included in the API sample application. `dsmApiVersion` and `ApiSessInfo` are type definitions found in the header file **dsmapitd.h**.

```
/*** Example 1 ***/
extern  uint32             dsmHandle;
extern  dsmApiVersion      apiVer;      /* application's API version */
extern  ApiSessInfo        dsmSessInfo;
char    *node;
char    *pw;               /* Must allocate space                  */
char    *owner;            /*     for these pointer variables       */
int16   rc;

apiVer.version = DSM_API_VERSION;     /* Set the application's API    */
apiVer.release = DSM_API_RELEASE;     /*   version from the constants */
apiVer.level   = DSM_API_LEVEL;       /*   in dsmapitd.h              */

rc = dsmInit(&dsmHandle,    /* Will contain session handle on return. */
             &apiVer,       /* Application's API version.            */
             node,owner,pw, /* Node, owner, & password               */
             NULL,          /* Name of our node type.                */
             NULL,NULL);    /* Use default configs and options.      */

if (rc)
{
   printf("*** Init failed: ");
   rcApiOut(rc);            /* Call function to print error message   */
}

rc = dsmTerminate(dsmHandle);
dsmHandle = 0;
memset(&dsmSessInfo,0x00,sizeof(ApiSessInfo));    /* Zero out block.  */
```

Figure 11. Example: Beginning and Ending a Session

The example in Figure 12 on page 20 demonstrates the procedure to change a password using **dsmChangePW**.

```
/*** Example 2 ***/
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, then try again or exit. */
/* If they do match, call dsmChangePW.                          */

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

Figure 12. Example: Changing a Password

Associating a Management Class With Objects

One of the key features offered by ADSM is the use of policy (management classes) to define how objects are stored and managed in ADSM storage. A *management class* is associated with an object when the object is backed up or archived. This management class determines the following:

- How frequently objects are backed up
- How many versions of the object are retained if backed up
- How long to keep archive copies
- Where the object is to be inserted in the storage hierarchy on the server

Management classes have two components:

- Backup copy groups
- 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. Thus, 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.

Note that 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.

Because the use of policy is a critical component of ADSM, the API requires that all objects sent to the server first be assigned a management class via the **dsmBindMC** call. ADSM supports the use of an *include-exclude list* to perform management class binding. **dsmBindMC** uses the current include-exclude list to perform the management class binding.

The API requires that **dsmBindMC** be called prior to backing up or archiving an object. **dsmBindMC** returns an `mcBindKey` structure that contains information on the management class and copy groups associated with the object. A valid backup or archive copy group destination is the key information required to determine whether the current object can be sent as part of a current multiple object transaction or if a new transaction must be started before sending the object to a server. The information returned by **dsmBindMC** is:

management class
This is the name of the management class that has been bound to the object. If desired, the application client can issue **dsmBeginQuery** to determine all attributes of this management class.

backup copy group
The information returned is whether 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 ADSM storage. An attempt to send it via **dsmSendObj** results in an error code being returned.

backup copy destination
This field identifies the ADSM storage pool to which the data is sent. If you are performing a multiple object backup transaction, then all copy destinations within that transaction must be the same. If an object has a different copy destination than previous objects in the transaction, the current transaction must be ended and a new transaction begun before the object can be sent. Attempts to send objects to different copy destinations within the same transaction result in an error code being returned.

archive copy group
The information returned is whether 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 ADSM storage. An attempt to send it via **dsmSendObj** results in an error code being returned.

archive copy destination
This field identifies the ADSM storage pool to which the data is sent. If you are performing a multiple object archive transaction, then all copy destinations within that transaction must be the same. If an object has a different copy destination than previous objects in the transaction, the current transaction must be ended and a new transaction begun before the object can be sent. Attempts to send objects to different copy destinations within the same transaction result in an error code being returned.

Querying Management Classes

Applications can query management classes to see what management classes are possible for a given node, and to see what the attributes are within the management class. Objects can only be bound to management classes via the **dsmBindMC** call, but applications might wish to query the management class attributes for displaying them to end users or for other purposes. See the section “Querying the ADSM System” for details.

Example

In the example in Figure 13, 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 structure `MCBindKey`.

```
uint16     send_type;
uint32     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(rc);
   rc = (RC_SESSION_FAILED);
   return;
}
```

Figure 13. Example: Associating a Management Class With an Object

Querying the ADSM System

The ADSM API has a variety of queries that applications can use. The management class query, for example, has already been mentioned. All queries to the system operate in the same manner for consistency and ease of implementation.

All queries that use **dsmBeginQuery** follow the same steps described below:

1 Issue the **dsmBeginQuery** call with the proper query type. Query types are:

Backup
Archive
Active backed up objects
File space
Management class

The **dsmBeginQuery** call tells the API in what format the data is coming back from the server, so that the proper fields can be placed in the data structures passed by the **dsmGetNextQObj** calls. The begin query call also allows the application client to set the scope of the query to be performed by properly specifying the parameters on the begin query call.

Note that on multi-user platforms the query of active backed up objects (also known as fast path) can be done *only* by the root user.

2 Issue the **dsmGetNextQObj** call.

A **dsmGetNextQObj** call must be issued to obtain each record from the query issued. This call passes a buffer that is large enough to hold the data returned from the query. Each query type has a corresponding data structure for the data returned. For instance, a backup query type has an associated `qryRespBackupData` structure that is filled in when the **dsmGetNextQObj** call is issued.

3 Check the return code.

The **dsmGetNextQObj** call usually returns one of the following codes (an error code might also be returned):

DSM_RC_MORE_DATA
DSM_RC_FINISHED

In the first case, issue the **dsmGetNextQObj** call again. In the second case, there is no more data, so issue the **dsmEndQuery** call.

4 Issue the **dsmEndQuery** call.

When all query data has been retrieved or no further query data is desired, the **dsmEndQuery** call must be issued to terminate the query process. This causes the ADSM API to flush any remaining data from the query stream and release any resources utilized for the query.

State Diagrams and Flow Charts

The state diagram for performing query operations is shown in Figure 14 on page 24.

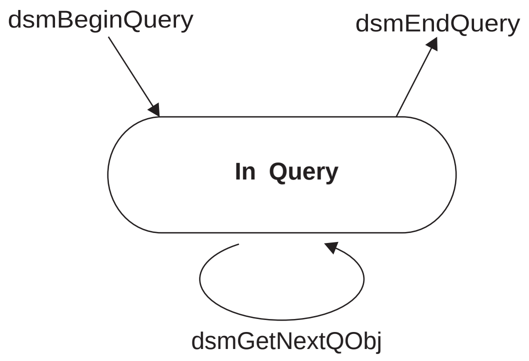

Figure 14. State Diagram for General Queries

The flow chart for performing query operations is shown in Figure 15.

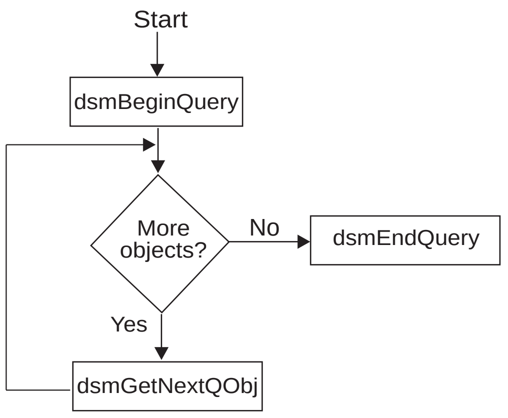

Figure 15. Flow Chart for General Queries

Example

In the example in Figure 16 on page 25, a management class query prints out the values of all the fields in the backup and archive copy groups for a particular management class.

```
int16                rc;
qryMCData            qMCData;
DataBlk              qData;
qryRespMCDetailData  qRespMCData, *mcResp;
char                 *mc, *s;
bool_t               done = bFalse;
uint32               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(rc);
   rc = (RC_SESSION_FAILED);
}
```

Figure 16 (Part 1 of 2). Example: Performing a System Query

```
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 %lu:\n",qry_item);
         printf("              Name: %s\n",mcResp->mcName);
         printf("    Backup CG Name: %s\n",mcResp->backupDet.cgName);
            .
            .   /* other fields of backup and archive copy groups */
            .
         printf("     Copy Destination: %s\n",mcResp->archDet.destName);
      }
      else
      {
         done = bTrue;
         if (rc != DSM_RC_FINISHED)
         {
            printf("*** dsmGetNextQObj failed: ");
            rcApiOut(rc);
         }
      }
      if (rc == DSM_RC_FINISHED) done = bTrue;
   }
   rc = dsmEndQuery(dsmHandle);
}
```

Figure 16 (Part 2 of 2). Example: Performing a System Query

Sending Data to a Server

The ADSM API allows application clients to send data, or named objects and their associated data, to ADSM server storage. Data can be either backed up or archived. Note that all send operations must be done within the bounds of a transaction.

The Transaction Model

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

A transaction is initiated by a call to **dsmBeginTxn** and is ended by a call to **dsmEndTxn**.

A single transaction is an atomic action. Data sent within the bounds of a transaction is either all committed to the system at the end of the transaction, or it is all rolled back if the transaction is ended prematurely.

Transactions can consist of either single object sends or multiple object sends. Smaller objects should be sent 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 the same copy destination. If you need to send an object to a different destination than the previous object, you must end the current transaction and start a new one. Within the new transaction, you can send the object to the new copy destination.

ADSM limits the number of objects that can be sent in a multiple object transaction. You can find this limit by calling **dsmQuerySessInfo** and examining the `maxObjPerTxn` field. This field shows the value of the TXNGroupmax option set on your server.

The application client must keep track of the objects sent within a transaction in order to perform retry processing or error processing if the transaction is ended prematurely. A transaction can be halted at any time by either the server or the client. Thus, the application client must be prepared to handle sudden transaction ends that it did not initiate.

Sending Objects to the Server

Application clients can send data, or named objects and their associated data, to ADSM storage using the API's backup and archive functions. The backup and archive components of the system allow for different management methodologies to be used for the data sent to ADSM storage.

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

Initiate a send operation by calling **dsmSendObj**. If you have more data than can be sent at one time, you can make repeated calls to **dsmSendData** to transfer the rest of the information. Call **dsmEndSendObj** to finish the send operation.

The *backup* component of the ADSM system supports multiple versions of named objects stored on the server. Thus, any object backed up to the server that has the same name as an object 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 are assigned to active and inactive objects on the server as defined by the management class constructs.

The *archive* component of the ADSM system allows objects to be stored 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 allows an application to archive the same object multiple times, but with different expiration times assigned to each copy of the object.

The end user's configuration, along with the `objCompressed` flag, determines whether ADSM will compress the object during a send. Also, objects with a `sizeEstimate` less than `DSM_MIN_COMPRESS_SIZE` will not be compressed.

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

```
COMPRESSAlways Yes
```

| **Note:** 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`.

State Diagrams and Flow Charts

The ADSM API is designed for straightforward logic flows and clear transitions between the various states of the application client. This clean state transition greatly enhances the quality and reliability of the system by catching logic flaws and program errors early in the development cycle. For instance, a **dsmSendObj** call cannot be made unless a transaction has been started and a **dsmBindMC** call was previously made for the object being backed up.

The state diagram for performing backup or archive operations within a transaction is shown in Figure 17 on page 29. The arrow on the right, going from the block marked “In Send Object” to **dsmEndTxn**, shows that a **dsmEndTxn** call can be issued after a call to **dsmSendObj** or **dsmSendData**. This might be desirable if an error condition occurred during the sending of an object, and you want to stop the whole operation. In normal circumstances, though, you call **dsmEndSendObj** before ending the transaction.

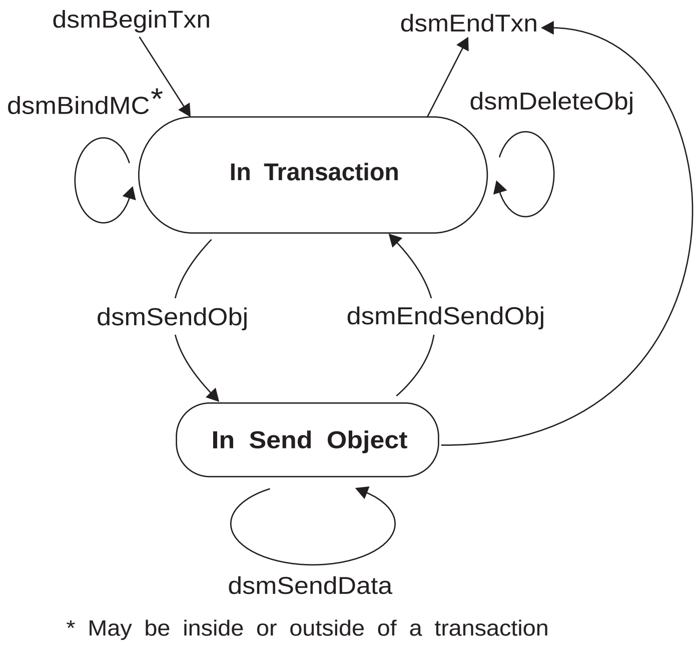

Figure 17. State Diagram for Backup and Archive Operations

The flow chart for performing backup or archive operations within a transaction is shown in Figure 18 on page 30.

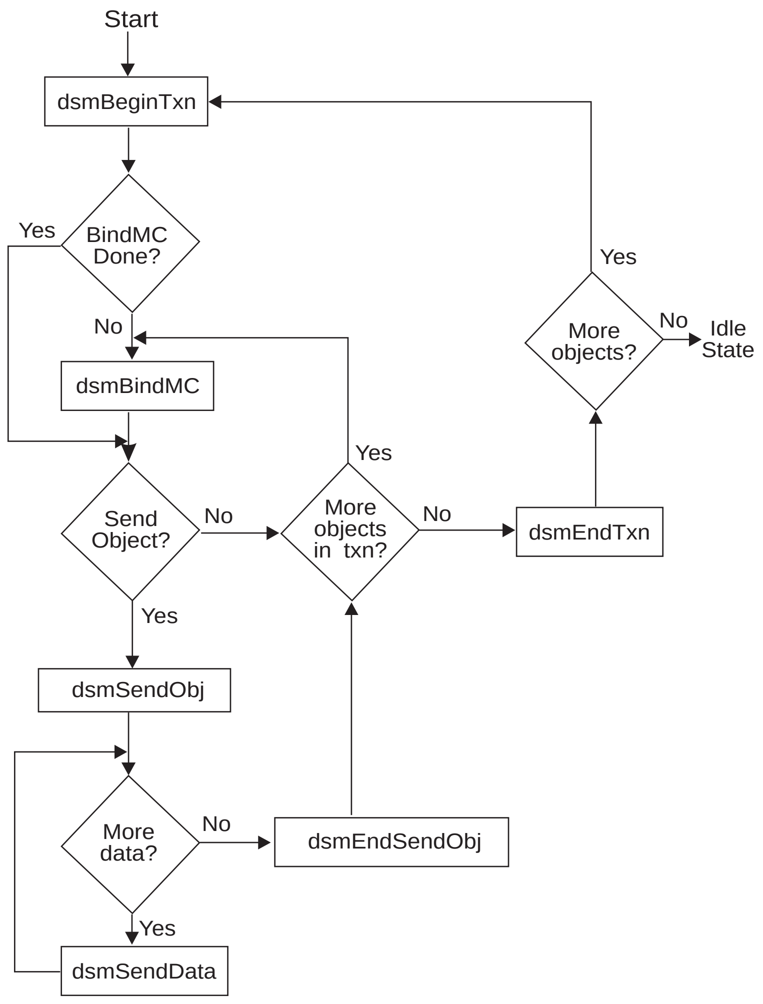

Figure 18. Flow Chart for Backup and Archive Operations

The key 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 it can be issued from inside or outside of a transaction boundary. It can also be issued from a different transaction if required. The only requirement for the **dsmBindMC** call is that it be made prior to backing up or archiving an object. If the object being backed up or archived is not associated with a management class, an error code is returned from **dsmSendObj**. In this situation, the transaction is halted by calling **dsmEndTxn** (this error condition is not shown in the flow chart).

The flow chart illustrates how an application would use multiple object transactions. It shows where decision points can be placed to determine if the object being sent fits within the bounds of a transaction or whether a new transaction must be started.

Example

The example in Figure 19 demonstrates the use of the API functions that send data to ADSM storage. **dsmSendObj** appears inside a `switch` statement, so that different parameters can be called depending on whether a backup or archive operation is being performed. **dsmSendData** is called from inside a loop that repeatedly sends data until a flag is set that allows the program execution to exit the loop. Note that the entire send operation is done from within the bounds of a 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.

An example showing the use of **dsmBindMC** is shown in Figure 13 on page 22.

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

/*  Call dsmBindMC if not done previously */
```

Figure 19 (Part 1 of 3). Example: Sending Data to a Server

```
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(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(rc);
      done = bTrue;
   }
   /* Adjust the dataBlk buffer for the next piece to send */
}
```

Figure 19 (Part 2 of 3). Example: Sending Data to a Server

```
rc = dsmEndSendObj(dsmHandle);
if (rc)
{
   printf("*** dsmEndSendObj failed: ");
   rcApiOut(rc);
}

if (in_txn)
{
   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(rc);
      printf("   reason = %u\n",txn_reason);
   }
}
```

Figure 19 (Part 3 of 3). Example: Sending Data to a Server

Receiving Data from a Server

The ADSM API allows application clients to receive data, or named objects and their associated data, from ADSM storage using the restore and retrieve functions of the product. *Restore* accesses objects that have previously been backed up, and *retrieve* accesses objects that have previously been archived.

Note: Only the API can restore or retrieve objects that have been backed up or archived with API calls.

Both restore and retrieve 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 keys that are used by ADSM to uniquely define the object on the server.

| ## Receiving Data: Partial Object Restore or Retrieve

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

| Partial object restore or retrieve is accomplished by associating the following two data fields with each object key:

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

These data fields are used on the **dsmBeginGetData** call. They determine what portion of the object is restored or retrieved, as follows:

- If both `offset` and `length` are zero, then the entire object is restored or retrieved from ADSM storage.
- If `offset` is greater than zero, but `length` is zero, then the object is restored or retrieved from the offset to the end.
- If `length` is greater than zero, then just the portion of the object from `offset` for the specified length is restored or retrieved.

Note: Partial object restore or retrieve works only with ADSM Version 2 servers. Also, 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`.

Receiving Data: Restore or Retrieve Procedure

After a query has been made, restore and retrieve operate in exactly the same way.

Once a session is established with the ADSM server, the typical procedure to restore or retrieve data is:

1 Query the ADSM server for either backup or archive data.

2 Determine the objects to be restored or retrieved from the server.

3 Sort the objects on the restore order field.

4 Issue the **dsmBeginGetData** call with the list of objects to be accessed.

5 Issue 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. Issue the **dsmEndGetObj** call after all data for an object has been obtained.

6 Issue the **dsmEndGetData** call after all data for all objects has been received or to terminate the receive operation.

This cycle can be repeated as many times as is necessary. A more detailed description of each step follows.

Step 1. Issuing Queries for Backup or Archive Data

Before any restore or retrieve operation can begin, you must first query the ADSM server to determine what objects can be received from storage. To issue the query, the application must fill in the proper parameter lists and structures for the **dsmBeginQuery** call. This includes the file space that the query is to examine and pattern-match entries for the high-level and low-level name fields. If the session was initialized with a NULL owner name, the owner field need not be specified, but if the session was initialized with an explicit owner name, then only objects that explicitly have that owner name associated with them are returned.

The query returns all information that was originally stored with the object, plus the following:

copyId
The `copyIdHi` and `copyIdLo` values provide an 8-byte number that uniquely identifies this object for this node in ADSM storage. Use this ID to request a specific object from storage for restore or retrieve processing.

restoreOrder
The `restoreOrderHi` and `restoreOrderLo` values provide a mechanism for receiving objects from ADSM storage in the most efficient manner possible. Sort the objects to be restored on this value to insure that tapes are mounted only once and are read from front to back.

mediaClass
The `mediaClass` value identifies the type of media on which the object is currently stored in the server. This field can have one of the following values:

`MEDIA_FIXED`	Local, online, fixed media such as hard disks
`MEDIA_LIBRARY`	Local media that can be mechanically mounted
`MEDIA_NETWORK`	Storage media accessible via a network server
`MEDIA_SHELF`	Media accessible only via human intervention
`MEDIA_OFFSITE`	Media stored in an off-site location
`MEDIA_UNAVAILABLE`	Media that are inaccessible for retrieval

The first value represents the fastest access to an object. Each successive value represents a class of media that involves increasing variation in the time required to retrieve an object.

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

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

Step 2. Selecting Objects to Receive

Once the backup or archive query has been performed, the application client must determine which objects, if any, are to be restored or retrieved.

Step 3. Sorting Objects by Restore Order

Once the objects to restore or retrieve are selected, they must be sorted in ascending order (low to hi) by the `restoreOrderHi` and `restoreOrderLo` fields. This sorting is critical to the performance of the restore operation. Sorting the objects on the `restoreOrder` fields means that the data is read from the server in the most efficient order. Thus, 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.

Step 4. Issuing the dsmBeginGetData Call

Once the objects to receive have been selected and sorted, they can be submitted to ADSM for restore or retrieve. The **dsmBeginGetData** call begins a restore or retrieve operation. Two key parameters must be filled out in these calls:

mountWait

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

Note that the application client can make an informed decision on this by checking the `mediaClass` field returned from the backup or archive query. Requesting objects to be skipped takes longer to process, because the system must indicate to the application client that the object was skipped due to being on offline media.

dsmGetObjListP

This parameter is a data structure that contains a list of all `objId`s that are to be restored or retrieved. If the objects are archive copies, then each `objId` is associated with a `partialObjData` structure that describes whether the entire `objId` or only a particular section of the object is to be retrieved.

Each `objId` is eight bytes long, 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.

The objects are returned to the application client in the order requested.

Step 5. Receiving Each Object to Be Restored/Retrieved

Once the **dsmBeginGetData** call has been issued, you must perform the following activities to receive each object being sent from the server:

1. Issue the **dsmGetObj** call to identify the object being requested from the data stream and, optionally, to obtain the first block of data associated with the object.

2. Issue more **dsmGetData** calls, as necessary, to obtain the remaining object data.

After all data for an object has been obtained, or no further data for the object is desired, then a **dsmEndGetObj** call must be issued. If more objects are to be received, issue the **dsmGetObj** call again. If the process is to be stopped (normally or abnormally), issue the **dsmEndGetData** call.

Step 6. Issuing the dsmEndGetData Call

After all data for all requested objects has been received, the **dsmEndGetData** call must be issued. You can also use this call to discard any remaining data in the restore stream for all objects not yet received.

State Diagrams and Flow Charts

The state diagram for performing restore or retrieve operations is shown in Figure 20. The arrow on the right, going from the block marked “In Get Object” to **dsmEndGetData**, shows that a **dsmEndGetData** call can be issued after a call to **dsmGetObj** or **dsmGetData**. This might be desirable if an error condition occurred while getting an object from ADSM storage and you want to stop the operation. In normal circumstances, though, you should call **dsmEndGetObj** first.

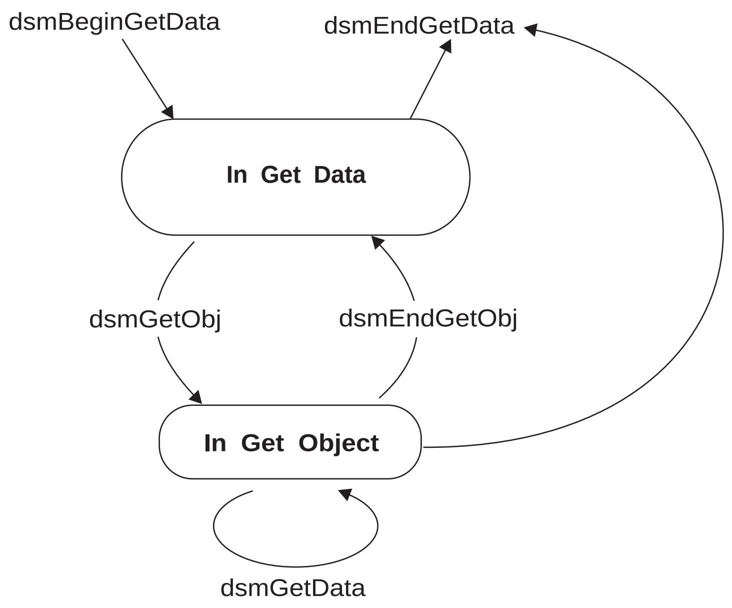

Figure 20. State Diagram for Restore and Retrieve Operations

The flow chart for performing restore or retrieve operations is shown in Figure 21 on page 38.

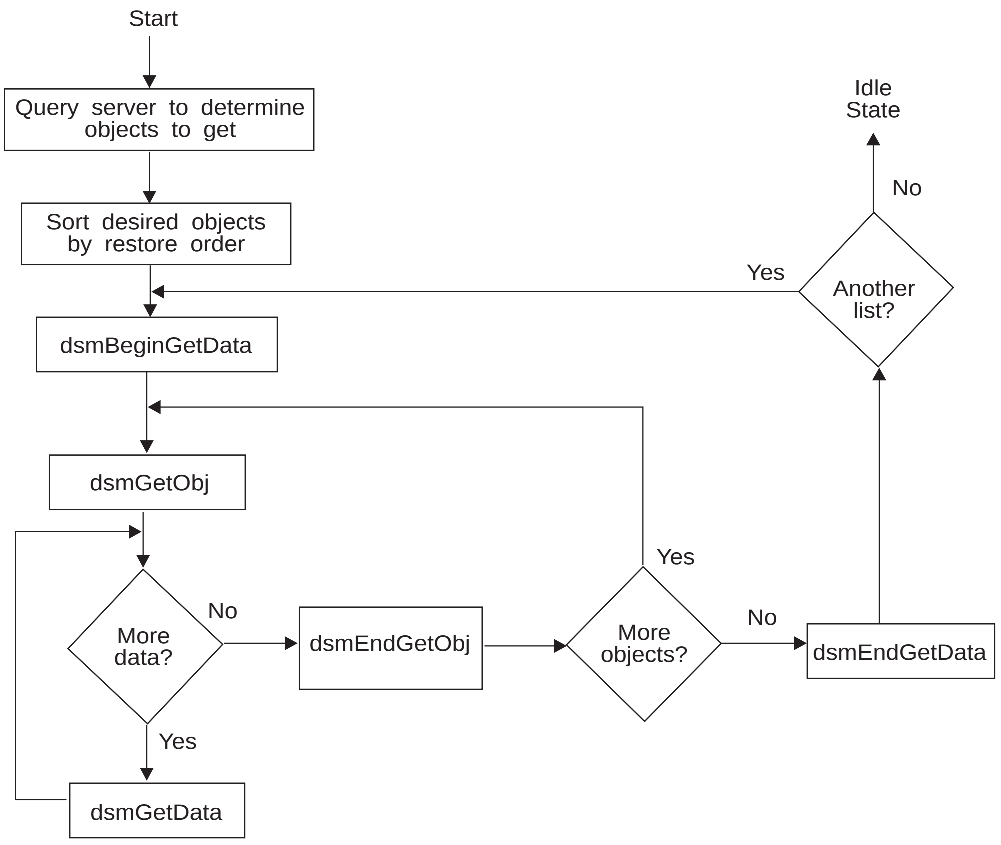

Figure 21. Flow Chart for Restore and Retrieve Operations

Example

The example in Figure 22 on page 39 demonstrates the use of the API functions that retrieve data from ADSM storage. **dsmBeginGetData** appears inside a `switch` statement, so that different parameters can be called depending on whether a restore or retrieve is being performed. **dsmGetData** is called from inside a loop that repeatedly gets data from the server until a flag is set that allows the program execution to exit the loop.

```
/* 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(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);
```

Figure 22 (Part 1 of 2). Example: Receiving Data from a Server

```
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(rc);
      done = bTrue;
   }
} /* while */
rc = dsmEndGetObj(dsmHandle);

rc = dsmEndGetData(dsmHandle);
return 0;
```

Figure 22 (Part 2 of 2). Example: Receiving Data from a Server

Deleting Objects from the Server

API applications can issue calls to either delete objects that have been archived or deactivate objects that have been backed up. The former is dependent on the node authorization given when the node was registered by an ADSM administrator. Administrators can specify whether nodes can delete archive objects.

dsmDeleteObj is used for both deleting archive objects and deactivating backup objects.

When deleting an archive object, the object is marked in ADSM 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 deactivating 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 based on the management class assigned.

Like the **dsmSendObj** call, a call to **dsmDeleteObj** is issued within the bounds of a transaction. The state diagram in Figure 17 on page 29 shows how a call to **dsmDeleteObj** is preceded by a call to **dsmBeginTxn** and followed by a call to **dsmEndTxn**.

Identifying the Object

The ADSM 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 to meet this requirement:

- 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 defined by the system administrator. 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:

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

When making decisions about naming objects for an application, keep in mind that it may be necessary to externalize the full object names to the end user. Specifically, the end user may 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 OS/2 and Windows platforms (but not UNIX), 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 Novell and AS/400 platforms, the first character of the low-level name must be a forward slash (/).

File Space Name

One of the most critical components of the name is the *file space name*. This name can be viewed as the name of a file system or disk drive, or any other high-level qualifier that groups related data together. ADSM uses the file space to identify the file system or disk drive the data is located on. Thus, actions can be performed on all entities within a file space with relative ease, such as querying all objects within a specified file space. Because the file space is such a critical component of the ADSM naming convention, ADSM has special calls to register, update, query, and delete file spaces.

The ADSM server also has administrative commands to query the file spaces on a given node in ADSM storage, and delete them if necessary. Thus, all data stored by the application client must have a file space name associated with it. Choose the name carefully to group like data together in the system.

An application client should choose different file space names than a backup-archive client would use, in order to avoid possible interference. The application client should publish its file space names, so that end users can identify the objects for Include and Exclude statements, if necessary.

Note that on the Microsoft Windows platform, when the file space is registered, you must enter a drive letter. However, the drive letter is not part of the actual file space name.

High-Level and Low-Level Names

Two other components of the object name are the *high-level* and *low-level name* qualifiers. The high-level qualifier can be viewed as the directory path the object belongs in, and the low-level qualifier as 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 the client is running on. The name does not have to exist as an object on the system or bear any relation to the actual data on the local file system, but the name must meet the standard naming rules in order to be properly processed by the **dsmBindMC** calls.

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. A *directory* is an object that contains only attributes.

Examples

| Following are four examples of object names. The first demonstrates what the application client would code on a UNIX platform:

```
/myfs/highlev/lowlev
```

The second example shows what the application client would code on a Windows or OS/2 platform:

```
myvol\\highlev\\lowlev
```

The double backslashes are ultimately translated to single backslashes. Note that file space names start with a slash on the UNIX platform, but need not do so on the Windows platform.

The third example shows what the application client would code on a Novell NetWare platform:

```
myvol:highlev/lowlev
```

| The fourth example shows what the application client would code on the AS/400
| platform:

```
| myfs/highlev/lowlev
```

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 initialized. This object owner value can be used to control access to the object.

If a session is initialized with an owner name of NULL, that session owner is treated with session (or root) authority. This session can perform any action on any object owned by this node regardless of the actual owner of that object. The session owner is set during the call to **dsmInit** in the `clientOwnerNameP` parameter.

If a session is initialized with a specific owner name, the session can only perform actions on objects that have that owner name associated with them. Thus, backups or archives into the system all must have this owner name associated with them. Also, any queries performed only return 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.

Note that an owner name is case sensitive.

Figure 23 summarizes the conditions under which a user has access to an object.

Figure 23. 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

Managing File Spaces

The file space identifier was introduced in “Identifying the Object” on page 41. 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.

A file space must first be registered with ADSM before any objects associated with that file space can be stored on the system. Use the **dsmRegisterFS** call to accomplish this task.

The file space identifier is used as the top-level qualifier in a three-part name hierarchy. Grouping related data together within a file space makes management of that data much simpler. For instance, a file space and all the objects within the file space can be deleted by either the application client or the ADSM server administrator.

File spaces also allow the application client to provide various information about the file space to the server, which can then be queried by an ADSM administrator. This information is stored in the `dsmFSUpd` structure and includes:

fstype
The file space type. This field is a character string set by the application client.

fsAttr[platform].fsInfo
A client information field used for client specific data.

capacity
The total amount of space in the file space (if applicable).

occupancy
The amount of space currently occupied in the file space (if applicable).

The use of the last two items depends on the application client. Some applications might not need information about the size of the file space, in which case these fields can be defaulted to zero.

After a file space is registered with the system, you can back up or archive objects as desired. A good practice (though not required) is to call **dsmUpdateFS** to update the file space's occupancy and capacity fields after a backup or archive operation. This insures that the values for the file system's occupancy and capacity are up-to-date.

If a file space is no longer needed, you can delete it with the **dsmDeleteFS** command.

File spaces can also be queried. (See "Querying the ADSM System" on page 22 for details.) On a file space query, all information available about the file space is returned in the `qryRespFSData` structure.

Examples

The examples in Figure 24 on page 45 demonstrate the use of the three file space calls.

```
/*  Register the file space if it hasn't already been done.  */

int16        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(rc);
   free(bkup_buff);
   return (RC_SESSION_FAILED);
}
```

Figure 24 (Part 1 of 2). Example: Working with File Spaces

```
/* 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);


/*  Delete the file space.  */

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

Figure 24 (Part 2 of 2). Example: Working with File Spaces

Putting It All Together

Figure 25 on page 47 contains an overall state diagram for the ADSM API. It contains all previous state diagrams shown plus a few other calls not depicted.

The key new points in this diagram are:

- The **dsmQueryApiVersion** call can be called at any time and has no state associated with it. An example is shown in Figure 10 on page 17.
- A set of calls that have to do with managing file spaces (**dsmRegisterFS**, **dsmUpdateFS**, **dsmDeleteFS**). These calls are made from within an idle session state. File spaces are queried using the **dsmBeginQuery** call. For more information on the file space calls, see "Managing File Spaces" on page 43.
- The **dsmBindMC** call can be issued from within an idle session state or from within a send object transaction state. See the example in Figure 13 on page 22.

- The **dsmChangePW** call is issued from within an idle session state. Note that if the **dsmInit** call returns with a password expired return code, the **dsmChangePW** call must be made before a valid session is started. An example using **dsmChangePW** is shown in Figure 12 on page 20.

| **Note:** 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.

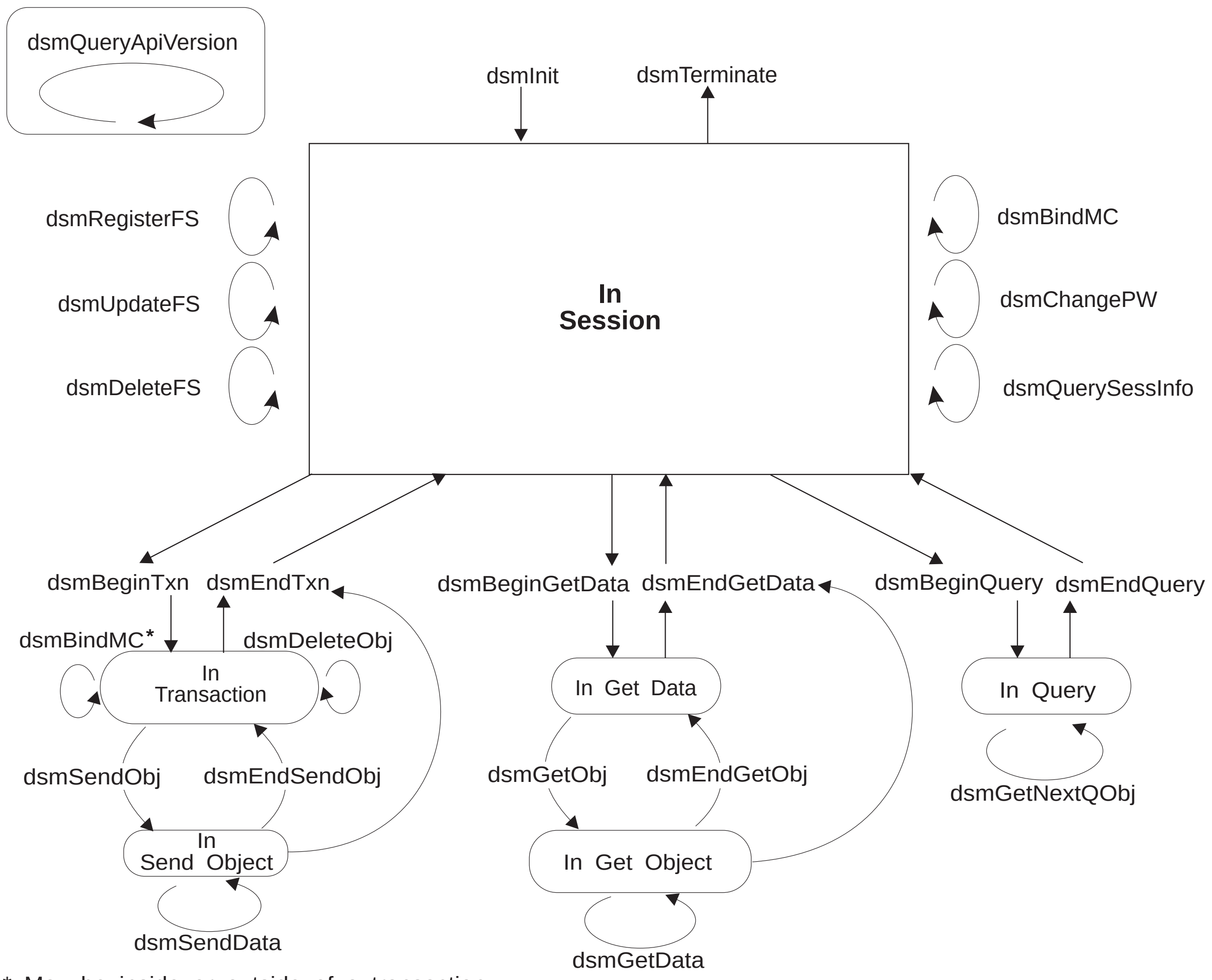

Figure 25. Global State Diagram for ADSM

Determining Size Limits

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

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 instead of hard-coded numbers. Refer to Appendix A, “API Type Definitions Source File” on page 91 for further information and a list of the current constants.

Chapter 4. API Function Definitions

This section describes the calls that constitute the application program interface. The functions are listed in alphabetical order.

For each function, the following items are described:

- **Purpose** — a description of the circumstances in which the function is used.
- **Syntax** — the actual C code for the function. This code is copied from the UNIX version of the header file **dsmapifp.h** (see Appendix B, “API Function Definitions Source File” on page 107). Note that this file differs slightly on other platforms. Application programmers for other platforms should check their version of **dsmapifp.h** for the exact syntax of the API definitions.
- **Parameters** — a description of each parameter in the function call. Each parameter is identified as (I) or (O), depending on whether it is used for input or output. A few parameters are designated as (I/O). These are data structures that are used for both input and output. The data types referred to in this section are defined in the header file **dsmapitd.h** (see Appendix A, “API Type Definitions Source File” on page 91).
- **Return Codes** — a list of the return codes specific to the function. Note that there are also general system errors, such as communication errors, server problems, and user mistakes, that could appear on any call, and these are not listed on the following pages. For a complete list of all the return codes with explanations, see Appendix D, “API Return Codes With Explanations” on page 123.

dsmBeginGetData

Use **dsmBeginGetData** to start a restore or retrieve operation on a list of objects in ADSM storage. The list of objects is contained in the `dsmGetList` structure. This list is created in the call to **dsmBeginQuery** that must precede a call to **dsmBeginGetData**.

The list contained in this call must first be sorted by the caller using the restore order fields obtained from the object query. This is critical to insure the objects are restored from ADSM storage in the most efficient mode possible without having to rewind or remount data tapes.

| The list can contain as many objects as are defined in DSM_MAX_GET_OBJ (4080).

The call to **dsmBeginGetData** should be followed by one or more calls to **dsmGetObj** to obtain each object within the list. After each object is obtained (or no further data for the object is desired), the **dsmEndGetObj** call must be issued.

When all objects have been obtained (or the get is to be canceled), the **dsmEndGetData** call is issued. The cycle can then be started again if required.

Syntax

```
int16 dsmBeginGetData
  (uint32       dsmHandle,
   bool_t       mountWait,
   dsmGetType   getType,
   dsmGetList  *dsmGetObjListP);
```

Parameters

uint32 dsmHandle (I)
This parameter is the handle that associates this call with a previous **dsmInit** call.

bool_t mountWait (I)
A boolean True/False value indicating whether the application client is willing to wait for offline media to be mounted if the desired data is currently offline.

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

dsmGetList *dsmGetObjListP (I)
| This parameter is the structure that contains information on 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 retrieve or restore, a list of associated offsets and lengths.
| If your application will be using the partial object restore or retrieve function, set the
| `dsmGetList.stVersion` field to `dsmGetListPORVersion`. Also, in a partial object
| restore or retrieve, you cannot compress data while sending it. To enforce this, set
| `ObjAttr.objCompressed` to `bTrue`.

Refer to Figure 22 on page 39 and Appendix A, “API Type Definitions Source File” on page 91 for further details on this structure.

| Refer to page 33 for more information on partial object restore or retrieve.

Return Codes

Figure 26. Return Codes for dsmBeginGetData

Return Code	Explanation
DSM_RC_ABORT_INVALID_OFFSET (33)	The offset specified during a partial object retrieve is greater than the length of the object.
DSM_RC_ABORT_INVALID_LENGTH (34)	The length specified during a partial object retrieve is greater than the length of the object, or the offset plus the length extends past the end of the object.
DSM_RC_NO_MEMORY (102)	There is no RAM left to complete request.
| DSM_RC_NUMOBJ_EXCEED (2029)	`dsmGetList.numObjId` is greater than DSM_MAX_GET_OBJ.
DSM_RC_OBJID_NOTFOUND (2063)	Object ID not found, so object was not restored.
DSM_RC_WRONG_VERSION_PARM (2065)	Application client's API version is different than the ADSM library version.

dsmBeginQuery

Use **dsmBeginQuery** to initiate a query request to ADSM for information about one of the following areas:

Archived data
Backed up data
Active backed up data
File spaces
Management classes

The query data returned from the call is obtained by one or more calls to **dsmGetNextQObj**.

When the query is complete (whether successful or not), the **dsmEndQuery** call is issued.

Syntax

```
int16 dsmBeginQuery
  (uint32        dsmHandle,
   dsmQueryType  queryType,
   dsmQueryBuff  *queryBuffer);
```

Parameters

uint32 dsmHandle (I)
This parameter is the handle that associates this call with a previous **dsmInit** call.

dsmQueryType queryType (I)
Identifies the type of query to perform. You can choose from:

qtArchive	Queries archived objects
qtBackup	Queries backed up objects
qtBackupActive	Queries only active backed up objects for the entire file space name you pass. This is referred to as “fast path” and is the most efficient way to query active objects from ADSM storage. **Note:** On multi-user platforms, you *must* be the root user.
qtFilespace	Queries registered file spaces
qtMC	Queries defined management classes

dsmQueryBuff *queryBuffer (I)
Identifies a pointer to a buffer that is mapped to a particular data structure associated with the query type you pass.

These structures contain the selection criteria for each query type. The fields in each structure must be filled out to specify the scope of the query to be performed. The first field of each structure is `stVersion`, which is the API version number. The data structures and their other fields are:

qryArchiveData

objName The full name of the object. 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 42). 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 name of the object's owner.

insDateLowerBound
The lower bound for the archive insert date (the date the object was archived). To get the default lower bound, set the `year` component to `DATE_MINUS_INFINITE`.

insDateUpperBound
The upper bound for the archive insert date. To get the default upper bound, set the `year` component to `DATE_PLUS_INFINITE`.

expDateLowerBound
The lower bound for the expiration date. The default values for both expiration date fields are the same as for the insert date fields.

expDateUpperBound
The upper bound for the expiration date.

descr The archive description. Enter an asterisk (*) to search on all descriptions.

qryBackupData

objName The full name of the object. 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 42). 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 name of the object's owner.

objState This field can have one of the three values `DSM_ACTIVE`, `DSM_INACTIVE`, or `DSM_ANY_MATCH`.

qryABackupData

objName The full name of the object. 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 42). 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`.

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 get information about all management classes. Note that an asterisk (*) will *not* work.

mcDetail This field has the value `bTrue` or `bFalse`. The value determines whether information on the management class's backup and archive copy groups is returned.

Return Codes

Figure 27. Return Codes for dsmBeginQuery

Return Code	Explanation
DSM_RC_NO_MEMORY (102)	There is no RAM left to complete request.
DSM_RC_FILE_SPACE_NOT_FOUND (124)	Specified file space was not found.
DSM_RC_NO_POLICY_BLK (2007)	No server policy information available.
DSM_RC_INVALID_OBJTYPE (2010)	Invalid object type.
DSM_RC_INVALID_OBJOWNER (2019)	Invalid object owner name.
DSM_RC_INVALID_OBJSTATE (2024)	Invalid object state.
DSM_RC_WRONG_VERSION_PARM (2065)	Application client's API version is different than the ADSM library version.

dsmBeginTxn

Use **dsmBeginTxn** to begin an ADSM transaction. The **dsmBeginTxn** call indicates to ADSM the beginning of one or more actions that are executed as an atomic unit, that is, all the actions succeed or none succeed. An action can be assumed to be either a single ADSM call or a series of ADSM calls that are made for a particular end. For example, a **dsmSendObj** call followed by a number of **dsmSendData** calls can be considered a single action. Similarly, a **dsmSendObj** that has a `dataBlkPtr` indicating a data area that fully contains the object to be backed up is also a single action.

Try to group multiple objects together in a single transaction for data transfer operations. Grouping objects results in significant performance improvements in the ADSM system. A certain amount of overhead is incurred in initiating and terminating a transaction from both a client and a server perspective. Sending multiple objects within a single transaction also means that multiple objects can be sent within a single low-level communication buffer, which greatly decreases the number of communication line turnarounds.

ADSM has certain limits to what can be performed within a single transaction. These restrictions are:

- There is a maximum number of objects that can be sent in a single transaction. This limit can be found from the data returned by **dsmQuerySessInfo** in the field `ApiSessInfo.maxObjPerTxn`.
- Objects sent to the server (backup or archive) within a single transaction must all have the same copy destination as defined in the management class binding for the object. This limit can be found from the data returned by **dsmBindMC** in the fields `mcBindKey.backup_copy_dest` and `mcBindKey.archive_copy_dest`.

The ADSM API has been designed so that either the application client can monitor and control these restrictions, or the API itself can monitor these restrictions and inform the application client when one or more have been reached via appropriate return codes from the API calls.

A **dsmBeginTxn** call is always coupled with a **dsmEndTxn** call. ADSM attempts to optimize the set of actions within a pair of **dsmBeginTxn** and **dsmEndTxn** calls.

Syntax

```
int16 dsmBeginTxn
  (uint32 dsmHandle);
```

Parameters

uint32 dsmHandle (I)
This parameter is the handle that associates this call with a previous **dsmInit** call.

dsmBeginTxn

Return Codes

There are no return codes specifically for this call.

dsmBindMC

Use **dsmBindMC** to associate, or bind, an ADSM management class to the passed object. This binding is performed by passing the object through the include/exclude list pointed to in the options file. If no match is found in this list for a specific management class, the default management class is assigned.

The parameters returned in the `mcBindKey` structure can be used by the application client to determine if this object should be backed up or archived, and whether a new transaction must be started due to different copy destinations (see **dsmBeginTxn**).

You must call **dsmBindMC** prior to calling **dsmSendObj**, because every object must have a management class associated with it. This call can be done inside or outside of the bounds of a transaction. For example, while in a multiple object transaction, if **dsmBindMC** indicates that the object has a different copy destination than the previous object, then 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

```
int16 dsmBindMC
  (uint32       dsmHandle,
   dsmObjName  *objNameP,
   dsmSendType  sendType,
   mcBindKey   *mcBindKeyP);
```

Parameters

uint32 dsmHandle (I)
This parameter is the handle that associates this call with a previous **dsmInit** call.

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.

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

stBackup	Specifies that this is a backup object
stArchive	Specifies that this is an archive object
stBackupMountWait	Specifies that this is a backup object
stArchiveMountWait	Specifies that this is 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 a `mcBindKey` structure where the management class information is returned. The application client can use the information returned here to

determine if this object fits within a multiple object transaction, or to do a management class query on the management class bound to the object.

Return Codes

Figure 28. Return Codes for dsmBindMC

Return Code	Explanation
DSM_RC_NO_MEMORY (102)	There is no RAM left to complete request.
DSM_RC_INVALID_PARM (109)	One of the parameters passed has an invalid value.
DSM_RC_TL_EXCLUDED (185)	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's API version is different than the ADSM library version.

dsmChangePW

Use **dsmChangePW** to change an ADSM password. On a multiple-user platform, such as UNIX, this call can only be used by the root user within the login user ID of root.

On the Windows, OS/2, and Novell platforms, the password can be specified in the **dsm.opt** file. In this situation, **dsmChangePW** does not update **dsm.opt**. After the call to **dsmChangePW** is made, **dsm.opt** must be updated separately.

| A successful execution of this call is required if **dsmInit** returns DSM_RC_VERIFIER_EXPIRED. The session will terminate if the **dsmChangePW** call fails in this situation.

| If **dsmChangePW** is called for some other reason, the session will remain open, no matter what the return code.

Syntax

```
int16 dsmChangePW
  (uint32   dsmHandle,
   char    *oldPW,
   char    *newPW);
```

Parameters

uint32 dsmHandle (I)
This parameter is the handle that associates this call with a previous **dsmInit** call.

char *oldPW (I)
This parameter is the caller's old password.

char *newPW (I)
This parameter is the caller's new password.

Return Codes

Figure 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)	API caller must be a root user.

dsmDeleteFS

Use **dsmDeleteFS** to delete a file space from ADSM storage.

To delete a file space, you must have the appropriate permissions granted to you by your ADSM administrator. You can find out whether you have the necessary permissions by calling **dsmQuerySessInfo**. This function returns a data structure of type `ApiSessInfo`, which includes two fields called `archDel` and `backDel`.

If the file space that you want to delete contains backup versions, you must have backup delete authority (`backDel = BACKDEL_YES`). If it contains archive copies, you must have archive delete authority (`archDel = ARCHDEL_YES`). If the file space contains both backup versions and archive copies, you need to have both types of delete authority.

Note that on a UNIX platform only a root user within the login user ID of root can delete a file space.

Syntax

```
int16 dsmDeleteFS
  (uint32          dsmHandle,
   char           *fsName,
   unsigned char   repository);
```

Parameters

uint32 dsmHandle (I)
This parameter is the handle that associates this call with a previous **dsmInit** call.

char *fsName (I)
This parameter is a pointer to the file space name to be deleted.

unsigned char repository (I)
This parameter indicates whether the file space to be deleted is a backup repository, archive repository, or both. The possible values for this field are:

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

Return Codes

Figure 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

Use **dsmDeleteObj** to deactivate backup objects or delete archive objects in ADSM storage. This function must be called from within a transaction (see **dsmBeginTxn**).

The maximum number of objects that can be deleted in a single transaction is determined by the value of `maxObjPerTxn`. You can get this value by calling **dsmQuerySessInfo**.

Note that you must have the appropriate permission granted to you by your ADSM administrator. To delete archive objects, you must have archive delete authority. You do not need backup delete authority to deactivate a backup object.

Syntax

```
int16 dsmDeleteObj
  (uint32      dsmHandle,
   dsmDelType  delType,
   dsmDelInfo  delInfo)
```

Parameters

uint32 dsmHandle (I)
This parameter is the handle that associates this call with a previous **dsmInit** call.

dsmDelType delType (I)
Indicates what type of object (backup or archive) is to be deleted. Possible values are:

dtArchive	Object to delete was previously archived
dtBackup	Object to deactivate was previously backed up

dsmDelInfo delInfo (I)
This parameter is a structure whose fields are used to identify the object. The fields are different, depending on whether the object is a backup object or an archive object. The structure for a backup object, called `delBack`, contains the object name and the object's copy group. The structure for an archive object, called `delArch`, contains the object ID.

The information in `delInfo` is obtained by issuing the query sequence described in "Querying the ADSM System" on page 22. The call to **dsmGetNextQObj** returns a data structure called `qryRespBackupData` for backup queries or `qryRespArchiveData` for archive queries. These data structures contain the information needed for `delInfo`.

Return Codes

Figure 31. Return Codes for dsmDeleteObj

Return Code	Explanation
DSM_RC_FS_NOT_REGISTERED (2061)	File space name not registered.
DSM_RC_WRONG_VERSION_PARM (2065)	Application client's API version is different than the ADSM library version.

dsmEndGetData

Use **dsmEndGetData** to signify the end of a **dsmBeginGetData** session for obtaining objects from ADSM storage.

dsmEndGetData can be invoked after all objects to restore have been processed or to terminate the get process prematurely. **dsmEndGetData** must be called to end a **dsmBeginGetData** session before other processing can continue.

Depending on when **dsmEndGetData** is called, the ADSM API might have to finish processing a partial data stream before the process can be stopped. The caller, therefore, should not always expect an immediate return from this call.

Syntax

```
int16 dsmEndGetData
  (uint32 dsmHandle);
```

Parameters

uint32 dsmHandle (I)
This parameter is the handle that associates this call with a previous **dsmInit** call.

Return Codes

There are no return codes specifically for this call.

dsmEndGetObj

Use **dsmEndGetObj** to signify the end of a **dsmGetObj** cycle for obtaining data for a specified object.

dsmEndGetObj can be invoked after an end of data has been received for the object signaling that all data has been received, or can be invoked to indicate no further data is to be received for this object. **dsmEndGetObj** must be called before another **dsmGetObj** call can be invoked.

Depending on when **dsmEndGetObj** is called, the ADSM API might have to finish processing a partial data stream before the process can be stopped. The caller, therefore, should not always expect an immediate return from this call.

Syntax

```
int16 dsmEndGetObj
  (uint32 dsmHandle);
```

Parameters

uint32 dsmHandle (I)
This parameter is the handle that associates this call with a previous **dsmInit** call.

Return Codes

Figure 32. Return Codes for dsmEndGetObj

Return Code	Explanation
DSM_RC_NO_MEMORY (102)	There is no RAM left to complete request.

dsmEndQuery

Use **dsmEndQuery** to signify the end of a **dsmBeginQuery** action.

The application client issues **dsmEndQuery** to complete a query. This call is either issued after all query responses have been obtained through **dsmGetNextQObj** or to end a query before all data is returned. Note that ADSM still sends the query data from the server to the client in this case, but the ADSM API just flushes any remaining data.

Once a **dsmBeginQuery** has been issued, a **dsmEndQuery** must be issued before any other activity can be started.

Syntax

```
int16 dsmEndQuery
  (uint32 dsmHandle);
```

Parameters

uint32 dsmHandle (I)
This parameter is the handle that associates this call with a previous **dsmInit** call.

Return Codes

There are no return codes specifically for this call.

dsmEndSendObj

Use **dsmEndSendObj** to signify the end of data being sent to ADSM storage.

dsmEndSendObj must be called to indicate the end of data from **dsmSendObj** and **dsmSendData** calls. Not doing so causes a protocol violation. The only exception to this rule is calling **dsmEndTxn** to abort the transaction, which then discards all data sent for that transaction.

Syntax

```
int16 dsmEndSendObj
  (uint32 dsmHandle);
```

Parameters

uint32 dsmHandle (I)
This parameter is the handle that associates this call with a previous **dsmInit** call.

Return Codes

Figure 33. Return Codes for dsmEndSendObj

Return Code	Explanation
DSM_RC_NO_MEMORY (102)	There is no RAM left to complete request.

dsmEndTxn

Use **dsmEndTxn** to end an ADSM transaction. **dsmEndTxn** is coupled with **dsmBeginTxn** to identify the ADSM call or set of ADSM calls that are to be treated as a transaction. The application client can specify on the **dsmEndTxn** call whether or not the transaction is to be committed or aborted.

All of the following calls must be performed within the bounds of a transaction:

dsmSendObj
dsmSendData
dsmEndSendObj
dsmDeleteObj

Syntax

```
int16 dsmEndTxn
  (uint32   dsmHandle,
   uint8    vote,
   uint16  *reason);
```

Parameters

uint32 dsmHandle (I)
This parameter is the handle that associates this call with a previous **dsmInit** call.

uint8 vote (I)
This parameter indicates whether or not the application client wants to commit all the actions done between the previous **dsmBeginTxn** call and this call. Its possible values are listed below.

```
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 halt the transaction.

uint16 *reason (O)
If the call to **dsmEndTxn** ends with an error or the value of `vote` is not agreed to, then this parameter will have a reason code indicating why the vote failed. Note that the return code for the call could be 0 and the reason code could be non-zero. Thus the application client must always check for errors on both the return code and the reason (`if (rc || reason)`) before successful completion can be assumed. See Appendix C, “API Return Codes Source File” on page 113 for a list of the possible reason codes. Numbers 1 - 50 in the return codes list are reserved for the reason codes.

Return Codes

Figure 34. Return Codes for dsmEndTxn

Return Code	Explanation
DSM_RC_INVALID_VOTE (2011)	The value specified for vote is invalid.
DSM_RC_CHECK_REASON_CODE (2302)	The transaction was aborted, so check the reason field.

dsmGetData

Use **dsmGetData** to get a byte stream of data from ADSM and place 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

```
int16 dsmGetData
  (uint32   dsmHandle,
   DataBlk *dataBlkPtr);
```

Parameters

uint32 dsmHandle (I)
This parameter is the handle that associates this call with a previous **dsmInit** call.

DataBlk *dataBlkPtr (I/O)
This parameter points to a structure that includes both a pointer to the buffer for the data that is to be received and the size of the buffer. On return, this structure contains the number of bytes actually transferred. See Appendix A, “API Type Definitions Source File” on page 91 for the type definition.

Return Codes

Figure 35. Return Codes for dsmGetData

Return Code	Explanation
DSM_RC_ABORT_INVALID_OFFSET (33)	The offset specified during a partial object retrieve is greater than the length of the object.
DSM_RC_ABORT_INVALID_LENGTH (34)	The length specified during a partial object retrieve is greater than the length of the object, or the offset plus the length extends past the end of the object.
DSM_RC_FINISHED (121)	Finished processing (issue dsmEndGetObj).
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)	Application client's API version is different than the ADSM library version.
DSM_RC_MORE_DATA (2200)	There is more data to get.

dsmGetNextQObj

Use **dsmGetNextQObj** to get the next query response from a previous **dsmBeginQuery** call and place it in the caller's buffer. **dsmGetNextQObj** is called one or more times, and each time it is called, a single query record is retrieved. If the application client wishes to end the query before retrieving all of the data, a **dsmEndQuery** call can be issued.

The `dataBlkPtr` must point to a buffer that is defined with the `qryResp*Data` structure type. The type of structure filled in on the query response is determined by the context in which **dsmGetNextQObj** is called. The structure associated with each type of query is:

Query	Response Structure
qtArchive	qryRespArchiveData
qtBackup	qryRespBackupData
qtBackupActive	qryARespBackupData
qtFilespace	qryRespFSData
qtMC	qryRespMCData, qryRespMCDetailData

Syntax

```
int16 dsmGetNextQObj
  (uint32   dsmHandle,
   DataBlk *dataBlkPtr);
```

Parameters

uint32 dsmHandle (I)
This parameter is the handle that associates this call with a previous **dsmInit** call.

DataBlk *dataBlkPtr (I/O)
This parameter points to a structure that includes both a pointer to the buffer for the data that is to be received and the size of the buffer. This buffer is the `qryResp*Data` structure described above. On return, this structure contains the number of bytes actually transferred. See Appendix A, “API Type Definitions Source File” on page 91 for the type definition of `DataBlk`.

Return Codes

Figure 36. Return Codes for dsmGetNextQObj

Return Code	Explanation
DSM_RC_ABORT_NO_MATCH (2)	No match for the query requested.
DSM_RC_FINISHED (121)	Finished processing (issue dsmEndQuery).
DSM_RC_UNKNOWN_FORMAT (122)	The file that ADSM 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 datablock.
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 than the ADSM library version.
DSM_RC_MORE_DATA (2200)	There is more data to get.
DSM_RC_BUFF_TOO_SMALL (2210)	Buffer is too small.

dsmGetObj

Use **dsmGetObj** to obtain the requested object data from the ADSM data stream and place it in the caller's buffer. **dsmGetObj** uses the object ID, offset, and length information to obtain the next object or partial object from the data stream.

The data for the indicated object is placed in the buffer pointed to by `DataBlk`. If more data is available, then one or more calls to **dsmGetData** must be made 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 usually be asked for in the order that they were listed on the **dsmBeginGetData** call in the `dsmGetList` parameter. The exception to this is when the application client wishes to pass over an object(s) in the data stream to get to an object later in the list. If the object indicated by the object ID is not the next object in the stream, then the data stream is processed until the object is found or the stream is exhausted. Use this feature with care, because large amounts of data might have to be processed and discarded to find the requested object.

Syntax

```
int16 dsmGetObj
  (uint32   dsmHandle,
   ObjID    *objIdP,
   DataBlk  *dataBlkPtr);
```

Parameters

uint32 dsmHandle (I)
This parameter is the handle that associates this call with a previous **dsmInit** call.

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

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

Return Codes

Figure 37. Return Codes for dsmGetObj

Return Code	Explanation
DSM_RC_ABORT_INVALID_OFFSET (33)	The offset specified during a partial object retrieve is greater than the length of the object.
DSM_RC_ABORT_INVALID_LENGTH (34)	The length specified during a partial object retrieve is greater than the length of the object, or the offset plus the length extends past the end of the object.
DSM_RC_FINISHED (121)	Finished processing (issue dsmEndGetObj).
DSM_RC_WRONG_VERSION_PARM (2065)	Application client's API version is different than the ADSM library version.
DSM_RC_MORE_DATA (2200)	There is more data to get.

dsmInit

Use **dsmInit** to start an ADSM API session and connect the client to ADSM storage. The application client can only have one active session open at a time. To open another session with different parameters, the **dsmTerminate** call must first be used to end the current session.

Syntax

```
int16  dsmInit
  (uint32        *dsmHandle,
   dsmApiVersion *dsmApiVersionP,
   char          *clientNodeNameP,
   char          *clientOwnerNameP,
   char          *clientPasswordP,
   char          *applicationType,
   char          *configfile,
   char          *options);
```

Parameters

uint32 *dsmHandle (O)
This parameter is the handle that identifies this initialization session and associates it with subsequent ADSM calls.

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 three constants `DSM_API_VERSION`, `DSM_API_RELEASE`, and `DSM_API_LEVEL` set in **dsmapitd.h**. A previous call to **dsmQueryApiVersion** must be performed to insure compatibility between the application client's API version 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 ADSM session. All ADSM sessions must have a node name associated with them. The maximum size allowed for a node name is set by the constant `DSM_MAX_NODE_LENGTH` in the file **dsmapitd.h**.

The node name is not case sensitive.

If this parameter is set to NULL, the API attempts to obtain the node name first from the options string passed, and, if it is not there, from the configuration file or options files. If these attempts to find the node name fail, the UNIX API uses the system host name, while other platforms' APIs return the code `DSM_RC_REJECT_ID_UNKNOWN`.

This parameter must be NULL if the PASSWORDACCESS option in the **dsm.sys** configuration file has been set to Generate. The API then uses the system host name.

char *clientOwnerNameP (I)
This parameter is a pointer to the owner of the ADSM session. If the platform on which the ADSM session is initialized is a multi-user platform, then 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** configuration file has been set to Generate. The API then uses the login user ID.

Note: On a multi-user platform, the owner name does not have 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 ADSM session runs. The maximum size allowed for a password is set by the constant `DSM_MAX_VERIFIER_LENGTH` in the file **dsmapitd.h**.

The password is not case sensitive.

If PASSWORDACCESS=Generate, then the value of this parameter is ignored.

char *applicationType (I)
This parameter identifies the application that is running the ADSM session. The value is defined by the application client. Note that this value is registered at the ADSM server the first time a session is initialized after it has been registered by an administrator. This value is then bound as part of the node name for the life of the node name.

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.

On the UNIX platform, the API configuration file is a superset of the user options file. It can include any of the supported options of the user options file plus the following options from the system options file.

COMPRESSION
DIRMC
INCLEXCL

| On the Windows, OS/2, Netware, and AS/400 platforms, the API configuration file
| can include any user option.

These options can be specified in the API configuration file to override their specification in the system options file (on the UNIX platform) and the client options file. The options files are defined when ADSM (client or API) is installed.

For the description and use of configuration files, refer to “Configuration Files and Options Files” on page 2. Another source of information is the *Installing the Clients* book.

char *options (I)
| This parameter points to a character string that can contain user options such as:

| NODENAME

| SERVERNAME (UNIX only)
| TCPServeraddr (non-UNIX)

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

The format of the options is as follows: each option specified in the option list begins with a dash (-) and is followed directly by the option keyword. The keyword, in turn, is followed directly by an equal sign (=) and then followed by the option parameter. If the option parameter contains a blank space, the parameter must be surrounded by single or double quotes. If more than one option is specified, the options are separated by blanks.

If `options` is NULL, then values for all options are taken from the user options file or the API configuration file. Descriptions and usage of each option can be found | in the **options.doc** file or in the *Installing the Clients* book.

Return Codes

Figure 38 (Page 1 of 2). Return Codes for dsmInit

Return Code	**Explanation**
DSM_RC_BAD_HOST_ID (-103)	Session rejected. Unexpected 3270 emulator error.
DSM_RC_PC3270_MISSING_DLL (-123)	PC3270 DLL not in user's path.
DSM_RC_3270COMM_MISSING_DLL (-124)	DSM3270.DLL not in user's path.
DSM_RC_ABORT_SYSTEM_ERROR (1)	The ADSM 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_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_UNMATCHED_QUOTE (177)	An unmatched quote is in the option string.
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=Generate.
DSM_RC_NO_NODE_REQD (2033)	Node parameter must be NULL when PASSWORDACCESS=Generate.
DSM_RC_WRONG_VERSION (2064)	Application client's API version has a higher value than the ADSM version.
DSM_RC_PASSWD_TOOLONG (2103)	The specified password is too long.
DSM_RC_NO_OPT_FILE (2220)	No configuration file could be found.

Figure 38 (Page 2 of 2). Return Codes for dsmInit

Return Code	Explanation
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 ADSM.
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 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.

dsmQueryApiVersion

Use **dsmQueryApiVersion** to perform a query request for the API library version being accessed by the application client.

The version is specified in the following three-part format: ***version.release.level***. Updates that require changes to the API have a corresponding change in either the ***version*** or ***release*** values. Updates to the ***level*** do not involve any changes in parameters or returned levels from the API, but indicate code fixes to the underlying API.

All updates to the API are made in an upward compatible format. This means that 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. If the **dsmQueryApiVersion** call returns a version or version/release that is older than that of the application client, then caution should be taken before proceeding, because some API calls might have been enhanced in a manner that is not supported by the end user's older version of API.

The application's API version number is stored in the **dsmapitd.h** header file as the constants `DSM_API_VERSION`, `DSM_API_RELEASE`, and `DSM_API_LEVEL`.

Syntax

```
void dsmQueryApiVersion
  (dsmApiVersion *apiVersionP);
```

Parameters

dsmApiVersion *apiVersionP (O)

This parameter is a pointer to the structure that contains the API library's version, release, and level components. For instance, 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
```

Return Codes

There are no return codes specifically for this call.

dsmQuerySessInfo

Use **dsmQuerySessInfo** to initiate a query request to ADSM for all information related to the operation of the specified session in `dsmHandle`. A structure of type `ApiSessInfo` is passed in the call, which is filled out with all available session related information. This call is typically issued after a successful **dsmInit** call.

The information returned in the `ApiSessInfo` structure includes the following:

- server information — port number, date/time, type
- client defaults — application type, delete permissions, delimiters, transaction limits
- session information — login ID, owner
- policy data — domain, active policy set, retention grace period

See Appendix A, "API Type Definitions Source File" on page 91 for details on the content of the structure passed and each field within it.

Syntax

```
int16 dsmQuerySessInfo
  (uint32         dsmHandle,
   ApiSessInfo   *SessInfoP);
```

Parameters

uint32 dsmHandle (I)
This parameter is the handle that associates this call with a previous **dsmInit** call.

ApiSessInfo *SessInfoP (I/O)
This parameter is used to pass the address of the structure that the ADSM API fills out. The application client is responsible for allocating storage for the structure and for filling out the field that indicates the version of the structure being used. Upon successful return, the fields in the structure are filled in with the appropriate information.

Return Codes

Figure 39. 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 than the ADSM library version.

dsmRCMsg

Use **dsmRCMsg** to get the message text that is associated with an ADSM API return code.

The msg parameter displays the 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.
```

Syntax

```
int16 dsmRCMsg
  (uint32      dsmHandle,
   int16       dsmRC,
   char       *msg);
```

Parameters

uint32 dsmHandle (I)
This parameter is the handle that associates this call with a previous **dsmInit** call. Currently this parameter is not used by **dsmRCMsg**. It is present to allow future enhancements to the function. The parameter must be passed for syntax checking, but its value is ignored.

int16 dsmRC (I)
This parameter is the ADSM API return code for which you want the associated message text. The API return codes are listed in the file **dsmrc.h** (see Appendix C, "API Return Codes Source File" on page 113).

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.

Return Codes

Figure 40. Return Codes for dsmRCMsg

Return Code	Explanation
DSM_RC_NULL_MSG (2002)	msg parameter for dsmRCMsg call is a NULL pointer.
DSM_RC_INVALID_RETCODE (2021)	Return code passed to dsmRCMsg call is an invalid one.

dsmRegisterFS

Use **dsmRegisterFS** to register a new file space for the node with the ADSM server. A file space must first be registered before any data can be backed up to it.

Application clients must not use the same file space names that a backup-archive client would use.

- On UNIX, run the **df** command to see what these names are.
- On Windows and OS/2, these names are generally the volume labels associated with the different drives on your system.
- On Novell NetWare, these names are normally volume names. There is always one called `sys:`, but other volume names can be almost anything.
- On AS/400, there is no backup-archive client.

Syntax

```
int16 dsmRegisterFS
  (uint32        dsmHandle,
   regFSData     *regFilespaceP);
```

Parameters

uint32 dsmHandle (I)
This parameter is the handle that associates this call with a previous **dsmInit** call.

regFSData *regFilespaceP (I)
This parameter is used to pass the name of the file space and associated information that is to be registered with the ADSM server.

Return Codes

Figure 41. 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 than the ADSM library version.
DSM_RC_FSINFO_TOOLONG (2106)	File space information too long.

dsmSendData

Use **dsmSendData** to send a byte stream of data to ADSM via a buffer. The application client can pass any type of data for storage on the server. This data is typically file data, but is not limited to such. **dsmSendData** can be called multiple times, in case the byte stream of data to be sent is large. Note that the buffer specified in **dsmSendData** cannot be reused by the application client until **dsmSendData** returns.

Note: If ADSM returns code 157 (`DSM_RC_WILL_ABORT`), issue a call to **dsmEndSendObj** and then to **dsmEndTxn** with a vote of `DSM_VOTE_COMMIT`. The application should then get back return code 2302 (`DSM_RC_CHECK_REASON_CODE`) and pass the reason code back to the application user. This will tell the user why the server is aborting the transaction.

Syntax

```
int16 dsmSendData
  (uint32   dsmHandle,
   DataBlk *dataBlkPtr);
```

Parameters

uint32 dsmHandle (I)
This parameter is the handle that associates this call with a previous **dsmInit** call.

DataBlk *dataBlkPtr (I/O)
This parameter points to a structure that includes both a pointer to the buffer from which the data is to be sent, as well as the size of the buffer. On return, this structure contains the number of bytes actually transferred. See Appendix A, “API Type Definitions Source File” on page 91 for the type definition.

Return Codes

Figure 42. Return Codes for dsmSendData

Return Code	Explanation
DSM_RC_NO_COMPRESS_MEMORY (154)	Insufficient memory available to do data compression or expansion.
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 be halted.
DSM_RC_WRONG_VERSION_PARM (2065)	Application client's API version is different than the ADSM library version.
DSM_RC_NEEDTO_ENDTXN (2070)	Need to end transaction.
DSM_RC_OBJ_EXCLUDED (2080)	The object is excluded by the include-exclude list.
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 will not be sent to the server.

dsmSendObj

Use **dsmSendObj** to initiate a request to send a single object to ADSM 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 allows the application client to either:

- pass the data and the attributes (the attributes are passed via 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.

An alternative is for the application client to specify only the attributes through the **dsmSendObj** call and to specify the object data through one or more calls to **dsmSendData**. For this latter alternative, set `dataBlkPtr` on the **dsmSendObj** call to NULL. Note that for certain object types, byte stream data might not be associated with the data at all. An example of such an object could be a directory entry with no extended attributes.

Before **dsmSendObj** can be called, a preceding **dsmBindMC** call must be made to properly bind a management class to the object being backed up or archived. The ADSM API retains this binding so that it can associate the proper management class with the object when it is sent to the server.

All object data sent to ADSM storage must be followed by a **dsmEndSendObj** call. Thus, if you do not have object data to send to the server or all data was contained within the **dsmSendObj** call, you must issue a **dsmEndSendObj** call before another **dsmSendObj** call can be made. If multiple data sends were required via the **dsmSendData** call, then the **dsmEndSendObj** follows the last send to indicate the state change.

Note: If ADSM returns code 157 (`DSM_RC_WILL_ABORT`), issue a call to **dsmEndTxn** with a vote of `DSM_VOTE_COMMIT`. The application should then get back return code 2302 (`DSM_RC_CHECK_REASON_CODE`) and pass the reason code back to the application user. This will tell the user why the server is aborting 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 deter-
| mine a more accurate *sizeEstimate* and send the data again.

dsmSendObj

Syntax

```
int16 dsmSendObj
  (uint32       dsmHandle,
   dsmSendType  sendType,
   void        *sendBuff,
   dsmObjName  *objNameP,
   ObjAttr     *objAttrPtr,
   DataBlk     *dataBlkPtr);
```

Parameters

uint32 dsmHandle (I)
This parameter is the handle that associates this call with a previous **dsmInit** call.

dsmSendType sendType (I)
This parameter specifies the type of send being performed. Possible values are:

stBackup Specifies that this is a backup object being sent to the server

stArchive Specifies that this is an archive object being sent to the server

stBackupMountWait
Specifies that this is a backup object for which you want the server to wait until the necessary device, such as a tape, is mounted

stArchiveMountWait
Specifies that this is 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 your application user may 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 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 “Identifying the Object” on page 41 for more information.

In a backup operation, the object type specifies whether the object is a file or a directory. In an archive operation, the object type must be a file.

ObjAttr *objAttrPtr (I)
This parameter is used to pass object attributes of interest to the application. See Appendix A, “API Type Definitions Source File” on page 91 for the type definition.

The attributes of particular interest are:

- *owner* — This attribute refers to the owner of the object. Whether the owner is declared to be a specific name or an empty string is important when getting the object back from ADSM storage. See “Setting the Owner Name” on page 43 for more information.

- *sizeEstimate* — The size estimate attribute is a best estimate of the total size of the data object to be sent to the server. *It is important that you try to be as accurate as possible* on this size, because the ADSM server uses this attribute for efficient space allocation and object placement within its storage resources. Not specifying this value could result in performance degradation on the server, because allocations will have to be made on a best guess basis.

  | If the size estimate specified is significantly smaller than the actual number of
  | bytes sent, the server may have difficulty allocating enough space and abort
  | the transaction with a reason code 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` will not be compressed.

  | If your object will have no bit data (only the attribute information from this call),
  | the *sizeEstimate* should be 0. Otherwise, the transaction will abort when going
  | to an ADSM Version 1 server.

  | The *sizeEstimate* value is not returned on a query. If you need this informa-
  | tion for your application, save the value in the *objInfo* area.

- *objCompressed* — This attribute is a boolean value that states whether or not the object data has already been compressed.

  If the object is compressed, ADSM does not try to compress it again. If it is not compressed, then ADSM decides whether to compress the object, based on the values of the COMPRESSION option set by the ADSM administrator and set in the API configuration file and the client option files (**dsm.sys** in UNIX and **dsm.opt** in Windows).

  | 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* — This attribute can be used to save information about the particular object. Note that no information is stored here automatically. When this attribute is used, the attribute `objInfoLength` must also be set to show the length of `objInfo`.

- *mcNameP* — This attribute contains the name of a management class that overrides the default management class 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 wish 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 actually transferred. See Appendix A, “API Type Definitions Source File” on page 91 for the type definition.

Return Codes

Figure 43. Return Codes for dsmSendObj

Return Code	Explanation
DSM_RC_NO_COMPRESS_MEMORY (154)	Insufficient memory available to do data compression or expansion.
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 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 than the ADSM library version.
DSM_RC_NEEDTO_ENDTXN (2070)	Need to end transaction.
DSM_RC_OBJ_EXCLUDED (2080)	The object is excluded by the include-exclude list.
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 will not be 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.

dsmTerminate

Use **dsmTerminate** to end a session with the ADSM server and clean up the ADSM environment. All ADSM API calls must occur within the bounds of a **dsmInit** and a **dsmTerminate** call with the exception of the **dsmQueryApiVersion** call.

Syntax

```
int16 dsmTerminate
  (uint32 dsmHandle);
```

Parameters

uint32 dsmHandle (I)
This parameter is the handle that associates this call with a previous **dsmInit** call.

Return Codes

There are no return codes specifically for this call.

dsmUpdateFS

Use **dsmUpdateFS** to update a file space in ADSM storage. This ensures that the ADSM administrator has an up-to-date record of your file space.

Syntax

```
int16 dsmUpdateFS
  (uint32      dsmHandle,
   char       *fs,
   dsmFSUpd   *fsUpdP,
   uint32      fsUpdAct);
```

Parameters

uint32 dsmHandle (I)
This parameter is the handle that associates this call with a previous **dsmInit** 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 which has the proper fields filled in for the update desired. Only those fields needing update plus the `fsName` need be filled in for the update.

uint32 fsUpdAct (I)
A two byte bit map indicating which of the above fields are to be updated. See the DSM_FSUPD definitions in Appendix A, "API Type Definitions Source File" on page 91 for a description of these bit masks.

Return Codes

Figure 44. 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 than the ADSM library version.
DSM_RC_FSINFO_TOOLONG (2106)	File space information is too long.

Appendix A. API Type Definitions Source File

This section contains structure definitions, type definitions, and various constants for the ADSM API. This is a copy of the header file **dsmapitd.h** used by the API.

```
/*********************************************************************
* ADSTAR Distributed Storage Manager (ADSTAR DSM)                    *
* API Client Component                                               *
*                                                                    *
* (C) Copyright IBM Corporation 1993, 1995                           *
*********************************************************************/

/************************************************************************/
/* Header File Name: dsmapitd.h                                         */
/*                                                                      */
/* Descriptive-name: Definitions for ADSTAR DSM external constants      */
/************************************************************************/
#ifndef _H_DSMAPITD
#define _H_DSMAPITD
/*=====================================================================*/
/*                    D E F I N E S                                    */
/*=====================================================================*/
/*---------------------------------------------------------------------+
|  API Version, Release, and Level to use in dsmApiVersion on dsmInit()  |
+---------------------------------------------------------------------*/
#define DSM_API_VERSION     2
#define DSM_API_RELEASE     1
#define DSM_API_LEVEL       5

/*---------------------------------------------------------------------+
| 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_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_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*/

/*-------------------------------------------------------------------------+
| Minimum field lengths                                                    |
+-------------------------------------------------------------------------*/
#define DSM_MIN_COMPRESS_SIZE  2048 /* minimum number of bytes an object  */
                                    /* needs before compression is allowed*/

/*-------------------------------------------------------------------------+
|  Values for object type in dsmObjName structure                          |
+-------------------------------------------------------------------------*/
#define DSM_OBJ_FILE        0x01        /* object has attrib info and data*/
#define DSM_OBJ_DIRECTORY   0x02        /* object has only attribute info */
#define DSM_OBJ_ANY_TYPE    0xFF        /* for use on query               */

/*-------------------------------------------------------------------------+
| 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_OCCUPANCY       ((unsigned) 0x00000020)
#define DSM_FSUPD_CAPACITY        ((unsigned) 0x00000040)

/*-------------------------------------------------------------------------+
|  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 */
#define 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            */

/*-------------------------------------------------------------------+
| Definitions for "media access class" 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 local, on-line, fixed media (such as
          hard disks). Represents minimal delays in retrieval. */
#define  MEDIA_FIXED          0x10

/* Library: represents the class of local, mountable media accessible
            through a mechanical mounting device, in which there are
            typically small variations in mount time. */
#define  MEDIA_LIBRARY        0x20

/* Network: represents storage media accessible via a network server. */
#define  MEDIA_NETWORK        0x30

/* Shelf: represents the class of local, mountable media accessible only
          via human intervention. There can be large variations in the
          mount time. */
#define  MEDIA_SHELF          0x40
```

```
/* Offsite: represents media stored in an off-site location. The
            media are not accessible via local mounting procedures
            or direct attachment through a network. */
#define  MEDIA_OFFSITE        0x50

/* Unavailable: represents media that are (for whatever reason)
                inaccessible for retrieval. */
#define  MEDIA_UNAVAILABLE    0xF0

/*======================================================================*/
/*                    T Y P E D E F S                                   */
/*======================================================================*/

/* "Undefine" the keyword "signed" for the SunOS non-ANSI compiler. */
#ifdef _SUN_NONANSI_
#define signed
#endif

/*----------------------------------------------------------------------+
|  Typedefs for different integer types                                 |
+----------------------------------------------------------------------*/
/* The alpha processor of Digital Equipment Corporation is the only */
/* platform currently supported that has a 64 bit architecture.     */
/* Therefore the '(u)int32' data types are 'int' and not 'long'.    */
#ifndef __alpha

typedef signed   char     int8   ;
typedef unsigned char     uint8  ;
typedef signed   short    int16  ;
typedef unsigned short    uint16 ;
typedef signed   long     int32  ;
typedef unsigned long     uint32 ;

typedef struct
{
   uint32   hi;                       /* Most significant 32 bits.        */
   uint32   lo;                       /* Least significant 32 bits.       */
} uint64 ;

#else

typedef signed   char     int8   ;
typedef unsigned char     uint8  ;
typedef signed   short    int16  ;
typedef unsigned short    uint16 ;
typedef signed   int      int32  ;
typedef unsigned int      uint32 ;

typedef struct
{
   uint32   hi;                       /* Most significant 32 bits.        */
   uint32   lo;                       /* Least significant 32 bits.       */
} uint64 ;

#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
{
        bFalse = 0x00,
        bTrue  = 0x01
} bool_t ;

/*-------------------------------------------------------------------------+
|  Type definition for date structure                                      |
+-------------------------------------------------------------------------*/
typedef struct
{
    uint16    year;                  /* year, 16-bit integer (e.g., 1990) */
    uint8     month;                 /* month, 8-bit integer (1 - 12)     */
    uint8     day;                   /* day. 8-bit integer (1 - 31)       */
    uint8     hour;                  /* hour, 8-bit integer (0 - 23)      */
    uint8     minute;                /* minute, 8-bit integer (0 - 59)    */
    uint8     second;                /* second, b-bit integer (0 - 59)    */
} dsmDate ;

/*-------------------------------------------------------------------------+
|  Type definition for Object ID on dsmGetObj() and in dsmGetList structure|
+-------------------------------------------------------------------------*/
typedef uint64  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 */
} 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 */
} dsmDelType ;

/*-----------------------------------------------------------------------+
|  Type definition for API Version on dsmInit() and dsmQueryApiVersion()   |
+-----------------------------------------------------------------------*/
typedef struct
{
        unsigned short version;       /* API version                     */
        unsigned short release;       /* API release                     */
        unsigned short level;         /* API level                       */
} dsmApiVersion;

/*-----------------------------------------------------------------------+
|  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 */
    uint8  objType;          /* for object type values, see defines above */
} dsmObjName;
```

```
/*-------------------------------------------------------------------------+
|  Type definition for Backup delete info on dsmDeleteObj()                |
+-------------------------------------------------------------------------*/
typedef struct
{
    uint16       stVersion ;                   /* structure version      */
    dsmObjName   *objNameP ;                   /* object name            */
    uint32       copyGroup ;                   /* copy group             */
} delBack ;

#define  delBackVersion    1

/*-------------------------------------------------------------------------+
|  Type definition for Archive delete info on dsmDeleteObj()               |
+-------------------------------------------------------------------------*/
typedef struct
{
    uint16       stVersion ;                   /* structure version      */
    uint64       objId ;                       /* object ID              */
} delArch ;

#define  delArchVersion  1

/*-------------------------------------------------------------------------+
|  Type definition for delete info on dsmDeleteObj()                       |
+-------------------------------------------------------------------------*/
typedef union
{
    delBack   backInfo ;
    delArch   archInfo ;
} dsmDelInfo ;

/*-------------------------------------------------------------------------+
|  Type definition for Object Attribute parameter on dsmSendObj()          |
+-------------------------------------------------------------------------*/
typedef struct
{
    uint16   stVersion;                              /* Structure version */
    char     owner[DSM_MAX_OWNER_LENGTH + 1];            /* object owner */
    uint64   sizeEstimate;        /* Size estimate in bytes of the object */
    bool_t   objCompressed;             /* Is object already compressed? */
    uint16   objInfoLength;           /* length of object-dependent info */
    char     *objInfo;                          /* object-dependent info */
    char     *mcNameP;                     /* mgmt class name for override */
}ObjAttr;

#define ObjAttrVersion  2

/*-------------------------------------------------------------------------+
| Type definition for mcBindKey returned on dsmBindMC()                    |
+-------------------------------------------------------------------------*/
```

```
typedef struct
{
    uint16     stVersion;                   /* structure version          */
    char       mcName[DSM_MAX_MC_NAME_LENGTH + 1];
                                           /* Name of mc bound to object. */
    bool_t     backup_cg_exists;                            /* True/false */
    bool_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 partial object data for dsmBeginGetData()         |
+----------------------------------------------------------------------*/
typedef struct
{
    uint16   stVersion;          /* Structure version                     */
    uint64   partialObjOffset;   /* offset into object to begin reading   */
    uint64   partialObjLength;   /* amount of object to read              */
} PartialObjData ;               /* partial object data                   */

#define PartialObjDataVersion 1  /*                                       */

/*----------------------------------------------------------------------+
| Type definition for object list on dsmBeginGetData()                  |
+----------------------------------------------------------------------*/
typedef struct
{
    uint16   stVersion ;         /* structure version                     */
    uint32   numObjId ;          /* number of object IDs in the list      */
    ObjID    *objId    ;         /* list of object IDs to restore         */
    PartialObjData *partialObjData ; /* list of partial object data       */
} 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
{
    uint16 stVersion ;                    /* structure version            */
    uint32 bufferLen;                    /* Length of buffer passed below */
    uint32 numBytes;                  /* Actual number of bytes read from */
                                             /* or written to the buffer */
    char   *bufferPtr;                                     /* Data buffer */
} DataBlk;
```

```
#define DataBlkVersion  1

/*-------------------------------------------------------------------+
|  Type definition for Mgmt Class queryBuffer on dsmBeginQuery()     |
+-------------------------------------------------------------------*/
typedef struct S_qryMCData
{
    uint16   stVersion;                              /* structure version */
    char     *mcName;                                  /* Mgmt class name */
                                                /* single name to get one */
                                            /* or empty string to get all */
    bool_t   mcDetail;                            /* Want details or not? */
} qryMCData;

#define qryMCDataVersion  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 */
     uint16   frequency;                       /* Copy (archive) frequency */
     uint16   retainVers;                                /* Retain version */
     uint8    copySer;       /* for copy serialization values, see defines */
     uint8    copyMode;         /* for copy mode values, see defines above */
     char     destName[DSM_MAX_CG_DEST_LENGTH + 1];      /* Copy dest name */
} 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 */
    uint16   frequency;                               /* Backup frequency */
    uint16   verDataExst;                         /* Versions data exists */
    uint16   verDataDltd;                        /* Versions data deleted */
    uint16   retXtraVers;                        /* Retain extra versions */
    uint16   retOnlyVers;                         /* Retain only versions */
    uint8    copySer;       /* for copy serialization values, see defines */
    uint8    copyMode;         /* for copy mode values, see defines above */
    char     destName[DSM_MAX_CG_DEST_LENGTH + 1];      /* Copy dest name */
} backupDetailCG;

/*-------------------------------------------------------------------+
|  Type definition for Query Mgmt Class detail response on dsmGetNextQObj()|
+-------------------------------------------------------------------*/
typedef struct S_qryRespMCDetailData
{
    uint16          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  1

/*-------------------------------------------------------------------------+
| Type definition for Query Mgmt Class summary response on dsmGetNextQObj()|
+-------------------------------------------------------------------------*/
typedef struct S_qryRespMCData
{
    uint16   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
{
    uint16       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

/*-------------------------------------------------------------------------+
| Type definition for Query Archive response on dsmGetNextQObj()           |
+-------------------------------------------------------------------------*/
typedef struct S_qryRespArchiveData
{
    uint16      stVersion;                            /* structure version */
    dsmObjName  objName;                       /* Filespace name qualifier */
    uint32      copyGroup;                            /* copy group number */
    char        mcName[DSM_MAX_MC_NAME_LENGTH + 1];             /* mc name */
    char        owner[DSM_MAX_OWNER_LENGTH + 1];             /* owner name */
    uint64      objId;                                   /* Unique copy id */
    uint64      restoreOrder;                             /* restore order */
    uint8       mediaClass;                          /* media access class */
    dsmDate     insDate;                          /* archive insertion date */
    dsmDate     expDate;                    /* expiration date for object */
    char        descr[DSM_MAX_DESCR_LENGTH + 1];   /* archive description */
    uint16      objInfolen;             /* length of object-dependent info*/
    char        objInfo[DSM_MAX_OBJINFO_LENGTH];  /*object-dependent info */
} qryRespArchiveData;

#define qryRespArchiveDataVersion  1

/*-------------------------------------------------------------------------+
|  Type definition for Archive sendBuff parameter on dsmSendObj()          |
+-------------------------------------------------------------------------*/
typedef struct S_sndArchiveData
{
    uint16   stVersion;                             /* structure version */
    char     *descr;                              /* archive description */
} sndArchiveData;

#define sndArchiveDataVersion  1

/*-------------------------------------------------------------------------+
|  Type definition for Backup queryBuffer on dsmBeginQuery()               |
+-------------------------------------------------------------------------*/
typedef struct S_qryBackupData
{
    uint16      stVersion;                          /* structure version */
    dsmObjName  *objName;                     /* full dsm name of object */
    char        *owner;                                    /* owner name */
    uint8       objState;                       /* object state selector */
                                 /* for possible values, see defines above */
} qryBackupData;

#define qryBackupDataVersion  1

/*-------------------------------------------------------------------------+
| Type definition for Query Backup response on dsmGetNextQObj()            |
+-------------------------------------------------------------------------*/
typedef struct S_qryRespBackupData
{
    uint16      stVersion;                          /* structure version */
    dsmObjName  objName;                      /* full dsm name of object */
    uint32      copyGroup;                          /* copy group number */
    char        mcName[DSM_MAX_MC_NAME_LENGTH + 1];           /* mc name */
    char        owner[DSM_MAX_OWNER_LENGTH + 1];           /* owner name */
    uint64      objId;                               /* Unique object id */
    uint64      restoreOrder;                           /* restore order */
    uint8       mediaClass;                        /* media access class */
    uint8       objState;                     /* Obj state, active, etc. */
    dsmDate     insDate;                        /* backup insertion date */
    dsmDate     expDate;                    /* expiration date for object */
    uint16      objInfolen;             /* length of object-dependent info*/
    char        objInfo[DSM_MAX_OBJINFO_LENGTH];  /*object-dependent info */
} qryRespBackupData;
```

```
#define qryRespBackupDataVersion  1

/*----------------------------------------------------------------------+
|  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
{
    uint16      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
{
    uint16      stVersion;                           /* structure version */
    dsmObjName  objName;                      /* full dsm name of object */
    uint32      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 */
    uint16      objInfolen;             /* length of object-dependent info*/
    char        objInfo[DSM_MAX_OBJINFO_LENGTH];  /*object-dependent info */
} qryARespBackupData;

#define qryARespBackupDataVersion  1

/*----------------------------------------------------------------------+
|  Type definition for DOS Filespace attributes                            |
+----------------------------------------------------------------------*/
typedef struct
{
    char         driveLetter ;          /* drive letter for filespace    */
    uint16       fsInfoLength;          /* fsInfo length used            */
    char         fsInfo[DSM_MAX_FSINFO_LENGTH];/*caller-determined data  */
} dsmDosFSAttrib ;

/*----------------------------------------------------------------------+
|  Type definition for UNIX Filespace attributes                           |
+----------------------------------------------------------------------*/
typedef struct
{
    uint16       fsInfoLength;          /* fsInfo length used            */
    char         fsInfo[DSM_MAX_FSINFO_LENGTH];/*caller-determined data  */
} dsmUnixFSAttrib ;

/*---------------------------------------------------------------------+
|  Type definition for Filespace attributes on all Filespace calls        |
+---------------------------------------------------------------------*/
typedef union
{
    dsmDosFSAttrib   dosFSAttr ;
    dsmUnixFSAttrib  unixFSAttr ;
} dsmFSAttr ;

/*---------------------------------------------------------------------+
|  Type definition for fsUpd parameter on dsmUpdateFS()
+---------------------------------------------------------------------*/
typedef struct S_dsmFSUpd
{
    uint16      stVersion ;            /* structure version              */
    char        *fsType ;              /* filespace type                 */
    uint64      occupancy ;            /* occupancy estimate             */
    uint64      capacity  ;            /* capacity estimate              */
    dsmFSAttr   fsAttr ;               /* platform specific attributes   */
} dsmFSUpd ;

#define dsmFSUpdVersion  1

/*---------------------------------------------------------------------+
|  Type definition for Filespace queryBuffer on dsmBeginQuery()           |
+---------------------------------------------------------------------*/
typedef struct S_qryFSData
{
    uint16  stVersion;                         /* structure version */
    char    *fsName;                           /* File space name */
} qryFSData;

#define qryFSDataVersion  1

/*---------------------------------------------------------------------+
| Type definition for Query Filespace response on dsmGetNextQObj()        |
+---------------------------------------------------------------------*/
typedef struct S_qryRespFSData
{
    uint16   stVersion;                             /* structure version */
    char     fsName[DSM_MAX_FSNAME_LENGTH + 1];        /* Filespace name */
    char     fsType[DSM_MAX_FSTYPE_LENGTH + 1] ;       /* Filespace type */
    uint64   occupancy;                       /* Occupancy est. in bytes. */
    uint64   capacity;                         /* Capacity est. in bytes. */
    dsmFSAttr fsAttr ;                        /*general fs info or attribs*/
} qryRespFSData;

#define qryRespFSDataVersion  1
```

```
/*-------------------------------------------------------------------------+
|  Type definition for regFilespace parameter on dsmRegisterFS()
+-------------------------------------------------------------------------*/
typedef struct S_regFSData
{
    uint16    stVersion;                         /* structure version */
    char     *fsName;                               /* Filespace name */
    char     *fsType;                               /* Filespace type */
    uint64    occupancy;                  /* Occupancy est. in bytes. */
    uint64    capacity;                    /* Capacity est. in bytes. */
    dsmFSAttr fsAttr ;                    /*general fs info or attribs*/
} regFSData;

#define regFSDataVersion  1

/*-------------------------------------------------------------------------+
|  Type definition for session info response on dsmQuerySessionInfo()      |
+-------------------------------------------------------------------------*/
typedef struct
{
  uint16     stVersion;             /* Structure version                */
     /*------------------------------------------------------------------*/
     /*          Server information                                      */
     /*------------------------------------------------------------------*/
  char       serverHost[DSM_MAX_SERVERNAME_LENGTH+1];
                                    /* Network host name of DSM server  */
  uint16     serverPort;            /* Server comm port on host         */
  dsmDate    serverDate;            /* Server's date/time               */
  char       serverType[DSM_MAX_SERVERTYPE_LENGTH+1];
                                    /* Server's execution platform      */
  uint16     serverVer;             /* Server's version number          */
  uint16     serverRel;             /* Server's release number          */
  uint16     serverLev;             /* Server's level number            */
  uint16     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 and lowlev*/
  uint8      compression;           /* Compression flag                 */
  uint8      archDel;               /* Archive delete permission        */
  uint8      backDel;               /* Backup delete permission         */
  uint32     maxBytesPerTxn;        /* for future use                   */
  uint16     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  */
  uint8      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  */
  uint16     gpBackRetn;            /* Grace-period backup retention    */
  uint16     gpArchRetn;            /* Grace-period archive retention   */
} ApiSessInfo;

#define ApiSessInfoVersion  1


#endif /* _H_DSMAPITD */
```

Appendix B. API Function Definitions Source File

This section contains the function definitions for the ADSM API. This is a copy of the header file **dsmapifp.h** used in the UNIX version of the product. The Windows, OS/2, and Novell versions differ slightly, and application programmers for these platforms should check their version of **dsmapifp.h** for the exact syntax of the API functions.

```
/*********************************************************************
* ADSTAR Distributed Storage Manager (ADSTAR DSM)                    *
* API Client Component                                               *
*                                                                    *
* (C) Copyright IBM Corporation 1993, 1995                           *
*********************************************************************/

/*************************************************************************/
/* Header File Name: dsmapifp.h                                          */
/*                                                                       */
/* Descriptive-name: ADSTAR DSM API function prototypes                  */
/*************************************************************************/
#ifndef _H_DSMAPIFP
#define _H_DSMAPIFP
/*=======================================================================*/
/*               P U B L I C   F U N C T I O N S                         */
/*=======================================================================*/

#ifdef _NO_PROTO_

extern int16      dsmBeginGetData();
extern int16      dsmBeginQuery();
extern int16      dsmBeginTxn();
extern int16      dsmBindMC();
extern int16      dsmChangePW();
extern int16      dsmDeleteObj();
extern int16      dsmDeleteFS();
extern int16      dsmEndGetData();
extern int16      dsmEndGetObj();
extern int16      dsmEndQuery();
extern int16      dsmEndSendObj();
extern int16      dsmEndTxn();
extern int16      dsmGetData();
extern int16      dsmGetNextQObj();
extern int16      dsmGetObj();
extern int16      dsmInit();
extern void       dsmQueryApiVersion();
extern int16      dsmQuerySessInfo();
extern int16      dsmRCMsg();
extern int16      dsmRegisterFS();
extern int16      dsmSendData();
extern int16      dsmSendObj();
extern int16      dsmTerminate();
extern int16      dsmUpdateFS();

#else  /* use ANSI C required prototypes */

extern int16      dsmBeginGetData(
       uint32               dsmHandle,
       bool_t               mountWait,
       dsmGetType           getType,
       dsmGetList            *dsmGetObjListP
);
extern int16      dsmBeginQuery(
       uint32                dsmHandle,
       dsmQueryType          queryType,
       dsmQueryBuff          *queryBuffer
);
extern int16      dsmBeginTxn(
       uint32                dsmHandle
);
extern int16      dsmBindMC(
       uint32                dsmHandle,
       dsmObjName            *objNameP,
       dsmSendType           sendType,
       mcBindKey             *mcBindKeyP
);
extern int16      dsmChangePW(
       uint32                dsmHandle,
       char                  *oldPW,
       char                  *newPW
);
extern int16      dsmDeleteObj(
       uint32                dsmHandle,
       dsmDelType            delType,
       dsmDelInfo            delInfo
);
extern int16      dsmDeleteFS(
       uint32                dsmHandle,
       char                  *fsName,
       unsigned char         repository
);
extern int16      dsmEndGetData(
       uint32                dsmHandle
);
extern int16      dsmEndGetObj(
       uint32                dsmHandle
);
extern int16      dsmEndQuery(
       uint32                dsmHandle
);
extern int16      dsmEndSendObj(
       uint32                dsmHandle
);
extern int16      dsmEndTxn(
       uint32                dsmHandle,
       uint8                 vote,
       uint16                *reason
);
extern int16      dsmGetData(
       uint32                dsmHandle,
       DataBlk               *dataBlkPtr
);
extern int16      dsmGetNextQObj(
       uint32                dsmHandle,
       DataBlk               *dataBlkPtr
) ;
extern int16      dsmGetObj(
       uint32                dsmHandle,
       ObjID                 *objIdP,
       DataBlk               *dataBlkPtr
);
extern int16      dsmInit(
       uint32                *dsmHandle,
       dsmApiVersion         *dsmApiVersionP,
       char                  *clientNodeNameP,
       char                  *clientOwnerNameP,
       char                  *clientPasswordP,
       char                  *applicationType,
       char                  *configfile,
       char                  *options
);
extern void        dsmQueryApiVersion(
       dsmApiVersion         *apiVersionP
);
extern int16       dsmQuerySessInfo(
       uint32                dsmHandle,
       ApiSessInfo           *SessInfoP
);
extern int16       dsmRCMsg(
       uint32                dsmHandle,
       int16                 dsmRC,
       char                  *msg
);
extern int16       dsmRegisterFS(
       uint32                dsmHandle,
       regFSData             *regFilespaceP
);
extern int16       dsmSendData(
       uint32                dsmHandle,
       DataBlk               *dataBlkPtr
) ;
extern int16       dsmSendObj(
       uint32                dsmHandle,
       dsmSendType           sendType,
       void                  *sendBuff,
       dsmObjName            *objNameP,
       ObjAttr               *objAttrPtr,
       DataBlk               *dataBlkPtr
);
extern int16       dsmTerminate(
       uint32                dsmHandle
);
extern int16       dsmUpdateFS(
       uint32                dsmHandle,
       char                  *fs,
       dsmFSUpd              *fsUpdP,
       uint32                fsUpdAct
);

#endif /* NO _PROTOTYPE_  */

#endif /* _H_DSMAPIFP */
```

Appendix C. API Return Codes Source File

The following is a list of the possible return codes from the APIs. This is a copy of the header file **dsmrc.h** used in the product.

The return codes are explained in more detail in Appendix D, “API Return Codes With Explanations” on page 123.

```
/************************************************************************
* ADSTAR Distributed Storage Manager (ADSTAR DSM)                       *
* API Client Component                                                  *
*                                                                       *
* (C) Copyright IBM Corporation 1993, 1995                              *
************************************************************************/

/***********************************************************************/
/* Header File Name:  dsmrc.h                                          */
/*                                                                     */
/* Descriptive-name:  Return codes from ADSTAR DSM APIs                */
/***********************************************************************/
#ifndef _H_DSMRC
#define _H_DSMRC

typedef int RetCode ;

#define DSM_RC_SUCCESSFUL                0 /* successful completion   */
#define DSM_RC_OK                        0 /* successful 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_INVALID_OFFSET          33    /* Partial Object Retrieve */
#define DSM_RS_ABORT_INVALID_LENGTH          34    /* Partial Object Retrieve */


/* 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_INVALID_OFFSET          DSM_RS_ABORT_INVALID_OFFSET
#define DSM_RC_ABORT_INVALID_LENGTH          DSM_RS_ABORT_INVALID_LENGTH

/* 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_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        103 /* communications link down          */
#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_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_DBCS_IN_RANGE       176 /*DBCS character not allowed within  */
#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.                            */

/*----------------------------------------------------------------------*/
/* Return codes for the Trusted Communication Agent                      */
/*----------------------------------------------------------------------*/
#define DSM_RC_TCA_ATTACH_SHR_MEM_ERR 200 /* Error attaching shared memory   */
#define DSM_RC_TCA_SHR_MEM_BLOCK_ERR 201 /* Shared memory block invalid      */
#define DSM_RC_TCA_SHR_MEM_IN_USE  202 /* Shared memory already in use       */
#define DSM_RC_TCA_SHARED_MEMORY_ERROR 291 /* Error alloc TCA shared memory  */
#define DSM_RC_TCA_FORK_FAILED     292 /* Error forking off TCA process      */
#define DSM_RC_TCA_SEGMENT_MISMATCH 293 /* Shared memory segs don't match    */
#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_NOT_ROOT        296 /* Invalid action for non-root user   */
#define DSM_RC_TCA_SEMGET_ERROR    297 /* Error getting semaphores           */
#define DSM_RC_TCA_SEM_OP_ERROR    298 /* Error in semaphore set or wait     */

/*----------------------------------------------------------------------*/
/* 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    */
```

```
/* 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 */

/* Comm3270 error codes */
#define DSM_RC_COMM_TIMEOUT       -101 /* Communication timeout              */
#define DSM_RC_EMULATOR_INACTIVE  -102 /* Emulator inactive or not responding*/
#define DSM_RC_BAD_HOST_ID        -103 /* Host session id is invalid         */
#define DSM_RC_HOST_SESS_BUSY     -104 /* Another OS/2 HLLAPI appl has sess. */
#define DSM_RC_3270_CONNECT_FAILURE -105 /* Could not startup host
                                            session side                     */
#define DSM_RC_NO_ACS3ELKE_DLL    -106 /* The ACSNETB.DLL could not be loaded*/
#define DSM_RC_EMULATOR_ERROR     -107 /* Emulator error detected            */
#define DSM_RC_EMULATOR_BACKLEVEL -108 /* Emulator error detected            */
#define DSM_RC_CKSUM_FAILURE      -109 /* 3270 cksum failed, pkt too big or  */
                                       /*   just plain got bad data.         */

/* The following Return codes are for EHLLAPI for Windows                    */
#define DSM_RC_3270COMMError_DLL        -110
#define DSM_RC_3270COMMError_GetProc    -111
#define DSM_RC_EHLLAPIError_DLL         -112
#define DSM_RC_EHLLAPIError_GetProc     -113
#define DSM_RC_EHLLAPIError_HostConnect -114
#define DSM_RC_EHLLAPIError_AllocBuff   -115
#define DSM_RC_EHLLAPIError_SendKey     -116
#define DSM_RC_EHLLAPIError_PacketChk   -117
#define DSM_RC_EHLLAPIError_ChkSum      -118
#define DSM_RC_EHLLAPIError_HostTimeOut -119
#define DSM_RC_EHLLAPIError_Send        -120
#define DSM_RC_EHLLAPIError_Recv        -121
#define DSM_RC_EHLLAPIError_General     -122
#define DSM_RC_PC3270_MISSING_DLL       -123
#define DSM_RC_3270COMM_MISSING_DLL     -124

/* 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           -201
#define DSM_RC_CPIC_TYPE_MISMATCH              -202
#define DSM_RC_CPIC_PIP_NOT_SPECIFY_ERR        -203
#define DSM_RC_CPIC_SECURITY_NOT_VALID         -204
#define DSM_RC_CPIC_SYNC_LVL_NO_SUPPORT        -205
#define DSM_RC_CPIC_TPN_NOT_RECOGNIZED         -206
#define DSM_RC_CPIC_TP_ERROR                   -207
#define DSM_RC_CPIC_PARAMETER_ERROR            -208
#define DSM_RC_CPIC_PROD_SPECIFIC_ERR          -209
#define DSM_RC_CPIC_PROGRAM_ERROR              -210
#define DSM_RC_CPIC_RESOURCE_ERROR             -211
#define DSM_RC_CPIC_DEALLOCATE_ERROR           -212
#define DSM_RC_CPIC_SVC_ERROR                  -213
#define DSM_RC_CPIC_PROGRAM_STATE_CHECK        -214
#define DSM_RC_CPIC_PROGRAM_PARAM_CHECK        -215
#define DSM_RC_CPIC_UNSUCCESSFUL               -216
#define DSM_RC_UNKNOWN_CPIC_PROBLEM            -217
#define DSM_RC_CPIC_MISSING_LU                 -218
#define DSM_RC_CPIC_MISSING_TP                 -219
#define DSM_RC_CPIC_MISSING_DLL                -220
#define DSM_RC_CPIC_DLL_LOADFAILURE            -221
#define DSM_RC_CPIC_FUNC_LOADFAILURE           -222


#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_OPT         2013 /* invalid option                   */
#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_BAD_CALL_SEQUENCE   2041 /* Sequence of DSM calls not allowed*/

#define DSM_RC_WILDCHAR_NOTALLOWED 2050 /* Wild card not allowed for hl,ll  */

#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_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_OPT_FILE         2220 /*No default user configuration file*/
#define DSM_RC_INVALID_KEYWORD     2221 /* Invalid option keyword           */
#define DSM_RC_PATTERN_TOO_COMPLEX 2222 /* Can't match Include/Exclude entry*/
#define DSM_RC_NO_CLOSING_BRACKET  2223 /* Missing closing bracket inc/excl */
#define DSM_RC_INVALID_SERVER      2225 /* Invalid server name from client  */
#define DSM_RC_NO_HOST_ADDR        2226 /* Not enuf info to connect server  */
#define DSM_RC_MACHINE_SAME        2227 /* -MACHINENAME same as real name   */
#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 */

#endif /* _H_DSMRC */
```

Appendix D. API Return Codes With Explanations

This appendix describes in more detail what the return codes mean, and how you, as an application developer, should deal with them.

In this section, the return codes are listed in numerical order. For each return code, the following information is given:

- the return code *number* — This number corresponds to the number in the header file **dsmrc.h** (see Appendix C, "API Return Codes Source File" on page 113).
- the *severity code* — This letter is an indication of the severity of the situation that caused the return code to be generated. The possible severity codes and their meanings are:

S	Severe error	Processing could not continue.
E	Error	Processing could not continue.
W	Warning	Processing can continue, but problems may develop later. You should be cautious.
I	Information	Processing continues. No user response necessary.

- the *symbolic name* — This name corresponds to the definition in the header file **dsmrc.h**. *You should always use the symbolic name for a return code in your application rather than the return code number.*

  Note that the symbolic names are case sensitive. Most are entirely in upper case letters, but the ones within the range -110 to -122 also contain lower case letters.
- the *explanation* — This field explains the circumstances under which this return code might be generated.
- the *system action* — This field describes what action ADSM is going to take in response to the return code.
- the *user response* — This field explains what you should do in response to the system action.

Many of the return codes describe errors that cause processing to stop. You may want to send a message to the end user that describes the problem and suggests some course of action. To identify different messages, you can use these return code values or develop your own numbering system.

-050 E DSM_RC_TCPIP_FAILURE

Explanation: An attempt to connect to the server using TCP/IP communications failed. This error can occur if the LAN connection went down or if your system administrator canceled a backup operation.

System Action: Session rejected. Processing stopped.

User Response: Retry the operation, or wait until the server comes back up and retry the operation. If the problem continues, see your system administrator for further help.

-051 E DSM_RC_CONN_TIMEDOUT

Explanation: The attempt to establish a TCP/IP connection timed out before the connection was made.

System Action: Processing stopped.

User Response: Check for a networking problem. If the problem continues, see your system administrator.

-052 E DSM_RC_CONN_REFUSED

Explanation: An attempt to establish a TCP/IP connection was rejected by the server.

System Action: Processing stopped.

User Response: The server was not fully initialized, is not currently running, was not enabled for TCP/IP communications, or an incorrect TCP/IP port number was specified. If the problem continues, see your system administrator.

-053 E DSM_RC_BAD_HOST_NAME

Explanation: An invalid TCP/IP host name or address was specified.

System Action: Processing stopped.

User Response: Check your options file for the correct TCPSERVERADDRESS statement. See your administrator for the correct name of the server.

-054 E DSM_RC_NETWORK_UNREACHABLE

Explanation: The TCP/IP host name specified in the TCPSERVERADDRESS statement cannot be reached.

System Action: Processing stopped.

User Response: Check your options file for the correct TCPSERVERADDRESS statement. See your administrator for the correct name of the server.

-055 E DSM_RC_WINSOCK_MISSING

Explanation: The TCP/IP WINSOCK.DLL file cannot be found.

System Action: Processing stopped.

User Response: Verify your TCP/IP installation.

-056 E DSM_RC_TCPIP_DLL_LOADFAILURE

Explanation: An error occurred while loading a library. The TCP/IP DLL load failed.

System Action: Processing stopped.

User Response: Verify your TCP/IP installation.

-057 E DSM_RC_TCPIP_LOADFAILURE

Explanation: An error occurred while locating a function. The TCP/IP load function failed.

System Action: Processing stopped.

User Response: Verify your TCP/IP installation.

-101 E DSM_RC_COMM_TIMEOUT

Explanation: Explain the message. What caused it?

System Action: What did the system really do?

User Response: What should the user do?

-101 E DSM_RC_COMM_TIMEOUT

Explanation: Explain the message. What caused it?

System Action: What did the system really do?

User Response: What should the user do?

-101 E DSM_RC_COMM_TIMEOUT

Explanation: Explain the message. What caused it?

System Action: What did the system really do?

User Response: What should the user do?

-101 E DSM_RC_COMM_TIMEOUT

Explanation: Explain the message. What caused it?

System Action: What did the system really do?

User Response: What should the user do?

-102 E DSM_RC_EMULATOR_INACTIVE

Explanation: The 3270 terminal emulator was not active or cannot be accessed by ADSM. Ensure that your 3270 terminal emulator is active and that it is at the proper level as supported by ADSM.

System Action: ADSM cannot connect to the server. ADSM canceled the current operation.

User Response: Ensure that the emulator is active and at the proper release level.

-103 E DSM_RC_BAD_HOST_ID

Explanation: The 3270 emulator session cannot be accessed by ADSM.

System Action: ADSM cannot connect to the server. ADSM canceled the current operation.

User Response: Ensure that the correct short name is specified for the 3270 terminal session that ADSM is to access. The short name is usually shown in the lower left corner of the 3270 session. Valid names are single alphabetic characters from a through z.

-104 E DSM_RC_HOST_SESS_BUSY

Explanation: Another application (typically a HLLAPI application) is running on your machine and has access to the 3270 host session.

System Action: ADSM cannot connect to the server. ADSM canceled the current operation.

User Response: Try running ADSM again to see if the condition clears. If the condition remains, see what other applications are running on your system and suspend or end the one that is preventing ADSM from accessing the server.

-105 E DSM_RC_3270_CONNECT_FAILURE

Explanation: ADSM cannot access the specified 3270 host session to connect to the server. Host session might be down or in an incorrect state.

System Action: ADSM cannot connect to the server. ADSM canceled the current operation.

User Response: Check to see if your host system is active. If so, ensure that the 3270 terminal session is in the proper state to receive the 3270 startup command sequence. If the system fails, make sure the server is active.

-106 E DSM_RC_NO_WSOCK32_DLL

Explanation: The 32-Bit Windows Sockets DLL WSOCK32.DLL could not be located

System Action: Communications link is not established. Session rejected.

User Response: See your system administrator.

-106 E DSM_RC_NO_WSOCK32_DLL

Explanation: The 32-Bit Windows Sockets DLL WSOCK32.DLL could not be located

System Action: Communications link is not established. Session rejected.

User Response: See your system administrator.

-107 E DSM_RC_EMULATOR_ERROR

Explanation: An unknown error was returned from the 3270 emulator session to ADSM.

System Action: ADSM canceled the current operation. Session rejected.

User Response: Ensure that the 3270 session is functioning properly. If the problem continues, see your service representative for further problem determination.

-108 S DSM_RC_EMULATOR_BACKLEVEL

Explanation: You are running a back level version of the 3270 emulator that is not supported by ADSM.

System Action: ADSM cannot connect to the server. ADSM canceled the current operation.

User Response: Install the current version of the 3270 emulator.

-109 E DSM_RC_CKSUM_FAILURE

Explanation: Corrupted data was received over the 3270 connection.

System Action: The client ends its current activity with the server.

User Response: Check your connection path for errors. Verify that you have installed and set up your 3270 emulator properly. Also, ensure that your 3270BUFFERSIZE as specified in your ADSM options file is not too large for your site. Typically, values of 4,000 bytes are accepted without error, but values over 32,000 bytes might cause this error if your communication link does not support the larger sizes.

-110 E DSM_RC_3270COMMError_DLL

Explanation: An error occurs while loading the ADSM 3270 support file into memory. This error only applies when using 3270 communications.

System Action: The connection to the server fails.

User Response: Because this error may be caused by insufficient memory, shutdown applications in progress and retry the operation. If the problem persists, contact your service representative.

-111 E DSM_RC_3270COMMError_GetProc

Explanation: An error occurs while loading one or more functions from the ADSM 3270 support file. This error only applies when using 3270 communications.

System Action: The connection to the server fails.

User Response: Because this error may be caused by insufficient memory, shutdown applications in progress and retry the operation. If the problem persists, contact your service representative.

-112 E DSM_RC_EHLLAPIError_DLL

Explanation: An error occurs while loading the PC3270W v3.00 EHLLAPI support file. This error only applies when using 3270 communications.

System Action: The connection to the server fails.

User Response: Because this error may be caused by insufficient memory, shutdown applications in progress, and retry the operation. If the problem persists, contact your service representative.

-113 E DSM_RC_EHLLAPIError_GetProc

Explanation: An error occurs while loading one or more functions from the PC3270W v3.00 EHLLAPI support file. This error only applies when using 3270 communications.

System Action: The connection to the server fails.

User Response: Because this error may be caused by insufficient memory, shutdown applications in progress, and retry the operation. If the problem persists, contact your service representative.

-114 E DSM_RC_EHLLAPIError_HostConnect

Explanation: Connection to the ADSM server using 3270 communications fails.

System Action: The connection to the server fails.

User Response: Contact your service representative.

-115 E DSM_RC_EHLLAPIError_AllocBuff

Explanation: A 3270 communications buffer cannot be allocated. This is usually caused by insufficient memory.

System Action: The connection to the server fails.

User Response: Shutdown some applications in progress or reduce the communication buffer sizes by using the 3270BUFFERSIZE option. If the problem persists, contact your service representative.

-116 E DSM_RC_EHLLAPIError_SendKey

Explanation: An error occurs sending the host startup command specified in the 3270HOSTCOMMAND option to the specified host session.

System Action: The connection to the server fails.

User Response: Make sure that the specified emulator session is in the correct state (Input is not inhibited, and so on).

-117 E DSM_RC_EHLLAPIError_PacketChk

Explanation: A packet or checksum error occurs when receiving data from the server.

System Action: The connection to the server fails.

User Response: Retry the operation. If the problem persists, contact your service representative.

-118 E DSM_RC_EHLLAPIError_ChkSum

Explanation: A packet or checksum error occurs when receiving data from the server.

System Action: The client session is ended.

User Response: Retry the operation. If the problem persists, contact your service representative.

-119 E DSM_RC_EHLLAPIError_HostTimeOut

Explanation: The client timed out while waiting for data to arrive from the server.

System Action: The connection to the server fails.

User Response: If system response time is very slow, try increasing the timeout value with the 3270HOSTTIMEOUT option. Also, make sure the specified emulator session is in the correct state.

-120 E DSM_RC_EHLLAPIError_Send

Explanation: An error occurs while sending data to the server.

System Action: The connection to the server fails.

User Response: Contact your service representative.

-121 E DSM_RC_EHLLAPIError_Recv

Explanation: Error receiving data from the server.

System Action: The connection to the server fails.

User Response: Contact your service representative.

-122 E DSM_RC_EHLLAPIError_General

Explanation: A general 3270 communication error occurs. The connection to the server fails.

System Action: The communications link is not established.

User Response: Contact your service representative.

-123 E DSM_RC_PC3270_MISSING_DLL

Explanation: The PC/3270 EHLLAPI DLL pcshll.dll cannot be found in the user's path.

System Action: Communications link is not established.

User Response: Contact your system administrator. Make sure that the PS/3270 EHLLAPI DLLs are in a directory that is included in the user's path.

-124 E DSM_RC_3270COMM_MISSING_DLL

Explanation: The ADSM DLL dsm3270.dll cannot be found in the user's path.

System Action: Communications link is not established.

User Response: Make sure that the ADSM DLL dsm3270.dll is in a directory that is included in the user's path.

-151 E DSM_RC_NETB_ERROR

Explanation: The client cannot access the NETBIOS server.

System Action: Session is not established.

User Response: Ensure that NETBIOS is loaded and that the NETBIOS server is active. See your system administrator on status of the LAN.

-152 E DSM_RC_NETB_NO_DLL

Explanation: The OS/2 LAN file ACSNETB.DLL is not available to establish a communications link, or the file is incorrect.

System Action: Communications link is not established.

User Response: See your system administrator.

-155 E DSM_RC_NETB_LAN_ERR

Explanation: LAN communications failure detected while in session with server.

System Action: Communications link is not established.

User Response: See your system administrator.

-158 E DSM_RC_NETB_NAME_ERR

Explanation: An attempt to add the client NETBIOS name failed, or an attempt to call the server NETBIOS name failed.

System Action: Processing stopped.

User Response: Verify that your NETBIOSNAME option value is unique. Verify that your NETBIOSSERVERNAME option value is correct. Verify that the server has NETBIOS support running. If the problem continues, see your system administrator for further help.

-159 E DSM_RC_NETB_TIMEOUT

Explanation: A timeout occurred when transmitting data with the NETBIOS protocol.

System Action: Processing stopped.

User Response: Make sure the server is operational. You may need to increase the NETBIOSTIMEOUT value or use a value of 0 for no timeout. If the problem continues, see your system administrator for further help.

-160 E DSM_RC_NETB_NOTINST

Explanation: The product for NETBIOS is not installed.

System Action: Processing stopped.

User Response: Verify that the product for NETBIOS, such as the LAN Support Program, is installed. If the problem continues, see your system administrator for further help.

-161 E DSM_RC_NETB_REBOOT

Explanation: A DOS or Windows NETBIOS error has occurred which requires that the adapter be reset. This may be caused by a software installation or adapter configuration error.

System Action: Processing stopped.

User Response: Reboot the machine. If the error recurs, check the ADSM error log to find the NETBIOS return code. See your system administrator for help with your NETBIOS installation/configuration problem.

-190 E DSM_RC_NP_ERROR

Explanation: An attempt to connect to the server using Named Pipes communications failed. This might have occurred if an incorrect NAMEDPIPENAME was specified in the options files or if your system administrator canceled a backup operation.

System Action: Processing stopped.

User Response: Retry the operation, or wait until the server comes back up and retry the operation. Ensure that the value specified on the NAMEDPIPENAME option is the same as the one used by the ADSM server. If the problem continues, contact your system administrator for further help.

-201 S DSM_RC_CPIC_ALLOCATE_FAILURE

Explanation: ADSM client cannot allocate a CPIC conversation to the ADSM server.

System Action: Processing stopped.

User Response: Ensure that the symbolic destination name matches a valid side information table entry in your local communications program.

-202 S DSM_RC_CPIC_TYPE_MISMATCH

Explanation: An unexpected CPIC error occurred. There is a CPIC conversation type mismatch between the client and the ADSM server.

System Action: Processing stopped.

User Response: This is a program error. See your service representative.

-203 S DSM_RC_CPIC_PIP_NOT_SPECIFY_ERR

Explanation: An unexpected CPIC error occurred. The CPIC PIP is not specified correctly.

System Action: Processing stopped.

User Response: This is a program error. See your service representative.

-204 S DSM_RC_CPIC_SECURITY_NOT_VALID

Explanation: The conversation cannot be allocated because the CPIC security is not valid.

System Action: Processing stopped.

User Response: Change the security in the CPIC side information table.

-205 S DSM_RC_CPIC_SYNC_LVL_NO_SUPPORT

Explanation: The conversation cannot continue because the CPIC synchronization level of either the local or remote system is not supported.

System Action: Processing stopped.

User Response: ADSM defaults to no synchronization level. See your service representative.

-206 S DSM_RC_CPIC_TPN_NOT_RECOGNIZED

Explanation: CPIC transaction program name not recognized. ADSM cannot find the name of the ADSM server.

System Action: Processing stopped.

User Response: Ensure that you have the correct ADSM server name listed in your CPIC side information entry and that the ADSM server is running.

-207 S DSM_RC_CPIC_TP_ERROR

Explanation: CPIC transaction program not available. The ADSM server is not responding to the local program's request.

System Action: Processing stopped.

User Response: Ensure that the ADSM server is running and retry the command. Stop and start the local communications program.

-208 S DSM_RC_CPIC_PARAMETER_ERROR

Explanation: A parameter to a CPIC call was in error.

System Action: Processing stopped.

User Response: An invalid symbolic destination or invalid transaction program name was given. Check these values and retry the command.

-209 S DSM_RC_CPIC_PROD_SPECIFIC_ERR

Explanation: ADSM detected a CPIC product-specific error that occurred when making a CPIC call.

System Action: Processing stopped.

User Response: Ensure that LU6.2 is working properly at your installation. See your system administrator.

-210 S DSM_RC_CPIC_PROGRAM_ERROR

Explanation: An unexpected CPIC program error occurred.

System Action: Processing stopped.

User Response: This is a program error. See your service representative.

-211 S DSM_RC_CPIC_RESOURCE_ERROR

Explanation: CPIC resource error. The conversation ended prematurely.

System Action: Processing stopped.

User Response: Exit ADSM and retry the ADSM command.

-212 W DSM_RC_CPIC_DEALLOCATE_ERROR

Explanation: CPIC conversation deallocation error. ADSM received an incorrect return code from the deallocate verb. This error does not affect the ADSM program.

System Action: None.

User Response: None.

-213 S DSM_RC_CPIC_SVC_ERROR

Explanation: A CPIC SVC error occurred.

System Action: Processing stopped.

User Response: See your service representative.

-214 S DSM_RC_CPIC_PROGRAM_STATE_CHECK

Explanation: CPIC program state check.

System Action: Processing stopped.

User Response: This is a program error. See your service representative.

-215 S DSM_RC_CPIC_PROGRAM_PARAM_CHECK

Explanation: A parameter to a CPIC call was in error.

System Action: Processing stopped.

User Response: An invalid symbolic destination or invalid transaction program name was given. Check these values and retry the command.

-216 S DSM_RC_CPIC_UNSUCCESSFUL

Explanation: CPIC session not immediately available. All the sessions for SNA communications are in use.

System Action: Processing stopped.

User Response: Stop and start SNA communications and retry the command.

-217 E DSM_RC_UNKNOWN_CPIC_PROBLEM

Explanation: An unexpected CPI Communication error occurs.

System Action: Processing stops.

User Response: Contact your service representative.

-218 S DSM_RC_CPIC_MISSING_LU

Explanation: For CPIC, you must supply a PARtnerluname if no SYMbolicdestination is given.

System Action: Processing stopped.

User Response: Retry the operation using an LU name or a symbolic destination name.

-219 S DSM_RC_CPIC_MISSING_TP

Explanation: For CPIC, you must supply a TPname if no SYMbolicdestination is given.

System Action: Processing stopped.

User Response: Retry the operation using the TPname option or define the SYMbolicdestination option.

-220 E DSM_RC_CPIC_MISSING_DLL

Explanation: ADSM cannot find the PWSCS CPI Communication support file (ACPOCPIC.DLL) in the current search path. This message only applies when using PWSCS communications.

System Action: The connection to the server fails.

User Response: Shutdown Windows and place the directory where this file resides in the DOS path statement.

-221 E DSM_RC_CPIC_DLL_LOADFAILURE

Explanation: An error occurs while loading the PWSCS CPI Communication support file into memory. This error only applies when using PWSCS communications.

System Action: The connection to the server fails.

User Response: Because this may be caused by insufficient memory, shutdown applications in progress, and retry the operation. If the problem persists, contact your service representative.

-222 E DSM_RC_CPIC_FUNC_LOADFAILURE

Explanation: An error occurs while loading one or more functions from the PWSCS CPI Communication support file. This error only applies when using PWSCS communications.

System Action: The connection to the server fails.

User Response: Because this may be caused by insufficient memory, shutdown applications in progress, and retry the operation. If the problem persists, contact your service representative.

0000 I DSM_RC_SUCCESSFUL

Explanation: The operation successfully completed.

System Action: None.

User Response: None.

0000 I DSM_RC_OK

Explanation: The operation successfully completed.

System Action: None.

User Response: None.

0001 E DSM_RC_ABORT_SYSTEM_ERROR

Explanation: The server detected a system error and notified the clients.

System Action: Processing stopped.

User Response: See your system administrator for further information on server activity.

0002 E DSM_RC_ABORT_NO_MATCH

Explanation: No objects on the server match the query operation being performed.

System Action: Processing stopped.

User Response: Ensure the names are properly entered.

0003 E DSM_RC_ABORT_BY_CLIENT

Explanation: The client system ended the operation with the server and ended the current transaction.

System Action: Processing stopped.

User Response: Restart the session.

0004 W DSM_RC_ABORT_ACTIVE_NOT_FOUND

Explanation: ADSM did not find an active object flagged for expiration on the server. The object is marked as expired by another ADSM operation.

System Action: None.

User Response: None.

0005 E DSM_RC_ABORT_NO_DATA

Explanation: ADSM tried to do a restore or retrieve on an object that has no data associated with it.

System Action: ADSM ended the current operation.

User Response: See your system administrator to verify the problem. If the problem continues, see your system administrator.

0006 E DSM_RC_ABORT_BAD_VERIFIER

Explanation: You entered an incorrect password (verifier).

System Action: Processing stopped.

User Response: Retry the session with the correct password.

0007 E DSM_RC_ABORT_NODE_IN_USE

Explanation: The node you are running on is in use by another operation on the server. This might be from another client or from some activity on the server.

System Action: Processing stopped.

User Response: Retry the operation, or see your system administrator to see what other operations are running for your node.

0008 E DSM_RC_ABORT_EXPDATE_TOO_LOW

Explanation: Archive expiration date is too low, the date must be greater than today's date.

System Action: ADSM canceled the current operation.

User Response: Retry archiving the file with an expiration date that is higher than today's date.

0009 W DSM_RC_ABORT_DATA_OFFLINE

Explanation: For the restore or retrieve operation, one or more of the requested files must be recalled from offline storage media (generally tape). The wait time depends on your site's offline storage management policies.

System Action: ADSM waits for offline storage media to become available and then continues.

User Response: None.

0010 E DSM_RC_ABORT_EXCLUDED_BY_SIZE

Explanation: The object is too large. The configuration of the server does not have any data storage space that accepts the object.

System Action: File skipped.

User Response: See your system administrator to determine the maximum file (object) size for which your site's server is configured.

0011 E DSM_RC_ABORT_NO_REPOSIT_SPACE

Explanation: The server does not have any space available to store the object.

System Action: ADSM ended the current operation.

User Response: Inform your system administrator that a storage pool on the server is full.

0012 E DSM_RC_ABORT_MOUNT_NOT_POSSIBLE

Explanation: Server media mount not possible. The server timed out waiting for a mount of an offline volume.

System Action: File skipped.

User Response: Retry later when server volumes can be mounted.

0013 E DSM_RC_ABORT_SIZESTIMATE_EXCEED

Explanation: The total amount of data for a backup or archive operation exceeds the estimated size originally sent to the server for allocating data storage space. This happens when many files are growing by large amounts while the backup or archive operation is in session.

System Action: Processing stopped.

User Response: Retry the operation. If the problem continues, check what other processes are running on the client machine that are generating large amounts of data. Disable those operations while the backup or archive operation is taking place.

0014 E DSM_RC_ABORT_DATA_UNAVAILABLE

Explanation: The file data is currently unavailable on the server. A retrieve or restore operation was attempted. Possible causes are:

- Data was corrupted at the server
- Server found a read error
- File is temporarily involved in a reclaim operation at the server
- Server requested a tape volume that was marked unavailable.

System Action: Processing stopped.

User Response: Retry the operation. If the problem continues, see your system administrator to determine the problem from the server console or the activity log. Check whether any requests were made for a tape volume that was unavailable. A tape volume may be marked unavailable if prior read errors were encountered or the volume is checked out of the tape library.

0015 E DSM_RC_ABORT_RETRY

Explanation: Unexpected Retry request. The server found an error while writing data to the server's data storage.

System Action: Client retries the operation.

User Response: None.

0016 E DSM_RC_ABORT_NO_LOG_SPACE

Explanation: The server ran out of recovery log space.

System Action: ADSM ended the current operation.

User Response: This error is a temporary problem. Retry later or see your system administrator.

0017 E DSM_RC_ABORT_NO_DB_SPACE

Explanation: The server ran out of database space.

System Action: ADSM ended the current operation.

User Response: See your system administrator.

0018 E DSM_RC_ABORT_NO_MEMORY

Explanation: The server ran out of memory.

System Action: ADSM ended the current operation.

User Response: This is a temporary problem. Retry later or see your system administrator.

0020 E DSM_RC_ABORT_FS_NOT_DEFINED

Explanation: The specified file space does not exist on the server. Your system administrator deleted the file space or another client using your client's node name deleted it.

System Action: ADSM canceled the current operation.

User Response: Check the file space name to see if it is correct and retry the operation.

0021 S DSM_RC_ABORT_NODE_ALREADY_DEFED

Explanation: Open registration failed because a node is defined on the server with the same name.

System Action: ADSM canceled the current operation.

User Response: Retry with another node name.

0022 S DSM_RC_ABORT_NO_DEFAULT_DOMAIN

Explanation: Open registration failed because a default policy domain does not exist for you to place your node.

System Action: ADSM canceled the current operation.

User Response: See your system administrator.

0023 S DSM_RC_ABORT_INVALID_NODENAME

Explanation: Open registration failed because the specified node name contains invalid characters.

System Action: ADSM canceled the current operation.

User Response: Retry with another node name that does not have any invalid characters.

0024 S DSM_RC_ABORT_INVALID_POL_BIND

Explanation: Server problem. Invalid policy binding.

System Action: Processing stopped.

User Response: Have your service representative check the error log.

0025 E DSM_RC_ABORT_DEST_NOT_DEFINED

Explanation: Server problem: Destination not defined.

System Action: Processing stopped.

User Response: Have your service representative check the error log.

0026 S DSM_RC_ABORT_WAIT_FOR_SPACE

Explanation: The client received an unexpected Wait For Space message from the server.

System Action: ADSM ended the current operation.

User Response: See your system administrator.

0027 E DSM_RC_ABORT_NOT_AUTHORIZED

Explanation: During a delete filespace operation, you specified a file space to which your node does not have permission to delete archived data and/or backed up data.

System Action: Delete processing fails.

User Response: See your system administrator.

0028 E DSM_RS_ABORT_RULE_ALREADY_DEFED

Explanation: You are trying to define authorization for the specified node, which already has authorization defined.

System Action: ADSM did not redefine authorization for the specified node.

User Response: Update the authorization, or delete the old rule and define a new one, or use the current authorization.

0029 S DSM_RC_ABORT_NO_STOR_SPACE_STOP

Explanation: The server does not have space available to store the object.

System Action: ADSM ended the current operation.

User Response: Report to your system administrator that a storage pool on the server is full.

0033 E DSM_RC_ABORT_INVALID_OFFSET

Explanation: The partialObjOffset value for partial object retrieve is invalid.

System Action: The system returns to the calling procedure.

User Response: Specify a valid value.

0034 E DSM_RC_ABORT_INVALID_LENGTH

Explanation: partialObjLength value for partial object retrieve is invalid.

System Action: The system returns to the calling procedure.

User Response: Specify a valid value.

0051 E DSM_RC_REJECT_NO_RESOURCES

Explanation: ADSM has all available sessions in use and cannot accept a new one at this time.

System Action: ADSM canceled the current operation.

User Response: Retry the operation. If the problem continues, see your system administrator to increase the number of concurrently active sessions to the server.

0052 E DSM_RC_REJECT_VERIFIER_EXPIRED

Explanation: Your ADSM password has expired.

System Action: ADSM canceled the current operation. You are not allowed to connect to the server until the password is updated.

User Response: Update your password.

0053 E DSM_RC_REJECT_ID_UNKNOWN

Explanation: The node name you entered is not known by the server, or you are attempting to access a file migrated to a different node.

System Action: ADSM canceled the current operation. You are not allowed to connect to the server until your node name is registered with the server. If attempting to access a migrated file, your nodename must be the same node which migrated the file.

User Response: Ensure that you entered your ADSM node name correctly. If yes, see your system administrator. Verify that the server is using closed registration and that your node name is registered with the server.

0054 E DSM_RC_REJECT_DUPLICATE_ID

Explanation: Another process using this node name is active with the server.

System Action: ADSM cannot connect to the server. ADSM canceled the current operation.

User Response: If you are running a UNIX-based system, ensure that another process is not active with ADSM under the same name. Also, ensure that your node name is unique to the server so that it cannot be used by another person. See your system administrator to identify the owner of that node name.

0055 E DSM_RC_REJECT_SERVER_DISABLED

Explanation: The server is in a disabled state and cannot be accessed for normal activity.

System Action: ADSM canceled the current operation.

User Response: Retry the operation after the server returns to an enabled state. If the problem continues, see your system administrator.

0056 E DSM_RC_REJECT_CLOSED_REGISTER

Explanation: No authorization. Registration is required by your system administrator. The server is not configured to allow open registration.

System Action: Session not started.

User Response: You must obtain an ADSM node and password from your system administrator.

0057 S DSM_RC_REJECT_CLIENT_DOWNLEVEL

Explanation: The server version and your client version do not match. The client code is downlevel.

System Action: ADSM canceled the current operation.

User Response: See your system administrator to see what version of ADSM to run for your location.

0058 S DSM_RC_REJECT_SERVER_DOWNLEVEL

Explanation: The server version and your client version do not match. The server code is downlevel.

System Action: ADSM canceled the current operation.

User Response: See your system administrator to see what version of ADSM to run for your location.

0059 E DSM_RC_REJECT_ID_IN_USE

Explanation: The node name you specified is in use on the server.

System Action: Session was not started.

User Response: The server is probably performing a task that prevents your node from establishing a session. Retry later or check with your system administrator.

0061 E DSM_RC_REJECT_ID_LOCKED

Explanation: The node name you specified is currently locked on the server.

System Action: Session was not started.

User Response: Check with your system administrator to find out why your node name is locked.

0062 S DSM_RC_SIGNONREJECT_LICENSE_MAX

Explanation: Adding a new enrollment will exceed the product license count for ADSM.

System Action: Execution of the client enrollment or connection request ends.

User Response: See your system administrator.

0063 E DSM_RC_REJECT_NO_MEMORY

Explanation: The server does not have enough memory to allow your client to establish a connection with the server.

System Action: Session was not started.

User Response: Retry later or see your system administrator.

0064 E DSM_RC_REJECT_NO_DB_SPACE

Explanation: The server ran out of database space.

System Action: Session was not started.

User Response: See your system administrator.

0065 E DSM_RC_REJECT_NO_LOG_SPACE

Explanation: The server ran out of recovery log space.

System Action: Session was not started.

User Response: This error is a temporary problem. Retry later or see your system administrator.

0066 E DSM_RC_REJECT_INTERNAL_ERROR

Explanation: The client cannot establish a connection to the server because of an internal server error.

System Action: Session was not started.

User Response: See your system administrator immediately.

0067 S DSM_RC_SIGNONREJECT_INVALID_CLI

Explanation: The server is not licensed for the requeting client type.

System Action: Execution of the client enrollment or connection request ends.

User Response: See your system administrator.

0101 I DSM_RC_USER_ABORT

Explanation: An abort signal to stop an operation was received.

System Action: Processing stopped.

User Response: Continue with normal operations.

0102 E DSM_RC_NO_MEMORY

Explanation: The program has exhausted all available storage.

System Action: Processing stopped.

User Response: Free any unnecessary programs, for example, terminate and stay resident programs (TSRs), that are running and retry the operation. Reducing the scope of queries and the amount of data returned can also solve the problem.

0103 E DSM_RC_TA_COMM_DOWN

Explanation: An unexpected communications failure occurred during an ADSM operation.

System Action: Processing stopped.

User Response: Verify that communications are active between the client and server machines. Server outages, processor outages, and communication controller outages can cause this error.

0104 E DSM_RC_FILE_NOT_FOUND

Explanation: The file being processed for backup, archive or migrate no longer exists on the client. Another process deletes the file before it can be backed up, archived or migrated by ADSM.

System Action: File skipped.

User Response: None.

0105 E DSM_RC_PATH_NOT_FOUND

Explanation: You specified an incorrect directory path.

System Action: Processing stopped.

User Response: Correct the syntax specified on the call and retry the operation.

0106 E DSM_RC_ACCESS_DENIED

Explanation: Access to the specified file or directory is denied. You tried to read from or write to a file and you do not have access permission for either the file or the directory.

System Action: Processing stopped.

User Response: Ensure that you specified the correct file or directory name, correct the permissions, or specify a new location.

0107 E DSM_RC_NO_HANDLES

Explanation: All file handles for your system are currently in use. No more are available.

System Action: Processing stopped.

User Response: Either free some file handles by ending other processes, or modify your system setup to allow for more files to be open at the same time.

0108 E DSM_RC_FILE_EXISTS

Explanation: The file being restored or retrieved exists.

System Action: File is replaced or skipped depending on client options.

User Response: None.

0109 E DSM_RC_INVALID_PARM

Explanation: The system encountered an internal program error due to an invalid parameter.

System Action: The system returns to the calling procedure.

User Response: Ask your service representative to check the error log.

0110 E DSM_RC_INVALID_HANDLE

Explanation: An internal system error occurred. A file operation failed because an invalid file handle was passed.

System Action: Processing stopped.

User Response: Report the problem to your system administrator, and then retry the operation.

0111 E DSM_RC_DISK_FULL

Explanation: No more files can be restored or retrieved because the destination disk is full.

System Action: Processing stopped.

User Response: Free up disk space, or restore or retrieve the file to another disk.

0113 E DSM_RC_PROTOCOL_VIOLATION

Explanation: A communications protocol error occurred. The communication subsystem is not properly defined or is itself in error.

System Action: ADSM ended the current operation.

User Response: Verify that the communication processes are operating properly, and then retry the operation.

0114 E DSM_RC_UNKNOWN_ERROR

Explanation: An unknown error occurred. This might be a low-level system or communication error that ADSM cannot handle or recover from.

System Action: Processing stopped.

User Response: Retry the operation. If the problem continues, determine where the problem exists. See your system administrator for further help.

0115 E DSM_RC_UNEXPECTED_ERROR

Explanation: An unexpected error occurred. This might be a low-level system or communication error that ADSM cannot handle or recover from.

System Action: Processing stopped.

User Response: Retry the operation. If the problem continues, determine where the problem exists. See your system administrator for further help.

0116 E DSM_RC_FILE_BEING_EXECUTED

Explanation: The current file cannot be opened to write to because it is currently being run by another operation.

System Action: File skipped.

User Response: Stop the operation that is running the file and retry the operation, or restore or retrieve the file to a different name or directory.

0117 E DSM_RC_DIR_NO_SPACE

Explanation: No more files can be restored or retrieved since the destination directory is full.

System Action: Processing stopped.

User Response: Free up disk space, or restore or retrieve the file to another disk.

0118 E DSM_RC_LOOPED_SYM_LINK

Explanation: While trying to resolve the file name, too many symbolic links were found.

System Action: File skipped.

User Response: Ensure that you do not have a looping symbolic link for the file.

0119 E DSM_RC_FILE_NAME_TOO_LONG

Explanation: The file name specified is too long to be handled by ADSM.

System Action: File is skipped.

User Response: See the appropriate *Using the Backup-Archive Client* book for the particular operating system, for the file names that are handled by ADSM.

0120 E DSM_RC_FILE_SPACE_LOCKED

Explanation: File system cannot be accessed because it is locked by the system.

System Action: ADSM cannot complete the operation.

User Response: See your system administrator.

0121 I DSM_RC_FINISHED

Explanation: The operation is finished.

System Action: The system returns to the calling procedure.

User Response: Proceed with next function call.

0122 E DSM_RC_UNKNOWN_FORMAT

Explanation: ADSM tried to restore or retrieve a file, but it had an unknown format.

System Action: File skipped.

User Response: See your system administrator.

0123 E DSM_RC_NO_AUTHORIZATION

Explanation: The client is not authorized to restore the other node's data.

System Action: The system returns to the calling procedure.

User Response: Get authorization from the other node.

0124 E DSM_RC_FILE_SPACE_NOT_FOUND

Explanation: The specified file space (domain) is incorrect or does not exist on the workstation.

System Action: Processing stopped.

User Response: Retry the operation specifying an existing domain (drive letter or file system name).

0125 E DSM_RC_TXN_ABORTED

Explanation: The current transaction between the server and the client stopped. A server, client, or communication failure cannot be recovered.

System Action: ADSM canceled the current operation.

User Response: Retry the operation. If the problem continues, see your system administrator to isolate the problem.

0126 E DSM_RC_SUBDIR_AS_FILE

Explanation: ADSM tried to create a directory path, but is unable to because a file exists that has the same name as a directory.

System Action: Processing stopped.

User Response: Remove the file that has the same name as the directory. Refer to the last restore/retrieve operation and check all directories along the path.

0127 E DSM_RC_PROCESS_NO_SPACE

Explanation: The disk space allocated for the client owner is full.

System Action: Processing stopped.

User Response: Free up disk space and retry the restore or retrieve operation.

0128 E DSM_RC_PATH_TOO_LONG

Explanation: The path name specified plus the path name in the restored file name combine to create a name whose length exceeds the system maximum.

System Action: Processing stopped.

User Response: Specify a destination path that, when combined, is less than the system maximum.

0129 E DSM_RC_NOT_COMPRESSED

Explanation: A file that was flagged as compressed was not compressed, and the system failed.

System Action: Processing stopped.

User Response: See your system administrator to report this problem. This error is a system failure.

0130 E DSM_RC_TOO_MANY_BITS

Explanation: You are trying to restore a file that was backed up and compressed on another client workstation that had more memory than your client workstation. You cannot restore this file. When the file is restored, it is expanded and your workstation does not have enough memory.

System Action: ADSM canceled the operation.

User Response: Obtain a machine with more memory and retry the operation.

0131 S DSM_RC_SYSTEM_ERROR

Explanation: An unexpected program failure occurred.

System Action: Processing stopped.

User Response: Retry the operation. If the problem continues, see your system administrator or your service representative.

0132 E DSM_RC_NO_SERVER_RESOURCES

Explanation: The server ran out of resources. A lack of storage or a condition does not allow any new activity.

System Action: ADSM canceled the current operation.

User Response: Retry the operation at a later time. If the problem continues, see your system administrator to isolate what resource is unavailable.

0133 E DSM_RC_FS_NOT_KNOWN

Explanation: The number defining the correspondence between drive letter or file (domain name) and volume label is not known to the server.

System Action: Processing stopped.

User Response: Report the program error to your service representative.

0134 E DSM_RC_NO_LEADING_DIRSEP

Explanation: The objName field does not have a leading directory separator.

System Action: The system returns to the calling procedure.

User Response: Correct the value for the objName.

0135 E DSM_RC_WILDCARD_DIR

Explanation: Wildcards are not allowed in the objName directory path.

System Action: The system returns to the calling procedure.

User Response: Correct the value for the objName.

0136 E DSM_RC_COMM_PROTOCOL_ERROR

Explanation: Communications protocol error. An unexpected communications message was received by the client.

System Action: ADSM canceled the current operation.

User Response: Verify that your communication path is functioning properly. If the problem continues, have your service representative check for a possible program error.

0137 E DSM_RC_AUTH_FAILURE

Explanation: Authentication failure. You entered an incorrect password.

System Action: ADSM canceled the current operation.

User Response: Enter your correct password. If you cannot remember the correct password, see your system administrator to have a new one assigned for your node name.

0138 E DSM_RC_TA_NOT_VALID

Explanation: The trusted agent execution/owner permissions are invalid.

System Action: Processing stopped.

User Response: Have your system administrator check the installation instructions for the client to ensure that the trusted agent permissions are set correctly.

0139 S DSM_RC_KILLED

Explanation: Processing stopped. This is a programming failure and the client program ends.

System Action: Processing stopped.

User Response: Retry the operation. If the problem continues, contact your system administrator.

0145 S DSM_RC_WOULD_BLOCK

Explanation: The trusted agent blocks the operation. This is a programming failure and the client program ends.

System Action: Processing stopped.

User Response: Retry the operation. If the problem continues, contact your system administrator.

0146 S DSM_RC_TOO_SMALL

Explanation: The area for the include/exclude pattern is too small. This is a programming failure and the client program ends.

System Action: Processing stopped.

User Response: Retry the operation. If the problem continues, contact your system administrator.

0147 S DSM_RC_UNCLOSED

Explanation: There is no closing bracket in the pattern. This is a programming failure and the client program ends.

System Action: Processing stopped.

User Response: Retry the operation. If the problem continues, contact your system administrator.

0148 S DSM_RC_NO_STARTING_DELIMITER

Explanation: The include or exclude pattern must start with a directory delimiter.

System Action: Processing stopped.

User Response: Correct the syntax for the pattern.

0149 S DSM_RC_NEEDED_DIR_DELIMITER

Explanation: The include/exclude pattern has a '...' without a beginning or ending directory delimiter.

System Action: Processing stopped.

User Response: Correct the syntax for the pattern.

0150 S DSM_RC_UNKNOWN_FILE_DATA_TYPE

Explanation: An unknown and unexpected error code occurred within the client program. The structured file data type is unknown. This is a programming failure and the client program ends.

System Action: Processing stopped.

User Response: Retry the operation. If the problem continues, contact your system administrator.

0151 S DSM_RC_BUFFER_OVERFLOW

Explanation: The data buffer overflowed. This is a programming failure and the client program ends.

System Action: Processing stopped.

User Response: Retry the operation. If the problem continues, contact your system administrator.

0154 E DSM_RC_NO_COMPRESS_MEMORY

Explanation: Not enough memory is available to do data compression or expansion. For a restore or retrieve, the file cannot be recalled from the server until more storage is made available. For a backup or archive, try running without compression if storage cannot be made available.

System Action: Processing stopped.

User Response: Free up extra storage for the operation to continue, or run the backup or archive process without compression enabled.

0155 E DSM_RC_COMPRESS_GREW

Explanation: During compression the compressed data grew in size compared to the original data.

System Action: Processing stopped.

User Response: Send the file uncompressed.

0156 E DSM_RC_INV_COMM_METHOD

Explanation: You specified a communication method that is not supported.

System Action: Processing stopped.

User Response: Specify a valid communication interface for the ADSM client and your operating system.

0157 S DSM_RC_WILL_ABORT

Explanation: The server encountered an error and will abort the transaction.

System Action: The transaction will be aborted. The reason code is passed on the dsmEndTxn call.

User Response: Issue the dsmEndTxn with a vote of DSM_VOTE_COMMIT and examine the reason code.

0158 E DSM_RC_FS_WRITE_LOCKED

Explanation: The file or directory being restored or retrieved from the server cannot be written to because the destination is write locked. Another operation might have the file open and will not allow it to be updated.

System Action: File skipped.

User Response: Either determine which operation has the file write locked, or restore the file to another name or location.

0159 I DSM_RC_SKIPPED_BY_USER

Explanation: A file was skipped during a restore operation because the file is off line and the application has chosen not to wait for a tape mount.

System Action: File skipped.

User Response: Verify the application sets the mountWait value correctly on dsmBeginGetData.

0160 E DSM_RC_TA_NOT_FOUND

Explanation: ADSM was unable to find the ADSM Trusted Agent module in the specified directory. For V1R1 the name is dsmapita, for V1R2 the name is dsmtca, for V2R1 the name is dsmapitca.

System Action: ADSM ends.

User Response: Make sure the Trusted Agent module is in the directory specified by DSMI_DIR.

0161 E DSM_RC_TA_ACCESS_DENIED

Explanation: An attempt to access a system function has been denied.

System Action: Processing stopped.

User Response: Contact your system administrator.

0162 E DSM_RC_FS_NOT_READY

Explanation: The file system/drive was not ready for access.

System Action: Processing stopped.

User Response: Ensure that the drive is available to ADSM, and then retry the operation.

0163 E DSM_RC_FS_IS_BAD

Explanation: The drive was not available for access. A directory exists that does not have either a '.' or '..' entry.

System Action: Processing stopped.

User Response: Ensure that the drive is operational, and then retry the operation. If unsuccessful, have your service representative check the error log.

0164 E DSM_RC_FIO_ERROR

Explanation: An error was found while reading from or writing to the file.

System Action: File skipped.

User Response: Check your system to ensure that it is operating properly. For OS/2, run CHKDSK /F for the failing drive which can be found in dsmerror.log.

0165 E DSM_RC_WRITE_FAILURE

Explanation: An error was found while writing to the file.

System Action: File skipped.

User Response: Check your system to ensure that it is operating properly.

0166 E DSM_RC_OVER_FILE_SIZE_LIMIT

Explanation: A file being restored or retrieved exceeds system set limits for this user.

System Action: File skipped.

User Response: Ensure that the system limits are set properly.

0167 E DSM_RC_CANNOT_MAKE

Explanation: The directory path for files being restored or retrieved cannot be created.

System Action: File skipped.

User Response: Ensure that you have the proper authorization to create the directory for file being restored or retrieved.

0168 E DSM_RC_NO_PASS_FILE

Explanation: The file containing the stored password for the specified *server-name* is unavailable.

System Action: ADSM ends.

User Response: The root user must set and store a new password.

0169 E DSM_RC_VERFILE_OLD

Explanation: Either the password is not stored locally, or it was changed at the server.

System Action: ADSM prompts you for the password if ADSM is running in the foreground.

User Response: If ADSM was running as a background process, issue any ADSM command from the foreground. Enter the password in answer to the prompt. Then try your background ADSM command again.

0173 E DSM_RC_INPUT_ERROR

Explanation: Unable to read commands entered from keyboard. ADSM cannot process your intended command.

System Action: Processing stopped.

User Response: Ensure that you are entering a correct command.

0174 E DSM_RC_REJECT_PLATFORM_MISMATCH

Explanation: Your node name is associated with a different type of operating system (such as DOS, OS/2, or AIX) and cannot be used on this system.

System Action: ADSM canceled the current operation.

User Response: If you need a new node name, see your system administrator to assign a new one to you. Generally, you have a unique node name for each machine and operating system pair that requires access to the server.

0175 E DSM_RC_TL_NOT_FILE_OWNER

Explanation: The file cannot be backed up because the client is not the file owner.

System Action: ADSM skips the file.

User Response: None.

0176 S DSM_RC_DBCS_IN_RANGE

Explanation: Only single-byte characters are allowed in an include/exclude list; you cannot use a double-byte character set (DBCS).

System Action: Processing stopped.

User Response: Remove the double-byte characters from the include/exclude list and retry the operation.

0177 S DSM_RC_UNMATCHED_QUOTE

Explanation: The quotes specified in the pattern are not the same and do not make a set.

System Action: Processing stopped.

User Response: Correct the pattern by using matching quotes in the syntax.

0181 E DSM_RC_PS_MULTBCG

Explanation: Multiple backup copy groups were found in a management class. Only one backup copy group is allowed per management class.

System Action: Processing stopped.

User Response: See your system administrator.

0182 E DSM_RC_PS_MULTACG

Explanation: Multiple archive copy groups were found in a management class. Only one archive copy group is allowed per management class.

System Action: Processing stopped.

User Response: See your system administrator.

0183 E DSM_RC_PS_NODFLTMC

Explanation: The default management class is missing in the policy set.

System Action: Processing stopped.

User Response: Have your system administrator set a default management class for the policy set.

0184 E DSM_RC_TL_NOBCG

Explanation: The management class for this file does not have a backup copy group specified. This file will not be backed up.

System Action: Processing stopped.

User Response: Add a valid backup copy group to the management class, and then retry the operation.

0185 W DSM_RC_TL_EXCLUDED

Explanation: You tried to back up or migrate a file (*file-name*) that was specified to be excluded from backup.

System Action: ADSM did not back up or migrate the file.

User Response: Specify the file using the Include option and retry the operation.

0186 E DSM_RC_TL_NOACG

Explanation: The management class for this file does not have an archive copy group specified. This file will not be archived.

System Action: Processing stopped.

User Response: Add a valid archive copy group to the management class, and then retry the operation.

0187 E DSM_RC_PS_INVALID_ARCHMC

Explanation: You entered an invalid management class.

System Action: ADSM is unable to do the requested operation.

User Response: Retry the operation using a valid management class.

0188 S DSM_RC_NO_PS_DATA

Explanation: Either no Active Policy Set data was found on the server or a fromnode option contained a nodename not found on the server.

System Action: Processing stopped.

User Response: See your system administrator.

0189 S DSM_RC_PS_INVALID_DIRMC

Explanation: An invalid management class was assigned to directories.

System Action: Processing stopped.

User Response: Have your service representative check the error log.

0190 S DSM_RC_PS_NO_CG_IN_DIR_MC

Explanation: The management class used for directories does not have a backup copy group.

System Action: Processing stopped.

User Response: Have your service representative check the error log.

0200 E DSM_RC_TCA_ATTACH_SHR_MEM_ERR

Explanation: An error has occured while attaching the trusted agent's shared memory.

System Action: Return to caller.

User Response: Stop application, check shared memory usage and retry the command. Read the tca.log file for the system error number.

0201 E DSM_RC_TCA_SHR_MEM_BLOCK_ERR

Explanation: The shared memory used by the trusted agent has an invalid block ID.

System Action: Return to caller.

User Response: Stop THE application and retry the command.

0202 E DSM_RC_TCA_SHR_MEM_IN_USE

Explanation: The shared memory attached by the trusted agent is being used.

System Action: Return to caller.

User Response: Stop the application and retry the command.

0291 E DSM_RC_TCA_SHARED_MEMORY_ERROR

Explanation: An error has occurred while obtaining or accessing the shared memory block used by the ADSM client and the Trusted Communication Agent.

System Action: ADSM ends.

User Response: Probable system error. If the problem persists, restart the workstation.

0292 E DSM_RC_TCA_FORK_FAILED

Explanation: An error has occurred starting the Trusted Communication Agent process; specifically, the fork() function has failed.

System Action: ADSM ends.

User Response: Probable system error. If the problem persists, restart the workstation.

0293 E DSM_RC_TCA_SEGMENT_MISMATCH

Explanation: The shared memory block used by the ADSM client and the Trusted Communication Agent is not mapped to the same segment address in both programs.

System Action: ADSM ends.

User Response: Probable ADSM implementation error. Contact IBM customer support.

0294 E DSM_RC_TCA_DIED

Explanation: The Trusted Communication Agent has terminated unexpectedly.

System Action: ADSM ends.

User Response: Check the error log for more information. Retry the activity. If the problem persists, contact IBM customer support.

0295 E DSM_RC_TCA_INVALID_REQUEST

Explanation: The Trusted Communication Agent has received an unknown request from the ADSM client.

System Action: ADSM ends.

User Response: Internal error. If the problem recurs, contact IBM customer support.

0296 E DSM_RC_TCA_NOT_ROOT

Explanation: An activity has been attempted that must be performed by the root user (for example, open registration or password update).

System Action: ADSM ends.

User Response: If the activity is required, the root user must perform it.

0297 E DSM_RC_TCA_SEMGET_ERROR

Explanation: An error has occurred while attempting to allocate semaphores.

System Action: Processing ends.

User Response: Probable system error. If the problem persists, restart the workstation.

0298 E DSM_RC_TCA_SEM_OP_ERROR

Explanation: An error has occurred while attempting to set or wait on a semaphore.

System Action: Processing ends.

User Response: Probable system error. If the problem persists, restart the workstation.

0600 E DSM_RC_DUP_LABEL

Explanation: The selected drive has a duplicate volume label. Because ADSM uses the volume label to keep track of backup/archive information, it cannot back up or archive files from a drive with a duplicate volume label.

System Action: ADSM cannot select the drive.

User Response: If the volume needs to be available to the system, exit ADSM, and assign a volume label to the drive. Restart ADSM and retry the operation.

0601 E DSM_RC_NO_LABEL

Explanation: The selected drive does not have a label.

System Action: ADSM is unable to do the requested operation without a drive or label entered.

User Response: If the drive is a floppy drive, place a disk with a volume label in it and retry the operation. If the disk is a hard drive, ensure the drive has a volume label, and retry the operation.

0610 E DSM_RC_NLS_CANT_OPEN_TXT

Explanation: The system is unable to open the message txt file (dscameng.txt or dsmclient.cat for AIX). On the AS/400 platform this file is QANSAPI/QAANSAMENG(TXT).

System Action: The system returns to the calling procedure.

User Response: Verify that the dscameng.txt file is in the directory pointed to by DSMI_DIR. For AIX, verify that the dsmclient.cat file has a symbolic link to /usr/lib/nls/msg/<locale>/dsmclient.cat .

0611 E DSM_RC_NLS_CANT_READ_HDR

Explanation: The system is unable to use the message text file (dscameng.txt or dsmclient.cat for AIX) because of an invalid header. On the AS/400 platform this file is QANSAPI/QAANSAMENG(TXT).

System Action: The system returns to the calling procedure.

User Response: Install the message text file again.

0612 E DSM_RC_NLS_INVALID_CNTL_REC

Explanation: The system is unable to use the message txt file (dscameng.txt or dsmclient.cat for AIX) because of an invalid control record. On the AS/400 platform this file is QANSAPI/QAANSAMENG(TXT).

System Action: The system returns to the calling procedure.

User Response: Install the message text file again.

0613 E DSM_RC_NLS_INVALID_DATE_FMT

Explanation: An invalid value is specified for DATEFORMAT.

System Action: The system returns to the calling procedure.

User Response: Specify a valid value.

0614 E DSM_RC_NLS_INVALID_TIME_FMT

Explanation: An invalid value is specified for TIMEFORMAT.

System Action: The system returns to the calling procedure.

User Response: Specify a valid value.

0615 E DSM_RC_NLS_INVALID_NUM_FMT

Explanation: An invalid value is specified for NUMBERFORMAT.

System Action: The system returns to the calling procedure.

User Response: Specify a valid value.

0620 E DSM_RC_LOG_CANT_BE_OPENED

Explanation: The system is unable to open the error log file.

System Action: The system returns to the calling procedure.

User Response: Verify the DSMI_LOG value and access permission. On the AS/400 platform, verify the value specified for ERRORLOGNAME in the API options file.

0621 E DSM_RC_LOG_ERROR_WRITING_TO_LOG

Explanation: There was an error writing to the log file.

System Action: The system returns to the calling procedure.

User Response: Verify the DSMI_LOG value and access permission. On the AS/400 platform, verify the value specified for ERRORLOGNAME in the API options file.

0622 E DSM_RC_LOG_NOT_SPECIFIED

Explanation: The system is unable to open the error log file.

System Action: The system returns to the calling procedure.

User Response: Verify the DSMI_LOG value and access permission. On the AS/400 platform, verify the value specified for ERRORLOGNAME in the API options file.

2000 E DSM_RC_NULL_OBJNAME

Explanation: There is no value provided for the object name pointer.

System Action: The system returns to the calling procedure.

User Response: Provide an address for the dsmObjName structure.

2001 E DSM_RC_NULL_DATABLKPTR

Explanation: There is no value provided for the data block pointer.

System Action: The system returns to the calling procedure.

User Response: Provide an address for the DataBlk structure.

2002 E DSM_RC_NULL_MSG

Explanation: The message parameter for dsmRCMsg is a NULL pointer.

System Action: The system returns to the calling procedure.

User Response: Allocate enough space for the message parameter.

2004 E DSM_RC_NULL_OBJATTRPTR

Explanation: There is no value provided for the object attribute pointer.

System Action: The system returns to the calling procedure.

User Response: Provide an address for the ObjAttr structure.

2006 E DSM_RC_NO_SESS_BLK

Explanation: The server did not respond with the session information.

System Action: The system returns to the calling procedure.

User Response: Verify the server status.

2007 E DSM_RC_NO_POLICY_BLK

Explanation: The server did not respond with the policy information.

System Action: The system returns to the calling procedure.

User Response: Verify the server policy definitions.

2008 E DSM_RC_ZERO_BUFLEN

Explanation: The value for the dataBlk bufferLen is zero.

System Action: The system returns to the calling procedure.

User Response: Provide a non-zero value for the bufferLen.

2009 E DSM_RC_NULL_BUFPTR

Explanation: There is no value provided for the dataBlk bufferPtr.

System Action: The system returns to the calling procedure.

User Response: Provide an address for the bufferPtr.

2010 E DSM_RC_INVALID_OBJTYPE

Explanation: The value for the objType is invalid.

System Action: The system returns to the calling procedure.

User Response: The value for dsmObjName.objType must be:

DSM_OBJ_FILE or DSM_OBJ_DIRECTORY for Backup, or

DSM_OBJ_FILE for Archive.

2011 E DSM_RC_INVALID_VOTE

Explanation: The dsmEndTxn vote is invalid.

System Action: The system returns to the calling procedure.

User Response: The vote must be DSM_VOTE_COMMIT or DSM_VOTE_ABORT.

2013 E DSM_RC_INVALID_OPT

Explanation: An invalid option was found.

System Action: The system returns to the calling procedure.

User Response: Verify the options in dsm.opt, dsm.sys, and the options string. On the AS/400 platform, verify the options in *LIBL/QOPTADSM(APIOPT). Check the error log for more details about the error.

2014 E DSM_RC_INVALID_DS_HANDLE

Explanation: The system encountered an error in the API internals.

System Action: The system returns to the calling procedure.

User Response: Shut down the process and retry the operation. Verify that any previous dsmInit calls were cleaned up and terminated by a dsmTerminate call. If the problem continues, contact your system administrator or service representative.

2015 E DSM_RC_INVALID_REPOS

Explanation: The repository type is invalid.

System Action: The system returns to the calling procedure.

User Response: For dsmDeleteFS the repository must be one of the following:

- DSM_ARCHIVE_REP
- DSM_BACKUP_REP
- DSM_REPOS_ALL.

2016 E DSM_RC_INVALID_FSNAME

Explanation: The filespace name is invalid.

System Action: The system returns to the calling procedure.

User Response: Filespace name should start with the directory delimiter.

2017 E DSM_RC_INVALID_OBJNAME

Explanation: The object name is invalid because of an empty string or there is no leading delimiter.

System Action: The system returns to the calling procedure.

User Response: Verify the format of the dsmObjName full path.

2018 E DSM_RC_INVALID_LLNAME

Explanation: The low level qualifier for the object name is invalid.

System Action: The system returns to the calling procedure.

User Response: Start the low level qualifier of the object name with the directory delimiter.

2019 E DSM_RC_INVALID_OBJOWNER

Explanation: The object owner must be either the root user, or the object owner must be the same as the session owner.

System Action: The system returns to the calling procedure.

User Response: Verify the session owner and object owner.

2020 E DSM_RC_INVALID_ACTYPE

Explanation: The dsmBindMC sendType is invalid.

System Action: The system returns to the calling procedure.

User Response: The sendType must be one of the following:

stBackup

stArchive

stBackupMountWait

stArchiveMountWait

2021 E DSM_RC_INVALID_RETCODE

Explanation: The dsmRC parameter for dsmRCMsg is an unsupported return code.

System Action: The system returns to the calling procedure.

User Response: Specify a valid value.

2022 E DSM_RC_INVALID_SENDTYPE

Explanation: The dsmSendObj sendType is invalid.

System Action: The system returns to the calling procedure.

User Response: The sendType must be one of the following:

stBackup

stArchive

stBackupMountWait

stArchiveMountWait

2023 E DSM_RC_INVALID_PARAMETER

Explanation: The dsmDeleteObj delType is invalid.

System Action: The system returns to the calling procedure.

User Response: The delType must be dtBackup or dtArchive.

2024 E DSM_RC_INVALID_OBJSTATE

Explanation: The query Backup objState is invalid.

System Action: The system returns to the calling procedure.

User Response: The qryBackupData.objState must be one of the following:

DSM_ACTIVE

DSM_INACTIVE

DSM_ANY_MATCH

2025 E DSM_RC_INVALID_MCNAME

Explanation: A query or send operation is unable to find the management class name.

System Action: The system returns to the calling procedure.

User Response: Verify the management class name.

2026 E DSM_RC_INVALID_DRIVE_CHAR

Explanation: The drive letter is not an alphabetic character. This return code is valid on Microsoft Windows or OS/2.

System Action: The system returns to the calling procedure.

User Response: Verify that the drive designation is an alphabetic character. The referenced field is dsmDosFSAttrib.driveLetter.

2027 E DSM_RC_NULL_FSNAME

Explanation: There is no value provided for the Register Filespace name.

System Action: The system returns to the calling procedure.

User Response: Provide a filespace name on dsmRegisterFS.

2028 E DSM_RC_INVALID_HLNAME

Explanation: The high level qualifier for the object name is invalid.

System Action: The system returns to the calling procedure.

User Response: High level qualifier of the object name should start with the directory delimiter.

2029 E DSM_RC_NUMOBJ_EXCEED

Explanation: The number of objects (numObjId) specified on the dsmBeginGetData call exceeds DSM_MAX_GET_OBJ.

System Action: The system returns to the calling procedure.

User Response: Check the number of objects before calling dsmBeginGetData. If it is greater than DSM_MAX_GET_OBJ, issue multiple Get call sequences.

2030 E DSM_RC_NEWPW_REQD

Explanation: There is no value provided for new password.

System Action: The system returns to the calling procedure.

User Response: Provide a new password on dsmChangePW.

2031 E DSM_RC_OLDPW_REQD

Explanation: There is no value provided for old password.

System Action: The system returns to the calling procedure.

User Response: Provide an old password on dsmChangePW.

2032 E DSM_RC_NO_OWNER_REQD

Explanation: PASSWORDACCESS=generate establishes a session with the current login user as the owner.

System Action: The system returns to the calling procedure.

User Response: When using PASSWORDACCESS=generate, set clientOwnerNameP to NULL.

2033 E DSM_RC_NO_NODE_REQD

Explanation: PASSWORDACCESS=generate establishes a session with the current hostname as the node.

System Action: The system returns to the calling procedure.

User Response: When using PASSWORDACCESS=generate, set clientNodeP to NULL.

2041 E DSM_RC_BAD_CALL_SEQUENCE

Explanation: The sequence of calls is invalid.

System Action: The system returns to the calling procedure.

User Response: Verify the transaction call sequence.

2050 E DSM_RC_WILDCHAR_NOTALLOWED

Explanation: On dsmSendObj, wildcards are not allowed for the objName.

System Action: The system returns to the calling procedure.

User Response: Provide a fs, hl, and ll on the dsmObjName.

2060 E DSM_RC_FSNAME_NOTFOUND

Explanation: The filespace to delete cannot be found.

System Action: The system returns to the calling procedure.

User Response: Verify the filespace name.

2061 E DSM_RC_FS_NOT_REGISTERED

Explanation: On dsmSendObj, dsmDeleteObj, or dsmUpdateFS, the filespace is not registered.

System Action: The system returns to the calling procedure.

User Response: Verify the filespace name.

2062 W DSM_RC_FS_ALREADY_REGED

Explanation: On dsmRegisterFS the filespace is already registered.

System Action: The system returns to the calling procedure.

User Response: Verify the filespace name.

2063 E DSM_RC_OBJID_NOTFOUND

Explanation: On dsmBeginGetData, the objID is NULL.

System Action: The system returns to the calling procedure.

User Response: Verify the following:

The dsmGetList is not NULL.

Each objID is not NULL.

The dsmGetList numObjId is not zero.

2064 E DSM_RC_WRONG_VERSION

Explanation: On dsmInit the caller's API version has a higher value than the ADSM library version.

System Action: The system returns to the calling procedure.

User Response: Install the latest ADSM API library and trusted agent module.

2065 E DSM_RC_WRONG_VERSION_PARM

Explanation: The caller's structure version is different than the ADSM library version.

System Action: The system returns to the calling procedure.

User Response: Ensure that the stVersion field is set with the value in the header file. Recompile the application with the latest header files.

2070 E DSM_RC_NEEDTO_ENDTXN

Explanation: This transaction must be ended and a new one must be started due to one of the following reasons:

The destination changed.

The byte limit is exceeded

The maximum number of objects is exceeded.

System Action: The system returns to the calling procedure.

User Response: Issue dsmEndTxn and start a new transaction session.

2080 E DSM_RC_OBJ_EXCLUDED

Explanation: The backup or archive object is excluded from processing.

System Action: The system returns to the calling procedure.

User Response: Verify the objName and Exclude lists.

2081 E DSM_RC_OBJ_NOBCG

Explanation: The backup object does not have a copy group.

System Action: The system returns to the calling procedure.

User Response: Verify server policy definitions.

2082 E DSM_RC_OBJ_NOACG

Explanation: The archive object does not have a copy group.

System Action: The system returns to the calling procedure.

User Response: Verify server policy definitions.

2090 E DSM_RC_APISYSTEM_ERROR

Explanation: Memory used by the ADSM API has been corrupted.

System Action: The system returns to the calling procedure.

User Response: Retry the operation. If the problem continues, contact your system administrator or service representative.

2100 E DSM_RC_DESC_TOOLONG

Explanation: The sendObj Archive description is too long.

System Action: The system returns to the calling procedure.

User Response: The sndArchiveData.descr string must be less than or equal to DSM_MAX_DESCR_LENGTH.

2101 E DSM_RC_OBJINFO_TOOLONG

Explanation: The sendObj ObjAttr.objInfo is too long.

System Action: The system returns to the calling procedure.

User Response: The objInfo field must be less than or equal to DSM_MAX_OBJINFO_LENGTH.

2102 E DSM_RC_HL_TOOLONG

Explanation: The sendObj dsmObjName.hl is too long.

System Action: The system returns to the calling procedure.

User Response: The hl field must be less than or equal to DSM_MAX_HL_LENGTH.

2103 E DSM_RC_PASSWD_TOOLONG

Explanation: The dsmChangePW password is too long.

System Action: The system returns to the calling procedure.

User Response: The password field must be less than or equal to DSM_MAX_VERIFIER_LENGTH.

2104 E DSM_RC_FILESPACE_TOOLONG

Explanation: The sendObj dsmObjName.fs is too long.

System Action: The system returns to the calling procedure.

User Response: The fs field must be less than or equal to DSM_MAX_FS_LENGTH.

2105 E DSM_RC_LL_TOOLONG

Explanation: The sendObj dsmObjName.ll is too long.

System Action: The system returns to the calling procedure.

User Response: The ll field must be less than or equal to DSM_MAX_LL_LENGTH.

2106 E DSM_RC_FSINFO_TOOLONG

Explanation: On RegisterFS or UpdateFS the fsAttr's fsInfo is too long.

System Action: The system returns to the calling procedure.

User Response: The fsInfo field must be less than or equal to DSM_MAX_FSINFO_LENGTH.

2200 I DSM_RC_MORE_DATA

Explanation: On dsmGetNextQObj or dsmGetData there is no more available data.

System Action: The system returns to the calling procedure.

User Response: Start the function again.

2210 E DSM_RC_BUFF_TOO_SMALL

Explanation: The dataBlk buffer is too small for the query response.

System Action: The system returns to the calling procedure.

User Response: On dsmGetNextQObj ensure that the dataBlk buffer is at least as big as the query response structure.

2220 S DSM_RC_NO_OPT_FILE

Explanation: The options file specified by *file-name* cannot be found.

System Action: The ADSM client ends.

User Response: See if you have the environment variable DSM_CONFIG (or DSMI_CONFIG for the API) set, which explicitly identifies the ADSM options file. (You can do this by entering the SET command at your system.) If this environment variable is set, ensure the file indicated by the variable exists. If it is not set, then ADSM looks for the file dsm.opt in the current directory. If neither of these cases is met, you receive this error message.

2221 E DSM_RC_INVALID_KEYWORD

Explanation: An invalid option keyword was found in the dsmInit configuration file, the option string, dsm.sys, or dsm.opt.

System Action: The system returns to the calling procedure.

User Response: Correct the spelling of the option keywords. Verify that the dsmInit configuration file only has a subset of the dsm.sys options. Check the error log for more details about the error.

2222 S DSM_RC_PATTERN_TOO_COMPLEX

Explanation: The include or exclude pattern issued is too complex to be accurately interpreted by ADSM.

System Action: Processing stopped.

User Response: Recode the include or exclude pattern as shown in one of the examples in the appropriate *Using the Backup-Archive Client* book for the particular operating system.

2223 S DSM_RC_NO_CLOSING_BRACKET

Explanation: The include or exclude pattern is incorrectly constructed. The closing bracket is missing.

System Action: Processing stopped.

User Response: Correct the syntax for the pattern.

2225 E DSM_RC_INVALID_SERVER

Explanation: The system options file does not contain the SERVERNAME option.

System Action: ADSM initialization fails and the program ends.

User Response: See the root user, and make sure that the system options file contains the server name.

2226 E DSM_RC_NO_HOST_ADDR

Explanation: The TCPSERVERADDRESS for this server is not defined in the server name stanza in the system options file.

System Action: ADSM initialization fails and the program ends.

User Response: See the root user, and make sure that the server you are trying to connect to has a valid TCPSERVERADDRESS defined in the system options file.

2227 E DSM_RC_MACHINE_SAME

Explanation: The NODENAME defined in the options file cannot be the same as the system HostName.

System Action: Initialization fails and the program ends.

User Response: See your system administrator or the root user.

2228 E DSM_RC_NO_API_CONFIGFILE

Explanation: The configuration file specified on dsmInit cannot be opened.

System Action: The system returns to the calling procedure.

User Response: Verify the file name.

2229 E DSM_RC_NO_INCLEXCL_FILE

Explanation: The Include/Exclude definition file was not found.

System Action: The system returns to the calling procedure.

User Response: Verify the file name on the Inclexcl option.

2230 E DSM_RC_NO_SYS_OR_INCLEXCL

Explanation: Either the dsm.sys file was not found, or the Inclexcl file specified in dsm.sys was not found.

System Action: The system returns to the calling procedure.

User Response: The dsm.sys file must be in the directory referenced by the environment variable DSMI_DIR. Verify the file name on the Inclexcl option in the dsm.sys file.

2231 E DSM_RC_REJECT_NO_POR_SUPPORT

Explanation: The ADSM server specified by the user does not support partial object retrieve.

System Action: The system returns to the calling procedure.

User Response: Specify an ADSM server which supports the partial object retrieve function.

2300 E DSM_RC_NEED_ROOT

Explanation: Only a UNIX root user can execute dsmChangePW or dsmDeleteFS.

System Action: The system returns to the calling procedure.

User Response: Run this program as a root user.

2301 E DSM_RC_NEEDTO_CALL_BINDMC

Explanation: You must issue dsmBindMC before dsmSendObj.

System Action: The system returns to the calling procedure.

User Response: Modify your program.

2302 I DSM_RC_CHECK_REASON_CODE

Explanation: After a dsmEndTxn call, the transaction is aborted by either the server or client with a DSM_VOTE_ABORT and the reason is returned.

System Action: The system returns to the calling procedure.

User Response: Check the reason field for the code which explains why the transaction has been aborted.

Appendix E. The ADSM X/Open API

The X/Open Backup Services API (XBSA) is a set of function definitions, data structures, and return codes developed by the X/Open Company. Its purpose is to present a standardized interface between applications that need to perform backup or archive operations and the enterprise solutions that provide these services. ADSM is such a solution.

Introduction

ADSM's X/Open API enables an application client to use the ADSM storage management functions. The X/Open API consists of a set of function calls that an application client can use to perform the following operations:

- Initialize and terminate an ADSM session
- Assign management classes to objects before storing them on an ADSM server
- Back up or archive objects to an ADSM server
- Restore or retrieve objects from an ADSM server
- Query an ADSM server for information about objects stored on the server
- Delete backed up and archived objects from an ADSM server

When you, as an application developer, install the X/Open API, you get the following:

- The files that an end user of an application would need:

  The X/Open API shared library
  The Trusted Communication Agent program
  Sample client options files
  Documentation

- 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 X/Open API for ADSM is available on the following platforms:

- AIX 3.2.5 and 4.1
- Solaris 2.3 and 2.4

For information on installing the X/Open API, see the *Installing the Clients* manual.

Getting Information and Support

The IBM Storage Systems Division (SSD) Software Developer's Program provides a range of services to software developers who want to use the X/Open API. Information about the SSD Software Developer's Program is available in:

- IBMSTORAGE forum on CompuServe
- SSD Software Developer's Program Information Package

To obtain the Software Developer's Program Information Package:

1. Call 800-4-IBMSSD (800-442-6773). Outside the U.S.A., call 408-256-0000.

2. Listen for the SSD Software Developer's Program prompt.
3. Request the Software Developer's Program Information Package.

IBM has two programs—*standard* and *premier*—to provide you with technical support and notification of updates.

See the **register.frm** file for details and application forms. You receive this file when you install the X/Open API.

Setting Up Options Files

Options files allow you to set the conditions and boundaries under which your ADSM session runs. The available options can be set by the ADSM administrator, the end user, or you. The values of various options allow you to do the following:

- Set up the connection to an ADSM server
- Control which objects are sent to the server and what management class they are associated with
- Set the format in which various object attributes appear, such as the date and time

The same option can appear in more than one options file. When this happens, the file with the highest priority takes precedence.

The different options files, in order of decreasing priority, are as follows:

1. *Administrator options*. Options set by an ADSM administrator, whether on the client or the server, override any options set by you or the end user.

   For example, the administrator can specify whether or not objects can be compressed before being sent to an ADSM server. In this case, setting the COMPRESSION option in the client options file has no effect. The administrator can also decide that the choice of allowing compression is to be determined by the client. Setting the COMPRESSION option in the client options file then determines whether objects are compressed before being stored.
2. The ADSM options files on the UNIX platform include the *user options file* (**dsm.opt**) and the *system options file* (**dsm.sys**). These files are set up by the end user when the ADSM API is first installed on the user's workstation. The options in these files can be overridden by the methods mentioned above.

For more detailed information on the options available, see either the *Installing the Clients* book or the *Using the UNIX Backup-Archive Clients* book.

Using the ADSM 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. You should install the sample application and look at its source code to better understand how the function calls can be used.

Building the Sample Application

The files listed in Figure 45 make up the source files and other files needed to build the sample application included with the X/Open API.

Figure 45. Files Available for Building 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
xapibkup.c xapidata.h xapiinit.c xapint64.h xapint64.c xapipref.c xapiproc.c xapipw.c xapiqry.c xapirc.c xapismp.c xapitype.h xapiutil.h xapiutil.c	Modules for the command line driven sample application
makexapi.aix makexapi.sol	Makefile to build xapismp for AIX Makefile to build xapismp for Solaris

Follow these steps to compile the sample application and test the installation. Note that several of the steps have slight variations, depending on which UNIX platform you are using.

In the following instructions, the source directory from which the files are copied may not be the same as the directory you are using on your workstation. If that is the case, substitute the directory that you set in the DSMI_DIR environment variable for the one in these instructions.

The target directory in these instructions has the name **/u/developer/testapi**. However, you can use any name for the target directory.

1 Copy the API library to the **/usr/lib** directory:

AIX:	`cp /usr/lpp/adsm/api/bin/libXApi.a /usr/lib`
Solaris:	`cp /usr/adsm/api/libXApi.so /usr/lib/libXApi.so`

Instead of copying the API library, you can create a symbolic link to the file from the **/usr/lib** directory. First change to the **/usr/lib** directory with the command:

```
cd /usr/lib
```

Then enter the following command:

AIX:	`ln -s /usr/lpp/adsm/api/bin/libXApi.a`
Solaris:	`ln -s /usr/adsm/api/libXApi.so`

2 For the Solaris platform, rename the Trusted Communication Agent and the message text file:

```
cp dsmapitca.sol2 dsmapitca
cp dscameng.txt.sol2 dscameng.txt
```

Check the permission bits and the owner of **dsmapitca** with the following command:

```
ls -l dsmapitca
```

If the permission bits are not set to `-rwsr-xr-x`, enter the command:

```
chmod 4755 dsmapitca
```

If the owner is not set to `root`, enter the command:

```
chown root dsmapitca
```

3 Copy the sample application files to the target directory:

AIX:	`cp /usr/lpp/adsm/api/bin/xapi* /u/developer/testapi`
Solaris:	`cp /usr/adsm/api/xapi* /u/developer/testapi`

4 Copy the header files to the target directory:

AIX:	`cp /usr/lpp/adsm/api/bin/*.h /u/developer/testapi`
Solaris:	`cp /usr/adsm/api/*.h /u/developer/testapi`

5 Copy the makefile to the target directory:

AIX:	`cp /usr/lpp/adsm/api/bin/makexapi.aix /u/developer/testapi`
Solaris:	`cp /usr/adsm/api/makexapi.sol /u/developer/testapi`

6 Compile the sample with the following command:

AIX:	`make -f makexapi.aix`
Solaris:	`make -f makexapi.sol`

7 Ensure that your environment variables, especially DSMI_DIR, and options files are set up. See "Setting Up Options Files" on page 148 and the *Installing the Clients* book for information.

8 Log on as root the first time for password registration.

9 Run **xapismp** to start the sample application.

Running the Sample Application

After you start the sample application by entering **xapismp**, follow the instructions that appear on the screen. Some things to remember when running the application are:

- You must run the Signon action before any other action.
- When entering 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.

Using the ADSM X/Open API

This section describes, in a task-oriented fashion, how to use the X/Open Application Program Interface. You should be familiar with this section prior to designing or writing an application that uses the X/Open API.

ADSM's X/Open API supports the functions in XBSA's Data Movement function group. These functions include the following:

BSABeginTxn	BSAGetNextQueryObject[1]
BSAChangeToken	BSAGetObject
BSACreateObject	BSAInit
BSADeleteObject	BSAMarkObjectInactive
BSAEndData	BSAQueryApiVersion
BSAEndTxn	BSAQueryObject
BSAGetData	BSASendData
BSAGetEnvironment	BSATerminate

In addition, the X/Open API supports the following function:

BSAResolveLifecycleGroup

See the *X/Open Specification* for detailed information on each function.

Note: The following functions are part of XBSA's Data Movement function group, but are not currently implemented in the X/Open API. Calls to these functions return the code `BSA_RC_BAD_CALL_SEQUENCE`.

BSACreateObjectF
BSAGetObjectF
BSASetEnvironment

1 In the *X/Open Specification*, BSAGetNextQueryObject was accidentally omitted from the list of functions in XBSA's Data Movement function group.

| The API package that you receive includes a sample application. Look at the source code for the sample application to see examples of the X/Open API functions in context.

Maintaining Version Control in the API

All APIs have some form of version control, and X/Open is no exception. You must be sure the version of the X/Open API that you use in your application is compatible with the version of the API library that the end users have installed on their workstations.

The first API call issued when using the X/Open API should usually be **BSAQueryApiVersion**. This call does the following for the application client:

- Confirms that the X/Open API library is installed and available on the end user's system
- Returns the version level of the API library being accessed by the application

The X/Open API is designed to be upward compatible, so that applications written to older versions or releases of the API library will still operate correctly if the end user is running a newer version.

Determining the release of the API library is critical, because some releases might have different memory requirements and data structure definitions. Downward compatibility might be possible on a case-by-case basis, but it is not a design goal to be so. Downward compatibility, if a requirement, is the responsibility of the application client.

The API library and the Trusted Communication Agent module (**dsmapitca**) must be at the same level.

The **BSAQueryApiVersion** call returns the version of the API library installed on the end user's workstation. You can then compare the returned value with the version of the X/Open API that the application client was built with.

The version number of the application client's API is hard-coded 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 API library installed on the user's system. Any other condition should be entered into with care.

The **BSAQueryApiVersion** call can be issued at any time, whether the API session has been initialized or not.

Starting and Terminating a Session

ADSM is a session-oriented product, and all activities must be performed within an ADSM session. To start a session, the application invokes 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 ADSM server as indicated in the parameters passed in the call or defined in the options files.

Note that the application client can only register new nodes with an ADSM server if the server has closed registration. If a server has open registration, then any nodes that are already registered with the server will be accepted by the application. However, if a server has open registration and **BSAInit** tries to register a new node, then 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 an ADSM session. `BSAObjectOwner` is used as the ADSM node name. `AppObjectOwner` contains the ADSM session owner name. The node name and password are used for session authentication with the ADSM server. The session owner name is used to determine which objects can be accessed during the session.

ADSM has two modes for handling passwords - Prompt and Generate. The mode is set in the PASSWORDACCESS option in the client options file. For Prompt mode, the node/owner/password must be supplied in the call to **BSAInit**. For Generate mode, the ADSM 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=Prompt, then the ADSM node and password (security token) must be supplied. The session owner can be whatever name you choose. 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=Generate, then no value can be supplied for `BSAObjOwner` or `AppObjOwner`. These fields must be empty strings. The node name used will be the machine name and the session owner will be the login user's name. The security token field is ignored in this situation.

If an application passes node and/or session owner values when the mode is Generate, it gets the return code `ADSM_RC_PSWD_GEN`. In this case, if your application supports PASSWORDACCESS=Generate, **BSAInit** must be reissued with empty ObjectOwner fields. If your application requires PASSWORDACCESS=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 used for the session. If **dsm.sys** has PASSWORDACCESS=Generate, these values will be node = hostname and owner = login user.

When using PASSWORDACCESS=Generate, the first ADSM session must be initiated by the root user. This is necessary to create the file where the password is stored.

Sessions are terminated by a **BSATerminate** call. This causes the X/Open API to close any connection with the ADSM server and free all resources associated with this session.

Note: Only one session can be active per invocation of the API. However, applications on UNIX platforms can circumvent this restriction by running with multiple processes, each process owning its own ADSM session.

Application Design Considerations

If the end user has set PASSWORDACCESS=Generate in the client options file, and is not the root user, then the Trusted Communication Agent (**dsmapitca**) child process is forked to manage the session with the ADSM server. The SIGCLD signal is used during termination. If PASSWORDACCESS=Prompt, no child process is used.

Session Security

ADSM, being a session-based system, has security components that allow applications to initialize sessions in a secure manner. These security measures prohibit unauthorized access to the server and help insure system integrity.

Every session that is started with the server has to go through a sign-on process. This sign-on process requires the use of 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. Thus, if a **BSAInit** call fails with the password expired return code (`BSA_RC_TOKEN_EXPIRED`), the password must be updated before the session can be successfully established.

Only the root session owner can change the password. First, issue 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 the section “Identifying the Object” on page 165 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 initialized.

Determining the Session Parameters

After **BSAInit** has been called to start a session, the application can issue a call to **BSAGetEnvironment** to determine the parameters set for the session. **BSAGetEnvironment** 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 ADSM node name. This corresponds to the `BSAObjOwner` field when PASSWORDACCESS is set to Prompt. When PASSWORDACCESS is Generate, this field contains the machine name.

The `objectOwner.appObjectOwner` field contains the ADSM owner name. This corresponds to the `AppObjOwner` field when PASSWORDACCESS is Prompt. When PASSWORDACCESS is 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);
```

The format of the output is:

```
envStrs[0] = "DSMSRVR=xxx"
envStrs[1] = "MAXOBJ=xx"
```

where:

- DSMSRVR is the ADSM server name.
- MAXOBJ is the number of objects that can be created within a single transaction.

Associating a Management Class With Objects

One of the key features offered by ADSM is the use of policy (management classes) to define how objects are stored and managed in ADSM storage. A *management class* is associated with an object when the object is backed up or archived. This management class determines the following:

How frequently objects are backed up
How many versions of the object are retained if backed up
How long to keep archive copies
Where the object is to be inserted in the storage hierarchy on the server

Management classes have two components: a *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 being backed up or archived. Thus, 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 critical component of ADSM, the API requires that all objects sent to the server first be assigned to a management class. There are two ways to do this:

- By using the *include-exclude list* — ADSM uses an include-exclude list to perform management class binding. The **BSACreateObject** and **BSAResolveLifecycleGroup** calls check the object being 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 no management class is specified or the object is not explicitly listed in the include-exclude list, the object is assigned to the default management class.

- By overriding 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 that 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 ADSM storage by the X/Open API is done within the bounds of a *transaction*. This provides a high level of data integrity for the ADSM product, but it does impose some restrictions that an application client must take into consideration.

A transaction is initiated by a call to **BSABeginTxn** and ended by a call to **BSAEndTxn**.

A single transaction is an atomic action. Data sent or received within the bounds of a transaction is either all committed at the end of the transaction or all rolled back if the transaction ends prematurely.

ADSM 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 need to send an object to or receive it from a different destination than the previous object, you must 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.

ADSM 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. A transaction can be halted at any time by either the server or the client. Thus, the application client must be prepared to handle sudden transaction ends that it did not initiate.

Querying the ADSM System

The X/Open API allows an application client to query an ADSM 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” on page 156).

A query operation consists of the following steps:

1 Issue the **BSABeginTxn** call to initiate 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, you can specify the description in the `desc` field and set the upper and lower bounds of the create and expiration times in the fields `createTimeLB`, `createTimeUB`, `expireTimeLB`, and `expireTimeUB`.

3 Issue the **BSAQueryObject** call.

To start the query operation, issue 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 on page 158.
- `BSA_RC_NO_MATCH` — No objects satisfied the search parameters. Go to step 5 on page 158.

4 Issue the **BSAGetNextQueryObject** call.

If more than one object satisfied the query parameters, then a **BSAGetNextQueryObject** call must be issued 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`, issue the **BSAGetNextQueryObject** call again. If there is no more data, go to the next step.

5 Issue the **BSAEndTxn** call to end the transaction.

When all query data has been retrieved or no further query data is desired, the **BSAEndTxn** call must be issued to end the transaction and terminate 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.

Flow Chart

The flow chart for performing query operations is shown in Figure 46.

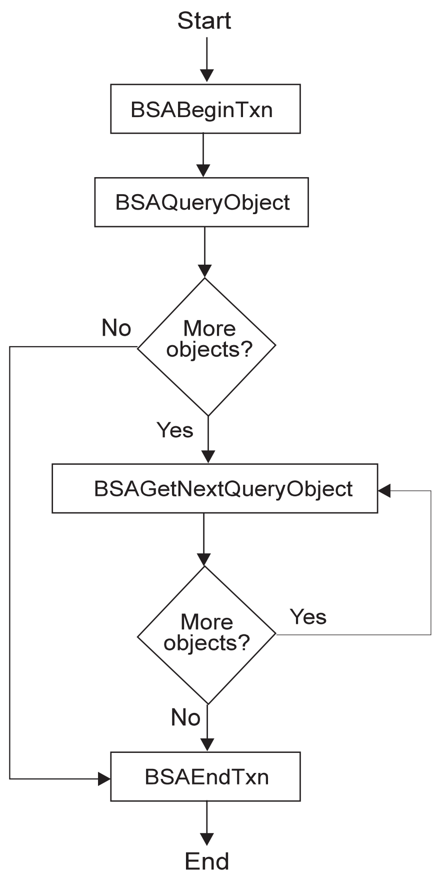

Figure 46. Flow Chart for Query Operations

Sending Data to a Server

The X/Open API allows application clients to send data to ADSM server storage. Data can be either backed up or archived. All send operations must be done within the bounds of a transaction (see "The Transaction Model" on page 156).

The *backup* component of the ADSM system supports multiple versions of named objects stored on the server. Thus, any object backed up to the server that has the same name as an object 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 are assigned to active and inactive objects on the server as defined by the management class constructs.

The *archive* component of the ADSM system allows objects to be stored 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 allows an application to archive the same object multiple times, but with different expiration times assigned to each copy of the object.

The value of the COMPRESSION option in the end user's **dsm.sys** file determines whether ADSM will compress the object during a send operation.

Some types of data (for example, data that is already compressed) may actually get bigger when processed with the compression algorithm. When this happens, the return code `ADSM_RC_ERROR` is generated and added to the ADSM error log (**dsierror.log**). If you recognize that this may 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 Issue the **BSABeginTxn** call to initiate a transaction.

2 Issue the **BSAResolveLifecycleGroup** call.

This call is optional. Use it to associate a particular management class with an object that you are storing on the ADSM server. If you don't 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" on page 155.

3 Issue the **BSACreateObject** call.

The **BSACreateObject** call takes an `ObjectDescriptor` structure as an input parameter. This structure contains information about the object being stored, such as the object's name and whether it is being backed up or archived.

The `ObjectDescriptor.Owner.bsaObjectOwner` value must match the value used on the **BSAInit** call. The `ObjectDescriptor.Owner.appObjectOwner` value must also match the one used on the **BSAInit** call, unless it was an empty string, signifying 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 ADSM. These sizes are determined by the constants ADSM_MAX_OBJINFO and ADSM_MAX_DESC in the **custom.h** header file.

BSACreateObject can also send the first block of data to the ADSM server. If the object has more data, go to the next step. If there is no more data, go to step 5.

4 Issue the **BSASendData** call.

Repeat this call as many times as necessary until the entire object has been sent to the ADSM server.

5 Issue the **BSAEndData** call.

The **BSAEndData** call signifies there is no more data for a particular object.

6 If you want to send more than one object to the ADSM server, repeat steps 3 through 5 for each object. Note that all objects sent within the same transaction must be for the same `objectspaceName`.

ADSM 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 Issue the **BSAEndTxn** call to end the transaction.

Flow Chart

The flow chart for performing backup or archive operations within a transaction is shown in Figure 47 on page 161.

The key 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

The X/Open API allows application clients to receive data from ADSM storage using the restore and retrieve functions of the product. *Restore* accesses objects that have previously been backed up, and *retrieve* accesses objects that have previously been archived.

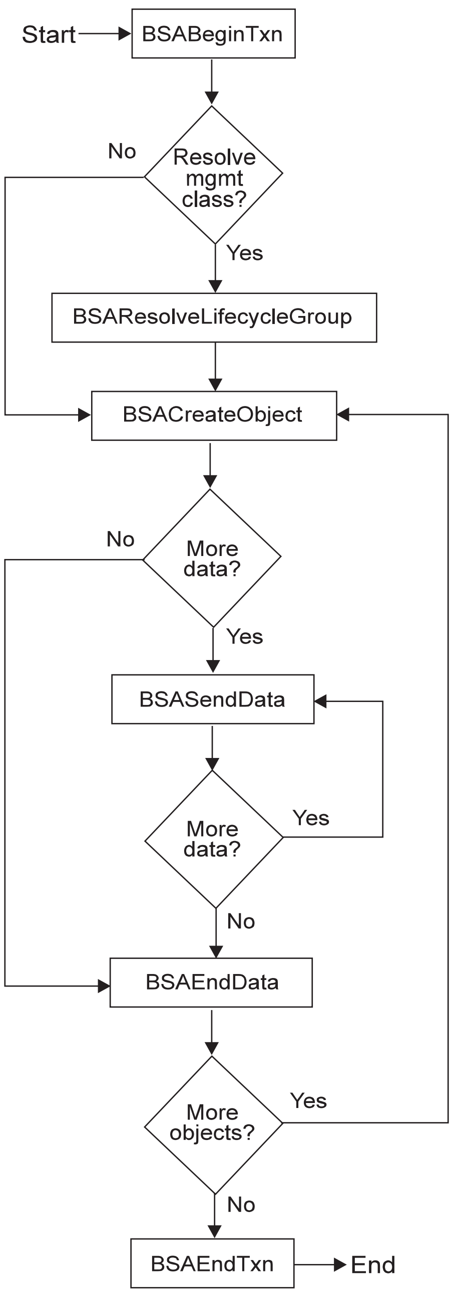

Figure 47. Flow Chart for Backup and Archive Operations

All restore and retrieve operations must be done within the bounds of a transaction (see "The Transaction Model" on page 156).

Note: Only the API can restore or retrieve objects that have been backed up or archived with API calls.

Once a session is established with the ADSM server, use the following procedure to restore or retrieve data:

1 Issue the **BSABeginTxn** call to initiate a transaction.

2 Issue the **BSAQueryObject** call to query the ADSM server for backup or archive data. (This step can be performed outside the transaction.)

Before beginning a restore or retrieve operation, you query the ADSM 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 issue the **BSAQueryObject** call with the QueryDescriptor.

If the session was initialized with a NULL owner name, the owner field need not be specified. If the session was initialized 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 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 the object descriptions.

All queries return all information that was originally stored with the object, plus the following:

copyid
The `copyid` provides an 8-byte number that uniquely identifies this object for this node in ADSM 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 ADSM storage in the most efficient manner possible. Sort the objects to be restored on this value to insure that tapes are mounted only once and are read from front to back.

You must retain some or all of the query information for later processing. Retain the `copyid` and `restoreOrder` fields because they are needed for the actual restore operation. You must also retain any other information needed to properly open a data file or identify a destination.

3 Determine the objects to be restored or retrieved 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. Thus, 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 Issue 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 being requested from the data stream.

BSAGetObject obtains the first block of data 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 Issue the **BSAGetData** call.

Repeat this call as many times as necessary until the entire object has been received from the ADSM server.

7 Issue the **BSAEndData** call.

The **BSAEndData** call signifies there is no more data for a particular object.

8 If you want to receive more than one object from the ADSM server, repeat steps 5 through 7 for each object.

9 Issue the **BSAEndTxn** call to end the transaction.

After all data for all requested objects has been received, the **BSAEndTxn** call must be issued. You can also use this call to discard any remaining data in the restore stream for objects not yet received.

Flow Chart

The flow chart for performing restore or retrieve operations is shown in Figure 48 on page 164.

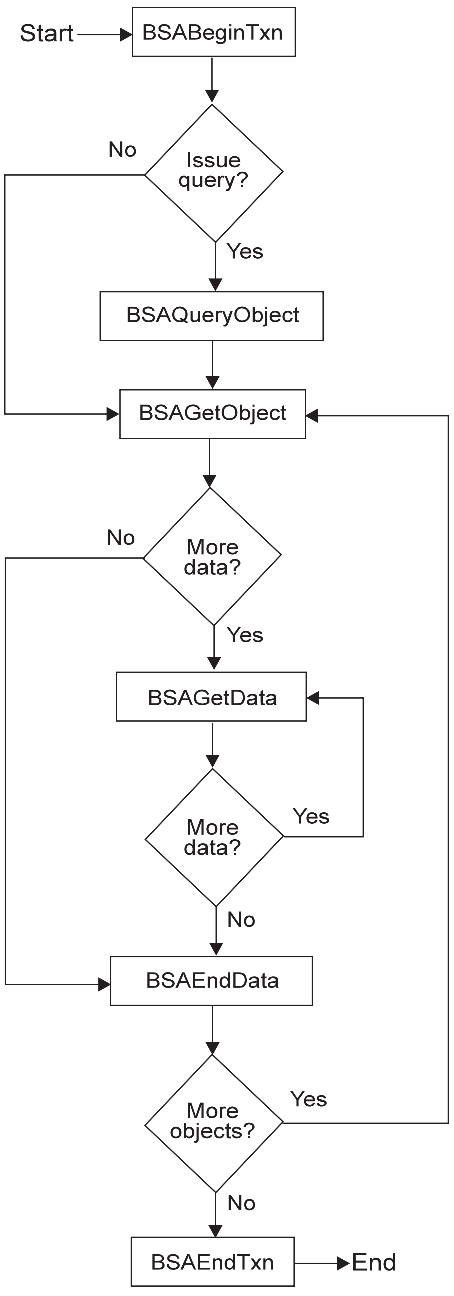

Figure 48. Flow Chart for Restore and Retrieve Operations

Deleting Objects from the Server

X/Open API applications can issue calls to either delete objects that have been archived or deactivate objects that have been backed up. The former is dependent on the node authorization given when the node was registered by an ADSM administrator. 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 ADSM 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 deactivating a backup object on the ADSM server, the object moves from an active state to an inactive state. These states have different retention policies associated with them based on the management class assigned.

Note that a call to **BSAMarkObjectInactive** affects all objects with the same `objType` and the same name.

A call to **BSADeleteObject** or to **BSAMarkObjectInactive** is always issued within the bounds of a transaction. The flow charts in Figure 49 show how a call to **BSADeleteObject** or **BSAMarkObjectInactive** is preceded by a call to **BSABeginTxn** and followed by a call to **BSAEndTxn**.

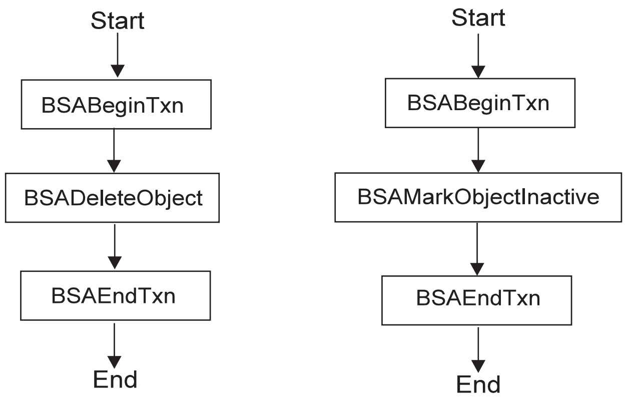

Figure 49. Flow Charts for Delete Archive (left) and Deactivate Backup (right) Operations

Identifying the Object

The ADSM 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 to meet this requirement:

- 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 defined by the system administrator. 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, keep in mind that it may be necessary to externalize the full object names to the end user. Specifically, the end user may need to specify the object in an Include or Exclude statement when the application is run.

Object Space Name

One of the most critical components of the name is the *object space name*. This name can be viewed as the name of a file system or disk drive, or any other high-level qualifier that groups related data together. ADSM uses the object space to identify the file system or disk drive the data is located on. Thus, 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 ADSM server also has administrative commands to query the object spaces on a given node in ADSM storage, and delete them if necessary. Thus, all data stored by the application client must have an object space name associated with it. Choose the name carefully to group like data together in the system.

An application client should choose 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 the client is running on. The name does not have to exist as an object on the system or bear any relation to the actual data on the local file system, but the name must meet the standard naming rules in order to be properly processed for management classes.

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. A *directory* is an object that contains only attributes.

ADSM also accepts the value `BSAObjectType_DATABASE`, but treats it as `BSAObjectType_FILE`.

Example

The following example demonstrates what the application client would code on a UNIX platform:

```
/myobjspace/pathname
```

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 initialized. This object owner value can be used to control access to the object.

If a session is initialized 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 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 initialized with a specific owner name, the session can only perform actions on objects that have that owner name associated with them. Thus, backups or archives into the system all must have this owner name associated with them. Also, 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.

Figure 50 summarizes the conditions under which a user has access to an object.

Figure 50. 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

Determining Size Limits

Certain data structures or fields in the X/Open API have size limitations. These structures are often names or other text fields that cannot exceed a predetermined length. Examples of fields with such limits are:

Archive description
Copy group destination
Copy group name
Object information size
Object owner name
Path name length
Password

These limits are defined as constants within the header files **custom.h** and **xbsa.h**. Any storage allocation should be based on these constants instead of hard-coded numbers. Refer to the header files for further information and a list of the current constants.

ADSM Changes to the XBSA Header Files

The X/Open API contains the header files **custom.h**, **xbsa.h**, and **policy.h**. ADSM uses these header files with the following changes:

Changes to custom.h

The ADSM X/Open API supports the following additional constants and return codes in **custom.h**:

```
/* XBSA library version, release, level
 */
#define BSA_API_VERSION     2
#define BSA_API_RELEASE     1
#define BSA_API_LEVEL       3

/* Constants used
 */
#define ADSM_MAX_DESC               100     /* ADSM max Desc size            */
#define ADSM_MAX_OBJINFO            100     /* ADSM max object info size     */
#define ADSM_LOWEST_BOUND           0x0000  /* value for LowerBound max      */
#define ADSM_HIGHEST_BOUND          0xFFFF  /* value for UpperBound max      */
#define ADSM_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 ADSM_RC_ERROR                       0x60 /* see ADSM error log       */
#define ADSM_RC_INVALID_NODE                0x61 /* BSAObjOwner not match Init*/
```

```
#define ADSM_RC_INVALID_COPYTYPE           0x62 /* invalid copyType          */
#define ADSM_RC_INVALID_OBJTYPE            0x63 /* invalid objectType        */
#define ADSM_RC_INVALID_STATUS             0x64 /* invalid object status     */
#define ADSM_RC_INVALID_ST_VER             0x65 /* invalid structure version */
#define ADSM_RC_OWNER_TOO_LONG             0x66 /* owner too long            */
#define ADSM_RC_PSWD_TOO_LONG              0x67 /* pswd  too long            */
#define ADSM_RC_PSWD_GEN                   0x68 /* pswd access = generate    */
```

Changes to xbsa.h

The ADSM 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.                   */
```

Changes to policy.h

The ADSM 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.                                          */
```

Glossary

This glossary defines important terms and abbreviations used in the ADSM library of books. If you cannot find the term you are looking for, see the Index or the *IBM Dictionary of Computing*.

This glossary may include terms and definitions from:

- The *IBM Dictionary of Computing*, ZC20-1699, copyright 1994 by McGraw-Hill.
- The *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036.
- The *Information Technology Vocabulary*, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC2/SC1).

A

absolute. A copy group mode value that indicates that an object is considered for backup even if it has not changed since the last time it was backed up. See *mode*. Contrast with *modified*.

active policy set. The policy set within a policy domain that contains the most recently activated policy. This policy set is used by all client nodes assigned to the current policy domain. See *policy set*.

active version. The most recent backup copy of an object stored in ADSM storage for an object that currently exists on a file server or workstation. An active version remains active and exempt from deletion until it is replaced by a new backup version, or ADSM detects during a backup that the user has deleted the original object from a file server or workstation.

administrative client. A program that runs on a file server, workstation, or mainframe. This program allows administrators to control and monitor ADSM servers through ADSM administrator commands. Contrast with *backup-archive client*.

administrator. A user who has been registered to the server as an administrator. Administrators can be assigned one or more privilege classes. Administrators can use the administrative client to enter ADSM server commands and queries according to their privileges.

ADSM. ADSTAR Distributed Storage Manager.

ADSTAR Distributed Storage Manager (ADSM). A client/server program that provides storage management and data access services to customers in a multivendor computer environment.

Advanced Program-to-Program Communications (APPC). An implementation of the SNA LU6.2 protocol that allows interconnected systems to communicate and share the processing of programs. See *Systems Network Architecture Logical Unit 6.2* and *Common Program Interface Communications*.

API. application program interface.

APPC. Advanced program-to-program communications.

application client. A software application that runs on a workstation or personal computer and uses the ADSM application program interface (API) function calls to back up, archive, restore, and retrieve objects. Contrast with *backup-archive client*.

application program interface (API). A set of functions that application clients can call to store, query, and retrieve data from ADSM storage.

archive. A function that allows users to copy one or more objects to a long-term storage device. Archive copies may be accompanied by descriptive information, may imply data compression software usage, and may be retrieved by archive date, object name, or description. Contrast with *retrieve*.

archive copy. An object or group of objects residing in an archive storage pool in ADSM storage.

archive copy group. A policy object that contains attributes which control the generation, destination, and expiration of archive objects. The archive copy group belongs to a management class.

archive retention grace period. The number of days ADSM retains an archive copy when the server is unable to rebind the object to an appropriate management class.

authentication. The process of checking and authorizing a user's password before allowing that user access to the ADSM server. Authentication can be turned on or off by an administrator with system privilege.

authorization rule. A specification that allows another user to either restore or retrieve a user's objects from ADSM storage.

B

backup. A function that allows users to copy one or more objects to a storage pool to protect against data loss. Contrast with *restore*.

backup-archive client. A program that runs on a file server, PC, or workstation and provides a means for ADSM users to back up, archive, restore, and retrieve objects. Contrast with *application client* and *administrative client*.

backup copy group. A policy object that contains attributes which control the generation, destination, and expiration of backup files. The backup copy group belongs to a management class.

backup retention grace period. The number of days ADSM retains a backup version when the server is unable to rebind the object to an appropriate management class.

backup version. An object, directory, or file space that a user has backed up that resides in a backup storage pool in ADSM storage. Though there may be more than one backup version of an object in ADSM storage, only one is considered the active version. See *active version* and *inactive version*.

bindery. A database that consists of three system files for a NetWare 3.11 server. The files contain user IDs and user restrictions.

binding. The process of associating an object with a management class name.

C

client. A program running on a file server or workstation that requests services of another program called the server.

client node. A file server or workstation on which the client program has been installed that has been registered with the server.

client options file. A user-editable file that contains processing options to identify ADSM servers, select communication methods, specify backup, archive, restore, and retrieve options, define scheduling options, and to choose formats for date, time, and numbers. Also called the *dsm.opt* file.

client/server. A communications network architecture in which one or more programs (clients) request computing or data services from another program (the server).

client system options file. A file, used on UNIX clients, that contains a number of processing options which identify the ADSM servers to be contacted for services. This file also specifies communications options, backup and archive processing options, and scheduling options. Also called the *dsm.sys* file. See *client user options file*.

client user options file. A user-editable file, used on UNIX clients, that contains processing options to identify the ADSM server to contact, to specify backup, archive, restore, and retrieve options, and to select formats for date, time, and numbers. Also called the *dsm.opt* file. See *client system options file*.

Common Programming Interface Communications (CPIC). A programming interface that allows program-to-program communication using SNA LU6.2. See *Systems Network Architecture Logical Unit 6.2*.

communication method. The method by which a client and server exchange information. The UNIX application client can use the TCP/IP or SNA LU6.2 method. The Windows application client can use the 3270, TCP/IP, NETBIOS, or IPX/SPX method. The OS/2 application client can use the 3270, TCP/IP, PWSCS, SNA LU6.2, NETBIOS, IPX/SPX, or Named Pipe method. The Novell NetWare application client can use the IPX/SPX, PWSCS, SNA LU6.2, or TCP/IP methods. See *IPX/SPX*, *Named Pipe*, *NETBIOS*, *Programmable Workstation Communication Service*, *Systems Network*

Architecture Logical Unit 6.2, and *Transmission Control Protocol/Internet Protocol*.

communication protocol. A set of defined interfaces that allows computers to communicate with each other.

configuration file. An optional file pointed to by your application that can contain the same options that are found in the client options file (for non-UNIX platforms) or in the client user options file and client system options file (for UNIX platforms). If your application points to a configuration file and values are defined for options, then the values specified in the configuration file override any value set in the client options files.

copy group. A policy object that contains attributes which control the generation, destination, and expiration of backup and archive files. There are two kinds of copy groups: backup copy group and archive copy group. Copy groups belong to management classes. See *frequency*, *destination*, *mode*, *retention*, *serialization*, and *version*.

CPIC. Common Programming Interface Communications.

D

default management class. A management class assigned to a policy set that is used to govern backed up or archived objects when a user does not explicitly associate an object with a specific management class through the include-exclude list.

destination. A copy group attribute that specifies the storage pool to which an object will be backed up or archived. At installation, ADSM provides two storage destinations named BACKUPPOOL and ARCHIVEPOOL.

domain. See *policy domain*.

dsm.opt file. See *client options file* and *client user options file*.

dsm.sys file. See *client system options file*.

dynamic. A copy group serialization value that specifies that ADSM accepts the first attempt to back up or archive an object, regardless of any changes made during backup or archive processing. See *serialization*. Contrast with *shared dynamic*, *shared static*, and *static*.

E

error log. A text file (dsierror.log) written on disk that contains ADSM processing error messages.

exclude. The process of identifying objects in an include-exclude list to prevent them from being backed up.

expiration. The process by which objects are identified for deletion because their expiration date or retention period has passed. Backed up or archived objects are marked for deletion based on the criteria defined in the backup or archive copy group.

F

file server. A dedicated computer and its peripheral storage devices that are connected to a local area network that stores both programs and files that are shared by users on the network.

file space. A logical space on the ADSM server that contains a group of files. In ADSM, users can restore, retrieve, or delete file spaces from ADSM storage. On UNIX systems, a file space is a logical space that contains a group of objects backed up or archived from the same file system.

frequency. A copy group attribute that specifies the minimum interval, in days, between backups.

fuzzy backup. A backup version of an object that might not accurately reflect what is currently in the object because ADSM backed up the object while the object was being modified.

fuzzy copy. An archive copy of an object that might not accurately reflect what is currently in the object because ADSM archived the object while the object was being modified.

I

inactive version. A copy of a backup file in ADSM storage that either is not the most recent version or the corresponding original object has been deleted from the client file system. Inactive backup versions are eligible for expiration according to the management class assigned to the object.

include-exclude file. A file, on UNIX clients, that contains statements which ADSM uses to determine whether to back up certain objects and to determine the associated management classes to use for backup or archive. See *include-exclude list*.

include-exclude list. A list of INCLUDE and EXCLUDE options that include or exclude selected objects for backup. An EXCLUDE option identifies objects that should not be backed up. An INCLUDE option identifies objects that are exempt from the exclusion rules or assigns a management class to an object or a group of objects for backup or archive services. The include-exclude list is defined either in the include-exclude file (for UNIX clients) or in the client options file.

IPX/SPX. Internetwork Packet Exchange/Sequenced Packet Exchange. IPX/SPX is Novell NetWare's proprietary communication protocol.

L

Local Area Network (LAN). A variable-sized communications network placed in one location. It connects servers, PCs, workstations, a network operating system, access methods, and communications software and links.

M

management class. A policy object that is a named collection of copy groups. A management class is associated with an object to specify how the server should manage backup versions or archive copies of workstation objects. See *binding* and *copy group*.

mode. A copy group attribute that specifies whether a backup file should be created for an object that was not modified since the last time the object was backed up. See *absolute* and *modified*.

modified. A backup copy group attribute that indicates that an object is considered for backup only if it has been changed since the last backup. An object is considered changed if the date, size, owner, or permissions have changed. See *absolute* and *mode*.

N

Named Pipe. A type of interprocess communication which allows message data streams to be passed between peer processes, such as between a client and a server.

NETBIOS. Network Basic Input/Output System. An operating system interface for application programs used on IBM personal computers that are attached to the IBM Token-Ring Network.

NetWare Loadable Module (NLM). Novell NetWare software that provides extended server functionality. Support for various ADSM and NetWare platforms are examples of NLMs.

node. See *client node*.

node name. A unique name used to identify a workstation, file server, or PC to the server.

O

object. A collection of data managed as a single entity.

owner. The owner of backup-archive files sent from a multi-user client node, such as AIX.

P

policy domain. A policy object that contains one or more policy sets. Client nodes are associated with a policy domain. See *policy set*, *management class*, and *copy group*.

policy set. A policy object that contains a group of management class definitions that exist for a policy domain. At any one time, there can be many policy sets within a policy domain, but only one policy set can be active. See *active policy set* and *management class*.

Programmable Workstation Communication Services (PWSCS). A product that provides transparent high performance communications between programs running on workstations or on host systems.

PWSCS. Programmable Workstation Communication Services.

R

registration. The process of identifying a client node or administrator to the server by specifying a user ID, password, and contact information. For client nodes, a policy domain, compression status, and deletion privileges are also specified.

restore. A function that allows users to copy a version of a backup file from the storage pool to a workstation or file server. The backup copy in the storage pool is not affected. Contrast with *backup*.

retention. The amount of time, in days, that inactive backed up or archived files are retained in the storage pool before they are deleted. The following copy group attributes define retention: retain extra versions, retain only version, retain version.

retrieve. A function that allows users to copy an archive file from the storage pool to the workstation or file server. The archive copy in the storage pool is not affected. Contrast with *archive*.

S

selective backup. A function that allows users to back up objects from a client domain that are not excluded in the include-exclude list and that meet the requirement for serialization in the backup copy group of the management class assigned to each object.

serialization. A copy group attribute that specifies whether an object can be modified during a backup or archive operation and what to do if it is. See *static*, *dynamic*, *shared static*, and *shared dynamic*.

server. A program running on a mainframe, workstation, or file server that provides shared services such as backup and archive to other various (often remote) programs called clients.

session. A period of time in which a user can communicate with a server to perform backup, archive, restore, or retrieve requests.

shared dynamic. An ADSM copy group serialization mode. This mode specifies that if an object changes during backup or archive and continues to be changed after a number of retries, the last retry commits the object to the ADSM server whether or not it changed during backup or archive.

shared static. A copy group serialization value that specifies that an object must not be modified during a backup or archive operation. ADSM attempts to retry the operation a number of times. If the object is in use during each attempt, the object is not backed up or archived. See *serialization*. Contrast with *dynamic*, *shared static*, and *static*.

SNA LU6.2. Systems Network Architecture Logical Unit 6.2.

static. A copy group serialization value that specifies that an object must not be modified during a backup or archive operation. If the object is in use during the first attempt, ADSM will not back up or archive the object. See *serialization*. Contrast with *dynamic*, *shared static*, and *static*.

storage pool. A named set of storage volumes that is used as the destination of backup or archive copies.

Systems Network Architecture Logical Unit 6.2 (SNA LU6.2). A set of rules for data to be transmitted in a network Application programs communicate with each other using a layer of SNA called Advanced Program-to-Program Communication (APPC).

T

TCP/IP. Transmission Control Protocol/Internet Protocol

Transmission Control Protocol/Internet Protocol (TCP/IP). A standard set of communication protocols that supports peer-to-peer connectivity of functions for both local and wide-area networks.

V

version. (1) A three-part designation for an instance of the API, consisting of the version, release, and level. (2) The maximum number of different backup copies of files retained for files. The following backup copy group attributes define version criteria: version data exists and versions data deleted.

W

wildcard character. A character such as an asterisk (*) or question mark (?) that is used to search for various combinations of alphanumeric and symbolic names. These names can reflect object names or character strings within a file, for example.

workstation. A programmable high level workstation (usually on a network) with its own processing hardware such as a high-performance personal computer. In a local area network, a personal computer that acts as a single user or client. A workstation can also be used as a server.

Index

Communicating Your Comments to IBM

ADSTAR Distributed Storage Manager
Using the Application
Program Interface
Version 2

Publication No. SH26-4002-00

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:
  - United States: 1-800-426-6209
  - Other countries: (+1)+408+256-7896
- If you prefer to send comments electronically, use this network ID:
  - IBMLink from U.S. and IBM Network: STARPUBS at SJEVM5
  - IBMLink from Canada: STARPUBS at TORIBM
  - IBM Mail Exchange: USIB3VVD at IBMMAIL
  - Internet: starpubs@vnet.ibm.com

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies.

Readers' Comments — We'd Like to Hear from You

ADSTAR Distributed Storage Manager
Using the Application
Program Interface
Version 2

Publication No. SH26-4002-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name	Address
Company or Organization	
Phone No.	

Readers' Comments — We'd Like to Hear from You
SH26-4002-00

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL   PERMIT NO. 40   ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
RCF Processing Department
G26/050
5600 Cottle Road
SAN JOSE, CA 95193-0001

Fold and Tape **Please do not staple** Fold and Tape

SH26-4002-00

Cut or Fold
Along Line

Printed in U.S.A.

SH26-4002-00