Quantum

API Guide API Guide API Guide API Guide API Guide

StorNext® SNAPI 2.0.1

StorNext

6-01375-05

StorNext API Guide, 6-01375-05, Ver. A, Rel. 2.0, April 2008, Made in USA.

COPYRIGHT STATEMENT

TRADEMARK STATEMENT

Contents

Chapter 1
Introduction

About This Guide

This guide contains information and instructions necessary to install and use the StorNext APIs (SNAPI). This guide is intended for system administrators, programmers, and anyone interested in learning about installing and using the StorNext APIs.

The StorNext API guide is divided into the following chapters:

- StorNext Storage Manager APIs
- StorNext File System APIs
- File System API Example
- Storage Manager API Example

StorNext API (SNAPI) contains StorNext Storage Manager APIs, StorNext File System APIs, and Client APIs that can be used to make calls to third-party applications, resulting in enhanced operations between third-party applications and StorNext.

Product Compatibility

The StorNext API (SNAPI) release is compatible with a specific StorNext release. Please refer to the following compatibility matrix to verify you are running compatible versions of both products.

SNAPI/StorNext Compatibility Matrix

StorNext API (SNAPI) Version	StorNext Version
SNAPI 1.0	StorNext 2.8.1
SNAPI 1.1.0	StorNext 3.0
SNAPI 2.0	StorNext 3.0.2, 3.1
SNAPI 2.0.1	StorNext 3.1.1, 3.1.2

Installing StorNext APIs

The StorNext File System APIs are automatically installed when you install the StorNext software.

The StorNext Storage Manager APIs on the CD must be installed before you can call them. For installation instructions, see Installing the APIs on the Storage Manager Host in the chapter StorNext Storage Manager APIs.

Running APIs Remotely

Beginning with SNAPI 2.0, you can now run StorNext Storage Manager APIs remotely from a client. Functionally, operation from a remote client is identical to running the APIs locally. If you want to use this feature, remote clients must be installed and configured as described in Installing the APIs on a Remote Client and Configuring SNAPI in the chapter StorNext Storage Manager APIs.

StorNext File System APIs continue to run remotely on clients as they always have.

Explanation of Warnings, Cautions and Notes

The following cautions, and notes appear throughout this document to highlight important information.

Caution: Indicates a situation that may cause possible damage to equipment, loss of data, or interference with other equipment.

Note: Indicates important information that helps you make better use of your system.

Related Documents You Might Need

The following documents available for StorNext can be found at www.quantum.com/manuals.

- StorNext User's Guide (6-01658-02)
- StorNext File System Quick Reference Guide (6-00361-21)
- StorNext Storage Manager Quick Reference Guide (6-00361-22)

Getting More Information or Help

More information about this product is available on the Customer Service Center website at www.quantum.com/csc. The Customer Service Center contains a collection of information, including answers to frequently asked questions (FAQs). You can also access software, firmware, and drivers through this site.

Quantum Technical Assistance Center

For further assistance, or if training is desired, contact the Quantum Technical Assistance Center:

North America:	1+800-284-5101
UK, France and Germany:	00800 4 QUANTUM
EMEA:	+44 1256 848 766
Worldwide Web:	www.quantum.com/support

Chapter 2

StorNext Storage Manager APIs

Introduction

This chapter describes the application programming interfaces (APIs) available for StorNext Storage Manager.

This chapter contains the following major topics:

- Installing the APIs on the Storage Manager Host
- Installing the APIs on a Remote Client
- Configuring SNAPI
- SNAPI Logs and Health Checks
- Use Cases
- API Descriptions and Arguments

Note: Unlike the StorNext File System APIs which are automatically installed with the StorNext software, you must install the Storage Manager APIs from the CD by following the installation instructions in Installing the APIs on the Storage Manager Host on page 6.

Throughout this chapter, the terms *library* and *archive* are used. *Library* refers to an actual physical library (e.g., with robotics), whereas *archive* is a more general term that may refer to a library, a vault, or anywhere else where media can be stored.

A vault is a logical archive type. It is any location that contains copies of media that have been removed from the operational system. Since a vault has no robotics, if a vault has tape drives, tape media must be manually mounted to the drive and dismounted from the drive.

Installing the APIs on the Storage Manager Host

This section describes how to install the SNAPI Storage Manager APIs and the SNAPI Server on the StorNext Storage Manager host. After installation is complete, the SNAPI Server will listen for requests from all local and remote SNAPI clients. The SNAPI Server will log operational and error messages to /usr/adic/SNAPI/logs/. The SNAPI Server will also handle all local and remote requests from snclix, the SNAPI command line interface utility.

1. Log on as root.
2. Insert and mount the product CD that contains the StorNext Storage Manager APIs.
3. Use the cd command to navigate to the directory that corresponds to your operating system:
   - SunOS590sparc_32
   - SunOS5100sparc_32
   - RedHat40AS_26ia64
   - RedHat40AS_26x86_32
   - RedHat40AS_26x86_64
   - SuSE90ES_26x86_64
   - SuSE90ES_26ia64
   - SuSE90ES_26x86_32
   - SuSE100ES_26x86_64
   - SuSE100ES_26ia64

- SuSE100ES_26x86_32

4. From the operating system directory, run the program install.snapi. This program automatically decompresses the APIs and copies them to your StorNext directory in the following location: /usr/adic/SNAPI.
5. You can access this API guide by locating the pdf file on the StorNext API CD .

   After installing the Storage Manger APIs, in order to successfully compile and link with your applications you need the location of the main header file and the libraries Quantum delivers.

   The main header file location: /usr/adic/SNAPI/inc/API.hh

   The dynamic library location: /usr/adic/SNAPI/lib/libsnapi.so

   The static library location: /usr/adic/SNAPI/lib/static/libsnapi.a

Installing the APIs on a Remote Client

This section describes how to install the SNAPI Storage Manager APIs on a remote client connected to a SNAPI Server host. As a prerequisite, SNAPI must have already been installed on the Storage Manager host as described in Installing the APIs on the Storage Manager Host.

To install a remote client:

1. Copy the script /usr/adic/SNAPI/bin/install.snapiclient from the Storage Manager host to the remote host.

   Note: To ensure that the install.snapiclient script runs properly, before proceeding verify that RCP is enabled between the remote client host and the StorNext Storage Manager host.

2. Run the install.snapiclient script on the remote host, passing it the name of the Storage Manager host. For example:
   install.snapiclient <SNAPI_Server_Host>

The installation script also installs the snclix utility, which can be used to run SNAPI API commands on the command line.

Note: An upgrade of SNAPI on a remote client is accomplished by running the installation procedure described above. To remove SNAPI from a remote client, simply remove all contents of the /usr/adic/SNAPI/ directory.

Configuring SNAPI

SNAPI configuration for both local and remote clients is accomplished by editing the /usr/adic/SNAPI/config/snapi.cfg file on the SNAPI server and remote hosts, respectively. The snapi.cfg file allows you to specify the following:

- SNAPI server names
- INET socket port value for remote clients
- Client timeout value. This value specifies how long the client should wait for a response from the SNAPI server. This value can be between 0-1000 minutes. The default value is 30 minutes.

You are not required to modify the snapi.cfg file before running the APIs unless you are enabling HA support for remote clients, or in other special cases. (See HA Failover Support for more information about enabling HA support.)

Snapi.cfg Example

The following example describes and illustrates each of the parameters in the snapi.cfg file. (You can find the example in the snapi.cfg.default file installed at /usr/adic/SNAPI/config/snapi.cfg.default.)

```
<!--  The serverName indicates the host on which the StorNext server is
      running. If this is an HA system, add a line to indicate the
      failover host. -->
<PARAMETER name="serverName" value="localhost"/>

<!--  The serverPort is used for remote client connections. There is no
      need to edit this value unless the port is already in use, in which
      case the port value on the StorNext server must be edited also. -->
<PARAMETER name="serverPort" value="61776"/>
```

```
<!-- The clientTimeOut is the maximum time (in secs) a client will spend
     attempting to fulfill a request to the server. Set the value to 0 to
     indicate the client should try indefinitely to fulfill the request. -->
<PARAMETER name="clientTimeOut" value="1800"/>
```

HA Failover Support

HA failover support applies only to remote clients, and is supported by the snapi.cfg file on the remote hosts. To enable HA support, specify the primary and secondary SNAPI server host names in an ordered list on separate lines in the /usr/adic/SNAPI/config/snapi.cfg file.

For example:

```
<PARAMETER name="serverName" value="zeus"/>
<PARAMETER name="serverName" value="hera"/>
```

Upon detecting failover, SNAPI internally will reissue the original request to the SNAPI server on behalf of the remote client. Connection attempts to the primary and secondary servers will continue until the client timeout limit is reached.

SNAPI Logs and Health Checks

Logs are available for the SNAPI server host. These logs use the existing StorNext logging framework, and are located at /usr/adic/SNAPI/logs/tac.

You can configure SNAPI logs by adjusting the parameters located at /usr/adic/SNAPI/logs/log_params.

Health checks are also available on the SNAPI server host. Health checks are also integrated with the existing StorNext Health Check framework.

To run health checks, from the StorNext home page, choose Health Check from the Service Menu. The Health Check Test screen appears. From this screen run one or both of these health checks:

- Network: This health check validates connections to the StorNext server

- Config: This health check validates the contents of the SNAPI configuration file (snapi.cfg)

Alternatively, you can accomplish the same thing manually by running the following tests:

- snapiverifyConnectivity: Verifies connection to the StorNext server. Located at /usr/adic/SNAPI/bin/snapiVerifyConnectivity.
- snapiverifyConfig: Validates the SNAPI configuration file (snapi.cfg) contents. Located at /usr/adic/SNAPI/bin/snapiVerifyConfig.

Use Cases

There are three different programmatic entry points for accessing the StorNext Storage Manager APIs:

- Using a C or C++ Library
- Using a C or C++ Library with XML Wrapper
- Using the SNAPI XML Command Line Interface (snclix)

This section contains examples showing how to use the APIs with these methods.

Note: Data types used in these APIs are defined in the header files located here: /usr/adic/SNAPI/inc/

Using a C or C++ Library

For each API, there is a corresponding C++ class with the same name as the API. Quantum provides both dynamic and static libraries, as well as a header file to link to the application program.

The following example illustrates typical API usage for this case.

```
#include <API.hh>

#include <sys/types.h>
#include <unistd.h>
#include <iostream>
#include <sstream>
```

```
#include <string>

using namespace std;
using namespace Quantum::SNAPI;

int
main()
{
  Status status;

  try
  {
    // Create object for requesting file attributes.
    GetFileAttribute getFileAttrReq("/snfs/testFile.dat");

    // Send request to the server and get overall request status code.
    // Note: This method may throw exceptions (see catch block below).
    status = getFileAttrReq.process();

    // Check the returned status code.
    if (status.getCode() != SUCCESS)
    {
      cout << "StatusCode: " << status.getCodeAsString() << endl;
      cout << "Description: " << status.getDescription() << endl;
      cout << "LocalStatus.StatusCode: " <<
           getFileAttrReq.getLocalStatus().getCodeAsString() << endl;
      cout << "LocalStatus.Description: " <<
           getFileAttrReq.getLocalStatus().getDescription() << endl;
    }
    else
    {
      // Get the requested data.
      FileInfo fileInfo = getFileAttrReq.getFileInfo();

      cout << "FileName:        " << fileInfo.getFileName() << endl;
      cout << "Location:        " << fileInfo.getLocationAsString() << endl;
      cout << "Existing Copies: " << fileInfo.getNumberOfExistingCopies() << endl;
      cout << "Target Copies:   " << fileInfo.getNumberOfTargetCopies() << endl;

      MediaList media = fileInfo.getMedia();

      for (int i=0; i<media.size(); i++)
      {
        cout << "Media ID:        " << media[i] << endl;
      }
    }
  }
  catch (SnException& exception)
  {
    switch (exception.getCode())
    {
      case SUBFAILURE:
      case FAILURE:
      case SYNTAXERROR:
      default:
      {
        cout << "Exception code:   " << exception.what() << endl;
        cout << "Exception detail: " << exception.getDetail() << endl;
      }
      break;
    }
  }
  catch (...)
  {
    cout << "Caught unknown exception." << endl;
  }

  return status.getCode();
}
```

Using a C or C++ Library with XML Wrapper

For this approach there is a single function called doXML available to the third-party program. This function's input and output are string type, and their contents are in XML format. Different APIs have different specifications on the XML input and output, as described for each API.

Note: It is your responsibility to provide an input string that satisfies the specification. Otherwise, a failure from the doXML function call might result.

The following example illustrates typical API usage for this case.

```
#include <API.hh>

#include <sys/types.h>
#include <unistd.h>
#include <iostream>
#include <sstream>
#include <string>
```

```
using namespace std;
using namespace Quantum::SNAPI;

int
main()
{
  Status status;

  // Initialize input command in XML format.
  XML xmlIn("<?xml version=\"1.0\"?>"
        "<COMMAND name=\"GetFileAttribute\">"
        "   <ARGUMENT name=\"fileName\" value=\"/snfs/testFile.dat\"/>"
        "</COMMAND>");

  XML xmlOut;

  try
  {
    // Perform the request.
    status = doXML(xmlIn, xmlOut);
  }
  catch (SnException& excptn)
  {
    cerr << "Exception code:   " << excptn.what() << endl;
    cerr << "Exception detail: " << excptn.getDetail() << endl;
    status = excptn.getCode();
  }

  // Stream the results to standard out.
  cout << xmlOut << endl;

  return status.getCode();
}
```

Using the SNAPI XML Command Line Interface (snclix)

The snclix utility is a command line utility that provides an interface for SNAPI APIs from the SNAPI server and remote client hosts. This utility operates on XML input and output, and is very similar to the doXML() function. Both of these take an XML request definition as input, and return an XML response with requested information embedded in the response.

Note: It is your responsibility to generate a properly formatted XML request and to subsequently access the relative portions of information in the XML response.

Any XML information that is not relevant to the specified request will cause an error. This also applies to cases where single arguments are expected but multiple arguments are supplied.

XML character data must comply with Section 2.4 of the XML specification. In particular, when used in value strings, the following characters must be escaped using their ASCII numeric character references:

& = "&#x26;"
< = "&#x3C;"
> = "&#x3E;"

In addition, any non-printing characters (such as new line, tab, etc.) must be escaped using their ASCII numeric character references.

When using the snclix utility, after you create a file containing properly formatted XML (or optionally an input stream,) you then pass the information to the snclix executable program.

Internally, snclix calls the doXML() function to process the request and send the resulting XML response to an output file (or optionally streams it to stdout).

The snclix utility is located at /usr/adic/SNAPI/bin/snclix and has the following usage format:

```
snclix [-i infile] [-o outfile] [-c config]
```

-i infile:	Specifies an input file. (File contents should be in XML format.)
-o outfile:	Specifies an output file. (File contents will be in XML format.) If an outfile exits, it will be overwritten.
-c config:	Specifies a configuration file.

Example of snclix Usage

```
snclix -i /tmp/GetFileAttributeRequest.xml -o /tmp/
GetFileAttributeResponse.xml
```

Where the /tmp/GetFileAttributeRequest.xml file contains the following input:

```
<?xml version="1.0"?>
<COMMAND name="GetFileAttribute">
    <ARGUMENT name="fileName" value="/snfs/testFile.dat"/>
</COMMAND>
```

And the /tmp/GetFileAttributeResponse.xml file contains the following output:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="GetFileAttribute" statusCode="0" status="SUCCESS">
    <FILEINFO statusCode="0" status="SUCCESS" statusDescription="Command Successful">
      <INFO name="fileName" value="/snfs1/testFile.dat" />
      <INFO name="location" value="DISK AND TAPE" />
      <INFO name="numberExistingCopies" value="1" />
      <INFO name="numberTargetCopies" value="1" />
      <INFO name="mediaID" value="000082(1)" />
     </FILEINFO>
</RESPONSE>
```

API Descriptions and Arguments

A brief description is provided for each API, as well as its input arguments and output variables. Also provided for each API is an example showing XML input and output, and the corresponding C++ class declaration.

The following StorNext Storage Manager APIs are described in this section:

- Backup
- CheckoutMedia
- CleanMedia
- CopyMedia
- EjectMedia
- EnterMedia
- FileRetrieve

- GetArchiveCapacity
- GetArchiveList
- GetBackupStatus
- GetDriveList
- GetFileAttribute
- GetFileTapeLocationI
- GetMediaList
- GetMediaStatus
- GetPolicy
- GetPortList
- GetSchedule
- GetSystemStatus
- MoveMedia
- PassThru
- RmDiskCopy
- SetArchiveState
- SetDirAttributes
- SetDriveState
- SetFileAttributes
- SetMediaState
- SetPolicy
- SetSchedule

For the C++ class declaration section, all API classes inherit from class Request, which has a public member function process(). The third-party program must call this function to have the API request processed. See the example code in Appendix B.

Note: The process() function should be called only once for each instantiation of an API object. API users should instantiate a new API object every time they need to call the process() function.

When you enter arguments for the StorNext Storage Manager APIs, wildcard characters such as the asterisk (*) are not supported. For example, when you run the FileRetrieve API you cannot enter '*' when entering the <filename> parameter.

Backup

This API initiates a backup operation.

Input

NA. This API has no command arguments.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Initiate a system backup operation. -->
<COMMAND name="Backup"/>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="Backup" statusCode="0" status="SUCCESS"
                 statusDescription="Backup initiation successful">
   <STATUSDETAIL statusCode="0" status="SUCCESS"
                 statusDescription="Backup initiation successful"/>
</RESPONSE>
```

C++ Class Declaration

```
class Backup : public Request
{
public:
    /// Default and Primary Contructor
    Backup();

    /// Method to return the local status
    const Status& getLocalStatus() const;
}
```

CheckoutMedia

This API allows you to check out media with the specified media IDs.

Input

mediaID: The ID of the media you want to check out. You can specify multiple media IDs to check out multiple media.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Checks out all specified media from the system. -->
<COMMAND name="CheckOutMedia">
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- mediaID : valid media ID -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- One or more mediaID must be specified. -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <ARGUMENT name="mediaID" value="025311"/>
    <ARGUMENT name="mediaID" value="025312"/>
```

```
</COMMAND>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="CheckOutMedia" statusCode="0" status="SUCCESS"
                                statusDescription="Command Successful">
    <!-- A status detail element will be returned for each media -->
      <STATUSDETAIL name="mediaID" value="025311"
                                statusCode="0" status="SUCCESS"
                                statusDescription="Command Successful"/>
      <STATUSDETAIL name="mediaID" value="025312"
                                statusCode="0" status="SUCCESS"
                                statusDescription="Command Successful"/>
</RESPONSE>
```

C++ Class Declaration

```
class CheckOutMedia : public Request
{
public:
     /// Primary Constructor
     /// This constructor is intended for primary instantiation of the object,
     /// given a single media ID.
     CheckOutMedia(const std::string& inMediaID);

     /// Secondary Constructor
     /// This constructor is for instantiation of the object from a list of
     /// media ID to eject.
     CheckOutMedia(const MediaList& inMediaList);

     /// Method to return the list of media
     const MediaList& getMediaList() const;

     /// Method to return the collection of status pairs
     const StatusPairList& getLocalStatus() const;

     /// Method to return the status pair for a specific mediaID
     const StatusPair& getLocalStatus(const std::string& mediaID) const;
}
```

CleanMedia

This API cleans media by removing inactive files from the specified media.

Input

mediaID: The ID of the media you want to clean. You can specify multiple media IDs to clean multiple media.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Remove inactive files from all specified media. -->
<COMMAND name="CleanMedia">
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <!-- mediaID : valid media ID -->
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <!-- One or more mediaID must be specified. -->
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <ARGUMENT name="mediaID" value="025311"/>
   <ARGUMENT name="mediaID" value="025312"/>
</COMMAND>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="CleanMedia" statusCode="0" status="SUCCESS"
                statusDescription="Command Successful">
    <!-- A status detail element will be returned for each media -->
     <STATUSDETAIL name="mediaID" value="025311"
                statusCode="0" status="SUCCESS"
                statusDescription="Command Successful"/>
     <STATUSDETAIL name="mediaID" value="025312"
                statusCode="0" status="SUCCESS"
```

```
                  statusDescription="Command Successful"/>
</RESPONSE>
```

C++ Class Declaration

```
class CleanMedia : public Request
{
public:
     /// Primary Constructor
     /// This constructor is intended for primary instantiation of the object.
     CleanMedia(const std::string& inMediaID);

     /// Secondary Constructor
     /// This constructor is intended for instantiation of the object from a
     /// vector of media ids.
     CleanMedia(const MediaList& inMedia);

     /// Method to return the list of media
     MediaList getMediaList() const;

     /// Method to return the list of local status pairs
     const StatusPairList& getLocalStatus() const;

     /// Method to return the specific local status pair for a given mediaID
     const StatusPair& getLocalStatus(const std::string& mediaID) const;
  }
```

CopyMedia

This API copies the content of all specified media to a piece of blank media.

Input

mediaID: The ID of the media you want to copy. You can specify multiple media IDs to copy multiple media.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Copy the content of all specified media to blank media. -->
<COMMAND name="CopyMedia">
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- mediaID : valid media ID -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- One or more mediaID must be specified. -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <ARGUMENT name="mediaID" value="025311"/>
    <ARGUMENT name="mediaID" value="025312"/>
</COMMAND>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="CopyMedia" statusCode="0" status="SUCCESS"
                            statusDescription="Command Successful">
    <!-- A status detail element will be returned for each media -->
     <STATUSDETAIL name="mediaID" value="025311"
                            statusCode="0" status="SUCCESS"
                            statusDescription="Command Successful"/>
     <STATUSDETAIL name="mediaID" value="025312"
                            statusCode="0" status="SUCCESS"
                            statusDescription="Command Successful"/>
</RESPONSE>
```

C++ Class Declaration

```
class CopyMedia : public Request
{
public:
    /// Primary Constructor
    /// This constructor is intended for primary instantiation of the object.
    CopyMedia(const std::string& inMediaID);

    /// Secondary Constructor
    /// This constructor is intended for instantiation of the object from a
    /// vector of media ids.
    CopyMedia(const MediaList& inMedia);

    /// Method to return the list of media
    MediaList getMediaList() const;

    /// Method to return the list of local status pairs
    const StatusPairList& getLocalStatus() const;

    /// Method to return the specific local status pair for a given mediaID
    const StatusPair& getLocalStatus(const std::string& mediaID) const;
}
```

EjectMedia

This API moves the specified media from the archive to a media I/E slot. EjectMedia does an automated media eject. This API must be used with the MoveMedia API.

After running this API, you must call the EnterMedia API to physically move the media from the mailbox into the destination library, or in the case of a vault, to logically enter the media into the vault.

Input

portID: The port used for ejecting media. The port ID is in the format 0,0,15,###. Only the digits following the final comma are needed, indicated as ### in the format example. It is not needed for vault archives.

mediaID: The ID of the media to eject.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Eject a specific media from the system. -->
<COMMAND name="EjectMedia">
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- portId : IE port id to eject media to -->
    <!-- mediaId : media to eject -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- Exactly one each of the above arguments are required. -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <ARGUMENT name="portId" value="16"/>
    <ARGUMENT name="mediaId" value="012345"/>
</COMMAND>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="EjectMedia" statusCode="0" status="SUCCESS"
                    statusDescription="Command Successful">
    <STATUSDETAIL name="mediaID" value="012345"
                    statusCode="0" status="SUCCESS"
                    statusDescription="Command Successful"/>
</RESPONSE>
```

C++ Class Declaration

```
class EjectMedia : public Request
{
public:
    /// Primary Constructor
    /// This constructor is intended for primary instantiation of the object,
    /// given a port Id and a single media Id.
    EjectMedia(const std::string& inMediaId,
            const std::string& inPortId = "0");

    /// Method to set port Id
    void setPortId(const std::string& inPortId);

    /// Method to return the media Id
    const std::string& getMediaId() const;

    /// Method to return the local status pair
    const StatusPair& getLocalStatus() const;
}
```

EnterMedia

This API adds (inserts) media into the specified archive from the I/E slot. This API must be used after you run the EjectMedia API, and must be used with the MoveMedia API.

Input

archiveID: The archive ID to which media is added.

portID: The port used for entering media. The port ID is in the format 0,0,15,###. Only the digits following the final comma are needed, indicated as ### in the format example. It is not needed for vault archives.

mediaID: The media ID to insert into the archive.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Enter a specific media into the system. -->
<COMMAND name="EnterMedia">
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <!-- archiveId : archive name to add media to -->
   <!-- portId : IE port id to move media from -->
   <!-- mediaId : media to add to archive -->
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <!-- Exactly one each of the above arguments are required. -->
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
```

```
    <ARGUMENT name="archiveId" value="archive2"/>
    <ARGUMENT name="portId" value="16"/>
    <ARGUMENT name="mediaId" value="012345"/>
</COMMAND>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="EnterMedia" statusCode="0" status="SUCCESS"
                        statusDescription="Command Successful">
     <STATUSDETAIL name="mediaID" value="012345"
                        statusCode="0" status="SUCCESS"
                        statusDescription="Command Successful"/>
</RESPONSE>
```

C++ Class Declaration

```
class EnterMedia : public Request
{
public:
     /// Primary Constructor
     /// This constructor is intended for primary instantiation of the object,
     /// given a archive Id, a port Id, and a single media Id.
     EnterMedia(const std::string& inArchiveId,
              const std::string& inMediaId,
              const std::string& inPortId = "0");

     /// Method to set portId
     void setPortId(const std::string& inPortId);

     /// Method to return the media Id
     const std::string& getMediaId() const;

     /// Method to return the local status pair
     const StatusPair& getLocalStatus() const;
}
```

FileRetrieve

This API retrieves the specified single file from secondary storage to primary storage.

Input

filename: The pathname of the file to be retrieved.

newFileName (optional): The name you want to give the retrieved file. The default value is an empty string, which means the file data is retrieved to disk using the original file name. This argument is optional if used by itself, but required if you use *startByte* and *endByte*.

startByte (optional): The starting byte for a partial retrieval. The default value is 0, which means a full retrieval is expected. Do not use this argument (in conjunction with the *endByte* argument) if you use the *copyId* argument.

endByte (optional): The end byte for a partial retrieval. The default value is 0, which means a full retrieval is expected. Do not use this argument (in conjunction with the *startByte* argument) if you use the *copyId* argument.

modAccessTime (optional): TRUE or FALSE. Indicates whether the retrieve will modify the file's access time. The default value is FALSE.

copyId (optional): Indicates which copy to be retrieved. The default value is 0, which refers to the primary copy. Do not use this argument if you use the *startByte* and *endByte* arguments.

Note: The *filename* argument is required. All other arguments are optional except as noted.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

newFileName: The name of the retrieved file.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Retrieve specific file data from storage media. -->
<COMMAND name="FileRetrieve">
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- fileName : name of file to retrieve -->
    <!-- newFileName : name of new file to retrieve to (default is "") -->
    <!-- startByte : start byte for partial retrieves (default is 0) -->
    <!-- endByte : end byte for partial retrieves (default is 0) -->
    <!-- modAccessTime : true, false (default is false) -->
    <!-- copyId : copy to retrieve (default is 0) -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- The fileName argument is required. All others are optional. -->
    <!-- All arguments may be specified only once. -->
    <!-- The copyId argument may not be specified with the startByte and -->
    <!-- endByte arguments. -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <ARGUMENT name="fileName" value="/csofs/storage/temp"/>
</COMMAND>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="FileRetrieve" statusCode="0" status="SUCCESS"
                    statusDescription="Command Successful">
     <RETRIEVEINFO name="filename" value="/csofs/storage/temp"
                    statusCode="0" status="SUCCESS"
                    statusDescription="Command Successful" />
</RESPONSE>
```

(The XML response is identical whether you use *startByte* and *endByte* or *copyId*.)

C++ Class Declaration

```
class FileRetrieve : public Request
{
public:
     /// First Constructor
     /// This constructor is intended for instantiation of the object for a
     /// full retrieve.
     FileRetrieve(const std::string& inFileName);

     /// Second Constructor
     /// This constructor is intended for instantiation of the object for a
     /// full retrieve of the specified filename into the new filename.
     FileRetrieve(const std::string& inFileName,
                    const std::string& inNewFileName);
```

```
/// Third Constructor
/// This constructor is intended for instantiation of the object for a
/// partial retrieve of the specified filename into the new filename
/// with startbyte and endbyte specifying the start/end of the retrieve.
FileRetrieve(const std::string& inFileName,
             const std::string& inNewFileName,
             const int64_t& inStartByte,
             const int64_t& inEndByte);

/// Fourth Constructor
/// This constructor is intended for instantiation of the object for a
/// full retrieve but with the copyId specified.
FileRetrieve(const std::string& inFileName,
             const int32_t& inCopyId);

/// Fifth Constructor
/// This constructor is intended for instantiation of the object for a
/// full retrieve of the specified filename into the new filename but
/// with the copyId specified.
FileRetrieve(const std::string& inFileName,
             const std::string& inNewFileName,
             const int32_t& inCopyId);

/// There are no set functions for copyid and startbyte/endbyte
/// because they are mutually exclusive options for this API.
/// These are handled by the various constructors defined above.

/// Set Function for filename
void setFileName(std::string& inFileName);

/// Set Function for modAccessTime
void setModAccessTime(bool& inModAccessTime);

/// Set Function for newfilename
void setNewFileName(std::string& inNewFileName);

/// Get Function for fileName
const std::string& getFileName() const;
```

```
    /// Get Function for modAccessTime
    const bool& getModAccessTime() const;

    /// Get Function for copyId
    const int32_t& getCopyId() const;

    /// Get Function for newFileName
    const std::string& getNewFileName() const;

    /// Get Function for startByte
    const int64_t& getStartByte() const;

    /// Get Function for endByte
    const int64_t& getEndByte() const;

    /// Method to return the local status pair
    const StatusPair& getLocalStatus() const;
}
```

GetArchiveCapacity

This API provides the amount of remaining storage capacity for all archives.

Input

NA. This API has no command arguments.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Returns the remaining storage capacity of all archives. -->
<COMMAND name="GetArchiveCapacity"/>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="GetArchiveCapacity" statusCode="0" status="SUCCESS"
                              statusDescription="Command Successful">
    <STRINGINFO statusCode="0" status="SUCCESS"
                        statusDescription="Command Successful">
        <!-- remainingMediaCapacity: (in GB) -->
        <INFO name="remainingMediaCapacity" value="99999"/>
    </STRINGINFO>
</RESPONSE>
```

C++ Class Declaration

```
class GetArchiveCapacity : public Request
{
public:
     /// Default and Primary Constructor
     GetArchiveCapacity();

     /// Method to return the total remaining capacity in GB.
     int32_t getCapacity() const;

     /// Method to return the local status
     const Status& getLocalStatus() const;
}
```

GetArchiveList

This API provides the current configuration settings for all archives.

Input

NA. This API has no command arguments.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

archiveName: The name of the archive whose configuration settings are listed.

state: The current state of the archive: ONLINE or OFFLINE.

archiveType: The disk type for the archive (e.g., SCSI).

serialNumber: The archive's serial number.

model: The archive's model.

numberSlots: The total number of slots in the archive.

numberSlotsUsed: The number of used slots in the archive.

firmwareVersion: The archive's firmware version.

totalSpace: The amount of total space in the archive, in gigabytes.

remainingSpace: The amount of space, in gigabytes, remaining in the archive.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Returns configuration settings for all archives. -->
<COMMAND name="GetArchiveList"/>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="GetArchiveList" statusCode="0" status="SUCCESS"
                    statusDescription="Command Successful">
   <ARCHIVEINFO statusCode="0" status="SUCCESS"
                 statusDescription="Command Successful">
      <!-- state: {online, offline} -->
      <!-- totalSpace: (in GB) -->
      <!-- remainingSpace: (in GB) -->
      <INFO name="archiveName" value="Andromeda"/>
      <INFO name="state" value="online"/>
      <INFO name="archiveType" value="SCSI"/>
      <INFO name="serialNumber" value="ADIC_1_9Y2230000"/>
      <INFO name="model" value="Scalar 24"/>
      <INFO name="numberSlots" value="24"/>
      <INFO name="numberSlotsUsed" value="18"/>
      <INFO name="firmwareVersion" value="237A"/>
      <INFO name="totalSpace" value="93504"/>
      <INFO name="remainingSpace" value="90000"/>
   </ARCHIVEINFO>
```

```
</RESPONSE>
```

C++ Class Declaration

```
class GetArchiveList : public Request
{
public:
    /// Default and Primary Constructor
    GetArchiveList();

    /// Method to return the list of Archives
    const std::vector<ArchiveInfo>& getArchiveInfo() const;

    /// Method to return the local status
    const Status& getLocalStatus() const;
}

class ArchiveInfo : virtual public Info
{
public:
    /// Default and Primary Constructor
    ArchiveInfo(const std::string& inName="unknown",
                const std::string& inState="unknown",
                const std::string& inArchiveType="unknown",
                const std::string& inSerialNumber="unknown",
                const std::string& inModel="unknown",
                const int32_t&     inNumberSlots=0,
                const int32_t&     inNumberSlotsUsed=0,
                const std::string& inFirmwareVersion="unknown",
                const int32_t&     inTotalSpace=0,
                const int32_t&     inRemainingCapacity=0);

    /// Method to return the name of the archive.
    const std::string& getName() const;

    /// Method to return the state of the archive.
    const std::string& getState() const;

    /// Method to return the type of the archive.
    const std::string& getArchiveType() const;

    /// Method to return the serialnumber of the archive.
```

```
const std::string& getSerialNumber() const;

/// Method to return the model of the archive.
const std::string& getModel() const;

/// Method to return the number of slots for media in the archive.
int32_t getNumberOfSlots() const;

/// Method to return the number of slots currently holding media in the
/// archive.
int32_t getNumberOfSlotsUsed() const;

/// Method to return the firmware version of the archive.
const std::string& getFirmwareVersion() const;

/// Method to return the total capacity of the archive in gigabytes.
int32_t getTotalSpace() const;

/// Method to return the total available (unused) capacity of the
/// archive in gigabytes.
int32_t getRemainingSpace() const;

/// Method to set the state of the archive.
void setState(const std::string& inState);

/// Method to set the type of the archive.
void setArchiveType(const std::string& inArchiveType);

/// Method to set the serialnumber of the archive.
void setSerialNumber(const std::string& inSerialNumber);

/// Method to set the model of the archive.
void setModel(const std::string& inModel);

/// Method to set the number of slots for media in the archive.
void setNumberOfSlots(const int32_t& inNumberSlots);

/// Method to set the number of slots currently holding media in
/// the archive.
void setNumberOfSlotsUsed(const int32_t& inNumberSlotsUsed);

/// Method to set the firmware version of the archive.
void setFirmwareVersion(const std::string& inFirmwareVersion);
```

```
    /// Method to set the total capacity of the archive in gigabytes.
    void setTotalSpace(const int32_t& inTotalSpace);

    /// Method to set the total available (unused) capacity of the
    /// archive in gigabytes.
    void setRemainingSpace(const int32_t& inRemainingSpace);

    /// Method to deflate this object for client-server transmision
    std::string deflate() const;

    /// Method to restore this object from a deflated string
    void inflate(std::string& inDeflatedObject);
}
```

Note: Currently, the GetArchiveList API assumes there is only one media type per archive. If the archive supports more than one media type, the output might not be accurate.

GetBackupStatus

This API provides the current status or progress of a backup in progress.

Input

NA. This API has no command arguments.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

info: Any relevant information about the backup operation.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Returns the progression status of currently running backup operation. -->
<COMMAND name="GetBackupStatus"/>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<RESPONSE name="GetBackupStatus" statusCode="0" status="SUCCESS"
                          statusDescription="Command Successful">
  <STRINGINFO statusCode="0" status="SUCCESS"
                   statusDescription="Command Successful">
    <!-- An INFO tag will surround each output line. -->
    <INFO value="Response text goes here."/>
    <INFO value="Response text goes here."/>
    <INFO value="Response text goes here."/>
    <INFO value="Response text goes here."/>
  </STRINGINFO>
</RESPONSE>
```

C++ Class Declaration

```
class GetBackupStatus : public Request
{
public:
    /// Default and Primary Contructor
    GetBackupStatus();

    /// Method to return the backup progression status
    const std::string& getBackupProgress() const;

    /// Method to return the local status
    const Status& getLocalStatus() const;
}
```

GetDriveList

This API provides a list of available drives and their attributes.

Input

NA. This API has no command arguments.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

driveName: The name of the drive whose attributes are listed.

state: The drive's current state: ONLINE or OFFLINE.

serialNumber: The drive's serial number.

type: The type of drive.

mountState: The drive's current mount state: MOUNTED or UNMOUNTED.

mediaID: A media ID of the drive.

firmwareVersion: The drive's firmware version.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Returns a list of drives and their attributes. -->
<COMMAND name="GetDriveList"/>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="GetDriveList" statusCode="0" status="SUCCESS"
                              statusDescription="Command Successful">
   <!-- The DriveInfo will be repeated to specify multiple drives -->
   <DRIVEINFO statusCode="0" status="SUCCESS"
                    statusDescription="Command Successful">
      <!-- state: {online, offline} -->
      <!-- mountState: {mounted, unmounted} -->
      <!-- mediaID: a media ID or empty string -->
      <INFO name="driveName" value="Andromeda_LTO_Drive1"/>
      <INFO name="state" value="online"/>
      <INFO name="serialNumber" value="1110236593"/>
      <INFO name="type" value="LTO"/>
      <INFO name="mountState" value="mounted"/>
      <INFO name="mediaID" value="04132"/>
      <INFO name="firmwareVersion" value="4AP0"/>
   </DRIVEINFO>
</RESPONSE>
```

C++ Class Declaration

```
class GetDriveList : public Request
{
public:
    /// Default Contructor
    GetDriveList();

    /// Method to return the drive information
    const std::vector<DriveInfo>& getDriveInfo() const;

    /// Method to return the local status
    const Status& getLocalStatus() const;
}

class DriveInfo : virtual public Info
{
public:
    /// Primary Constructor
    DriveInfo(const std::string& inName="",
              const std::string& inState="",
              const std::string& inSerialNumber="",
              const std::string& inType="",
              const std::string& inMountState="",
              const std::string& inMediaID="",
              const std::string& inFirmwareVersion="");

    /// Method to retrieve the name of the drive
    const std::string& getName() const;

    /// Method to retrieve the state of the drive
    const std::string& getState() const;

    /// Method to retrieve the serial number of the drive
    const std::string& getSerialNumber() const;

    /// Method to retrieve the type of the drive
    const std::string& getType() const;

    /// Method to retrieve the mount-state of the drive
    const std::string& getMountState() const;

    /// Method to retrieve the media ID the drive contains, if any.
    const std::string& getMediaID() const;
    /// Method to retrieve the firmware version currently loaded on the drive
    const std::string& getFirmwareVersion() const;

    /// Method to deflate this object for client-server transmision
    std::string deflate() const;

    /// Method to restore this object from a deflated string
    void inflate(std::string& inDeflatedObject);
}
```

GetFileAttribute

This API returns attribute information for the specified file, which is located in either primary storage or secondary storage (tape or storage disk). See the Output section for complete details on the information retrieved.

Input

filename: The file's pathname.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

filename: The file name.

location: DISK, TAPE, or DISK AND TAPE.

policyClass: The name of the policy class associated with the specified file.

numberExistingCopies: The number of existing copies. Valid values are 1 to [*numberTargetCopies*].

numberTargetCopies: The number of total copies.

mediaID: The media ID where the copy resides. There could be a list of media IDs.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Returns the attributes for a specific file. -->
<COMMAND name="GetFileAttribute">
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <!-- fileName : name of file -->
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <!-- The fileName argument is required exactly once. -->
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <ARGUMENT name="fileName" value="/snfs/myDirectory/myFile.dat"/>
</COMMAND>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="GetFileAttribute" statusCode="0" status="SUCCESS"
                        statusDescription="Command Successful">
   <FILEINFO statusCode="0" status="SUCCESS"
                  statusDescription="Command Successful">
      <!-- location: {DISK, TAPE, DISK AND TAPE} -->
      <INFO name="filename" value="/snfs/myDirectory/myFile.dat"/>
      <INFO name="location" value="TAPE"/>
      <INFO name="policyClass" value="policyclass1"/>
      <INFO name="numberExistingCopies" value="1"/>
      <INFO name="numberTargetCopies" value="1"/>
      <INFO name="mediaID" value="000001"/>
   </FILEINFO>
</RESPONSE>
```

C++ Class Declaration:

```
class GetFileAttribute : public Request
{
public:
    /// Primary Constructor
    /// This constructor is intended for primary instantiation of the object,
    /// providing it the filename
    GetFileAttribute(const std::string& filename);
    /// Copy constructor
    GetFileAttribute(const GetFileAttribute& obj);

    /// Destructor
    virtual ~GetFileAttribute();

    /// Assignment Operator
    GetFileAttribute& operator=(const GetFileAttribute& obj);

    /// Method to reset object with new filename
    void reset(const std::string& filename);

    /// Method to return the file information
    const FileInfo& getFileInfo() const;

    /// Method to return the local status pair
    const StatusPair& getLocalStatus() const;
}


class FileInfo : virtual public Info
{
public:
    /// Location values
    enum Location
    {
      unknown=0,
      disk,
      tape,
      diskandtape,
    };

    /// Default and Primary Constructor
    /// This can construct from a single mediaID string.
    FileInfo(const std::string& inName="",
            const Location&    inLocation=unknown,
            const std::string& inPolicyClass="",
            const int32_t&     inNumberOfExistingCopies=0,
            const int32_t&     inNumberOfTargetCopies=0,
            const std::string& inMediaID="");
    /// Secondary Constructor
    /// This constructs from a vector of mediaID.
    FileInfo(const std::string& inName,
             const Location&    inLocation,
             const std::string& inPolicyClass,
             const int32_t&     inNumberOfExistingCopies,
             const int32_t&     inNumberOfTargetCopies,
             const MediaList&   inMedia);

    /// Method that retrieves the file name.
    const std::string& getFileName() const;

    /// Method that retrieves the file location.
    const Location& getLocation() const;

    /// Method that retrieves the file location (as string).
    const std::string& getLocationAsString() const;

    /// Method that retrieves the file's policy class.
    const std::string& getPolicyClass() const;

    /// Method that retrieves the number of existing copies of the file.
    int32_t getNumberOfExistingCopies() const;

    /// Method that retrieves the number of target copies of the file.
    int32_t getNumberOfTargetCopies() const;

    /// Method that retrieves the collection of media IDs.
    const MediaList& getMedia() const;

    /// Method to deflate this object for client-server transmision
    std::string deflate() const;

    /// Method to restore this object from a deflated string
    void inflate(std::string& inDeflatedObject);
}
```

GetFileTapeLocationI

This API returns location information for the specified file, including the copy ID, segment number, starting block, and segment size. See the Output section for complete details about the information returned.

Input

filename: The file's pathname.

copyID: Indicates which copy from which to get segment information. The default value is 1, the primary copy.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

filename: The file name.

copyID: The copy ID.

segment: The segment number. (There can be multiple segments.)

mediaID: The media ID where this segment resides.

archiveID: The archive ID where the media belongs.

startBlock: The starting block.

offset: The offset from startBlock.

segmentSize: The segment size.

blockSize: The media block size.

Note: The file's data can be located by *startBlock* and *offset*. Then *segsize* will tell the amount of data to be read. *blksize* is required for doing a correct read operation; otherwise, enough data might not be read when reading a block. Both StorNext and AMASS tape formats are supported by this API.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Return the tape location info for a specific file. -->
<COMMAND name="GetFileTapeLocation">
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <!-- fileName : name of file -->
   <!-- copyID : copy number -->
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <!-- Exactly one each of the above arguments are required. -->
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <ARGUMENT name="fileName" value="/snfs/myDirectory/myFile.dat"/>
   <ARGUMENT name="copyID" value="1"/>
</COMMAND>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="GetFileTapeLocation" statusCode="0" status="SUCCESS"
                              statusDescription="Command Successful">
   <LOCATIONINFO statusCode="0" status="SUCCESS"
                              statusDescription="Command Successful">
        <INFO name="fileName" value="/stornnnnext/snfs1/d1/regfile" />
        <INFO name="copyID" value="1" />
        <SEGMENTINFO name="segmentNumber" value="1">
            <INFO name="mediaID" value="000455" />
            <INFO name="archiveID" value="scsi_archive1" />
            <INFO name="startBlock" value="7" />
            <INFO name="offset" value="128" />
            <INFO name="segmentSize" value="100000" />
            <INFO name="blockSize" value="524288" />
        </SEGMENTINFO>
        <SEGMENTINFO name="segmentNumber" value="2">
            <INFO name="mediaID" value="000455" />
            <INFO name="archiveID" value="scsi_archive1" />
            <INFO name="startBlock" value="7" />
            <INFO name="offset" value="100256" />
            <INFO name="segmentSize" value="100000" />
            <INFO name="blockSize" value="524288" />
        </SEGMENTINFO>
        <SEGMENTINFO name="segmentNumber" value="3">
            <INFO name="mediaID" value="000455" />
            <INFO name="archiveID" value="scsi_archive1" />
            <INFO name="startBlock" value="7" />
            <INFO name="offset" value="200384" />
            <INFO name="segmentSize" value="100000" />
            <INFO name="blockSize" value="524288" />
        </SEGMENTINFO>
    </LOCATIONINFO>
</RESPONSE>
```

C++ Class Declaration

```
class GetFileTapeLocation : public Request
{
public:
    /// Primary Constructor
    /// This constructor is intended for primary instantiation of the object,
    /// providing it the filename
    GetFileTapeLocation(const std::string& filename,
                        const int16_t&     copyid = 1);

    /// Method to set the copy id.
    void setCopyID(const int16_t inCopyID);

    /// Method to return the location info
    const LocationInfo& getLocationInfo() const;

    /// Method to return the local status pair
    const StatusPair& getLocalStatus() const;
}

class LocationInfo : virtual public Info
{
public:
    /// Primary Constructor
    LocationInfo(const std::string& inName="",
                 const int16_t&     inCopyID=1);

    /// Method that set the file name
    void setFileName(const std::string& filename);

    /// Method that set the copy id
    void setCopyID(const int16_t& copyid);
    /// Method that retrieves the file name.
    const std::string& getFileName() const;

    /// Method that retrieves the file location.
    const int16_t& getCopyID() const;

    /// Method that add a segment info to the segment list
    void addSegment(const SegmentInfo& seginfo);

    /// Method that retrieves the collection of segment locations
    const std::vector<SegmentInfo>& getSegmentList() const;

    /// Method to deflate this object for client-server transmision
    std::string deflate() const;

    /// Method to restore this object from a deflated string
    void inflate(std::string& inDeflatedObject);
}
```

GetMediaList

This API returns either a list of media IDs or the number of media that meets the specified criteria.

Input arguments within brackets {} must be entered explicitly as shown. For example, when entering the argument for the *format* argument, you must enter “count” or “list” (without the quotation marks).

All arguments are optional. If you do not specify any arguments, the total number of media is returned.

Input

format: {count or list}

location: {archive, vault, checkout, or unknown} If no argument is specified, archive is used as the default value.

archiveID: The archive ID.

classification: {data, backup, or cleaning}

availability: {available or unavailable}

writeAccess: {writeProtected or notWriteProtected} *integrity*: {suspect or notSuspect}

space: {full or blank}

percentUsed: The percentage of media space used, from 0.00 to 100.00.

copyID: The number of the copy used, from 1 - 8.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

location: archive, vault, checkout, or unknown.

archiveID: The archive ID.

classification: data, backup, or cleaning.

availability: available or unavailable.

writeAccess: writeProtected or notWriteProtected.

integrity: suspect or notSuspect.

space: full or blank.

count: The number of media that meet the specified criteria.

medialist: A list of media that meet the specified criteria.

Note: Output will include either count or list (not both), depending on what you specified as an input argument. Output will also repeat the selection criteria specified by the input.

XML Example

The following example illustrates how to return the number of blank media for a given archive.

Request Example 1:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Return a list of media based on specific query criteria. -->
<COMMAND name="GetMediaList">
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- format : count, list (default is count) -->
    <!-- location : archive, vault, checkout, unknown -->
    <!-- archiveID : archive name -->
    <!-- classification : data, backup, cleaning -->
    <!-- availability : available, unavailable -->
    <!-- writeAccess : writeProtected, notWriteProtected -->
    <!-- integrity : suspect, notSuspect -->
    <!-- space : blank, partial, full -->
    <!-- percentUsed : 0.00 ... 100.00 -->
    <!-- copyID : 1 ... 8 -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- All arguments are optional and may be specified only once. -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <ARGUMENT name="format" value="count"/>
    <ARGUMENT name="location" value="archive"/>
    <ARGUMENT name="availability" value="available"/>
</COMMAND>
```

Response Example 1:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="GetMediaList" statusCode="0" status="SUCCESS"
                              statusDescription="Command Successful">
    <MEDIALIST statusCode="0" status="SUCCESS"
                    statusDescription="Command Successful">
        <INFO name="location" value="archive"/>
        <INFO name="availability" value="available"/>
        <INFO name="count" value="6"/>
    </MEDIALIST>
</RESPONSE>
```

Request Example 2:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<COMMAND name="GetMediaList">
    <ARGUMENT name="format" value="list"/>
    <ARGUMENT name="location" value="archive"/>
    <ARGUMENT name="availability" value="available"/>
</COMMAND>
```

Response Example 2:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="GetMediaList" statusCode="0" status="SUCCESS"
                              statusDescription="Command Successful">
```

```
   <MEDIALIST statusCode="0" status="SUCCESS"
                statusDescription="Command Successful">
      <INFO name="location" value="archive"/>
      <INFO name="availability" value="available"/>
      <INFO name="mediaID" value="000001"/>
      <INFO name="mediaID" value="000002"/>
      <INFO name="mediaID" value="000003"/>
      <INFO name="mediaID" value="000004"/>
      <INFO name="mediaID" value="000005"/>
      <INFO name="mediaID" value="000006"/>
   </MEDIALIST>
</RESPONSE>
```

Request Example 3:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<COMMAND name="GetMediaList">
   <ARGUMENT name="archiveID" value="scsi_archive1"/>
   <ARGUMENT name="space" value="blank"/>
</COMMAND>
```

Response Example 3:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="GetMediaList" statusCode="0" status="SUCCESS"
                      statusDescription="Command Successful">
    <MEDIALIST statusCode="0" status="SUCCESS"
                 statusDescription="Command Successful">
       <INFO name="archiveID" value="scsi_archive1" />
       <INFO name="space" value="blank" />
       <INFO name="count" value="3" />
    </MEDIALIST>
</RESPONSE>
```

C++ Class Declaration

```
class GetMediaList : public Request
{
public:
    /// Output format values
    enum Format { count=0, list };

    /// Location values used as selection criteria
```

```
enum Location { archive=0, vault, checkout, unknown };

/// Classification values used as selection criteria
enum Classification { data=0, backup, cleaning };

/// Availability values used as selection criteria
enum Availability { available=0, unavailable };

/// Write access values used as selection criteria
enum WriteAccess { writeProtected=0, notWriteProtected };

/// Integrity values used as selection criteria
enum Integrity { suspect=0, notSuspect };

/// Space values used as selection criteria
enum Space { blank=0, partial, full };

/// Default Constructor (format defaults to count)
GetMediaList();

/// Primary Constructor
/// Note: This constructor will be deprecated in a future release.
GetMediaList(const Format&          inFormat,
             const Location&        inLocation,
             const std::string&     inArchiveID,
             const Classification&  inClassification,
             const Availability&    inAvailability,
             const WriteAccess&     inWriteAccess,
             const Integrity&       inIntegrity,
             const Space&           inSpace);

/// Set the output format for the media list.
void setFormat(const Format& inFormat);

/// Set the location for the media selection criteria.
void setLocation(const Location& inLocation);

/// Set the archive ID for the media selection criteria.
void setArchiveID(const std::string& inArchiveID);
```

```
/// Set the classification for the media selection criteria.
void setClassification(const Classification& inClassification);

/// Set the availability for the media selection criteria.
void setAvailability(const Availability& inAvailability);

/// Set the write access for the media selection criteria.
void setWriteAccess(const WriteAccess& inWriteAccess);

/// Set the integrity for the media selection criteria.
void setIntegrity(const Integrity& inIntegrity);

/// Set the space for the media selection criteria.
void setSpace(const Space& inSpace);

/// Set the percentUsed for the media selection criteria.
void setPercentUsed(const float inPercentUsed);

/// Set the copyID for the media selection criteria.
void setCopyID(const int32_t inCopyID);

/// Return the output format specified for the media list.
const Format& getFormat() const;

/// Return the media location selection criteria
const Location& getLocation() const;

/// Return the media archive ID selection criteria
const std::string& getArchiveID() const;

/// Return the media classification selection criteria
const Classification& getClassification() const;

/// Return the media availability selection criteria
const Availability& getAvailability() const;

/// Return the media write access selection criteria
const WriteAccess& getWriteAccess() const;

/// Return the media integrity selection criteria
```

```
const Integrity& getIntegrity() const;

/// Return the media space selection criteria
const Space& getSpace() const;

/// Return the media percentUsed selection criteria
const float getPercentUsed() const;

/// Return the media copyID selection criteria
const int32_t getCopyID() const;

/// Return the media list output format as a string.
const std::string& getFormatAsString() const;

/// Return the media location selection criteria as a string type.
const std::string& getLocationAsString() const;

/// Return the media archive id selection criteria as a string type.
const std::string& getArchiveIDAsString() const;

/// Return the media classification selection criteria as a string type.
const std::string& getClassificationAsString() const;

/// Return the media availability selection criteria as a string type.
const std::string& getAvailabilityAsString() const;

/// Return the media write access selection criteria as a string type.
const std::string& getWriteAccessAsString() const;

/// Return the media integrity selection criteria as a string type.
const std::string& getIntegrityAsString() const;

/// Return the media space selection criteria as a string type.
const std::string& getSpaceAsString() const;

/// Method to return the count of Media
int32_t getMediaCount() const;

/// Method to return the list of Media
const std::vector<std::string>& getMediaList() const;
```

```
    /// Method to return the local status
    const Status& getLocalStatus() const;
}
```

GetMediaStatus

This API returns status for the specified piece of media. See the Output section for complete details on the information retrieved.

Input

mediaID: The media ID. There could be a list of media IDs.

Output

location: The current state for the specified media ID: CHECKOUT, INTRANSIT, or UNKNOWN

writeProtected: Y or N.

status: AVAIL or UNAVAIL.

spaceUsed: The amount of used space in bytes.

spaceFree: The amount of free space in bytes.

classification: Media classification: DATA, BACKUP, or CLEANING.

mediaID: The media ID.

policyClass: The policy class associated with the media.

location: Archive name.

type: The media type.

writeProtected: Y or N.

mountCount: The number of mounts.

status: AVAIL or UNAVAIL.

suspectCount: The total number of times media was marked as suspect.

spaceUsed: The amount of used space in bytes.

spaceFree: The amount of free space in bytes.

percentUsed: The percentage of space used.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Returns status info on all specified media. -->
<COMMAND name="GetMediaStatus">
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- mediaID : valid media ID -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- One or more mediaID must be specified. -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <ARGUMENT name="mediaID" value="025311"/>
    <ARGUMENT name="mediaID" value="025312"/>
</COMMAND>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="GetMediaStatus" statusCode="0" status="SUCCESS"
                              statusDescription="Command Successful">
    <MEDIAINFO statusCode="0" status="SUCCESS"
                    statusDescription="Command Successful">
        <!-- location: {<archiveName>, checkout, intransit, unknown} -->
        <!-- writeProtected: {Y, N} -->
        <!-- status: {AVAIL, UNAVAIL} -->
        <!-- spaceUsed: (in bytes) -->
        <!-- spaceFree: (in bytes) -->
        <!-- classification: {data, backup, cleaning} -->
        <INFO name="mediaID" value="025311"/>
        <INFO name="policyClass" value="policyclass1"/>
        <INFO name="location" value="Andromeda"/>
        <INFO name="type" value="LTO"/>
        <INFO name="writeProtected" value="N"/>
        <INFO name="mountCount" value="13"/>
        <INFO name="status" value="AVAIL"/>
        <INFO name="suspectCount" value="0"/>
        <INFO name="spaceUsed" value="6,442,451,356"/>
        <INFO name="spaceFree" value="191,973,294,080"/>
        <INFO name="percentUsed" value="3.25"/>
        <INFO name="
    </MEDIAINFO>
</RESPONSE>
```

C++ Class Declaration:

```
class GetMediaStatus : public Request
{
public:
    /// Primary Constructor
    /// This constructor is intended for primary instantiation of the object,
    /// providing it the mediaId
    GetMediaStatus(const std::string& inMediaID);

    /// Constructor from a list of media
    /// This constructor is intended to create the request from a list of media
    GetMediaStatus(const MediaList& inMedia);

    /// Method to return a MediaInfo for a specific media
    MediaInfo getMediaInfo(const std::string& inMediaID);

    /// Method to return list of media info
    /// The media info list contains only the good media results.
    const std::vector<MediaInfo>& getMediaInfo() const;

    /// Method to return the list of status pairs
    const StatusPairList& getLocalStatus() const;

    /// Method to return a specific status pair for a given mediaID
    const StatusPair& getLocalStatus(const std::string& mediaID) const;
}

class MediaInfo : virtual public Info
{
public:
    /// Default and Primary Constructor
    MediaInfo(const std::string& inMediaID="unknown",
              const std::string& inPolicyClass="unknown",
              const std::string& inLocation="unknown",
              const std::string& inType="unknown",
              const std::string& inWriteProtected="unknown",
              const uint32_t     inMountCount=0,
              const std::string& inStatus="unknown",
              const uint32_t     inSuspectCount=0,
              const std::string& lastAccessTime="unknown",
              const uint64_t     inSpaceUsed=0,
              const uint64_t     inSpaceFree=0,
              const float        inPercentUsed=0,
              const std::string& inClassification="unknown");

    /// Method to return the media ID
    const std::string& getMediaID() const;

    /// Method to return the media policy class
    const std::string& getPolicyClass() const;

    /// Method to return the media location
    const std::string& getLocation() const;

    /// Method to return the media type
    const std::string& getType() const;

    /// Method to return flag indicating if the media is write-protected
    const std::string& getWriteProtected() const;

    /// Method to return the mount count for the media
    const uint32_t getMountCount() const;

    /// Method to return the media status
    const std::string& getStatus() const;

    /// Method to return the total number of times the media was marked
    /// suspect due to errors.
    const uint32_t getSuspectCount() const;

    /// Method to return the media last access time
    const std::string& getLastAccessTime() const;

    /// Method to return the amount of space used on the media
    const uint64_t getSpaceUsed() const;

    /// Method to return the amount of free space on the media
    const uint64_t getSpaceFree() const;

    /// Method to return the percentage of space used on the media
    const float   getPercentUsed() const;

    /// Method to return the classification of media (backup, data, clean, etc.)
    const std::string& getClassification() const;

    /// Method to deflate this object for client-server transmision
    std::string deflate() const;
```

```
    /// Method to restore this object from a deflated string
    void inflate(std::string& inDeflatedObject);
}
```

GetPolicy

This API retrieves the specified policy.

Input

policyClass: The desired policy class. Enter a valid policy class name or **all** to retrieve all policy classes.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

policyClass: The policy class name(s).

numberCopies: The number of copies currently maintained for the policy class.

maxInactiveVersions: The maximum number of versions currently maintained for the policy class.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Returns configuration info on one or all policy classes. -->
<COMMAND name="GetPolicy">
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <!-- policyClass : valid policy class name or "all" (default is all) -->
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <!-- The policyClass argument is optional and may be specified -->
   <!-- only once. -->
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <ARGUMENT name="policyClass" value="mypolicyclass"/>
</COMMAND>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="GetPolicy" statusCode="0" status="SUCCESS"
                                statusDescription="Command Successful">
    <POLICYINFO statusCode="0" status="SUCCESS"
                        statusDescription="Command Successful">
        <INFO name="policyClass" value="mypolicyclass"/>
        <INFO name="numberCopies" value="2"/>
        <INFO name="maxInactiveVersions" value="10"/>
    </POLICYINFO>
</RESPONSE>
```

C++ Class Declaration

```
class GetPolicy : public Request
{
public:
     /// Default Contructor
     GetPolicy(const std::string& inPolicyClass="all");

     /// Method to return the list of PolicyInfo objects
     const std::vector<PolicyInfo>& getPolicyInfo() const;

     /// Method to return a specific PolicyInfo object
     const PolicyInfo& getPolicyInfo(const std::string& inPolicyClass) const;

     /// Method to return the list of status pairs
     const StatusPairList& getLocalStatus() const;

     /// Method to return a specific status pair for a given policy class
     const StatusPair& getLocalStatus(const std::string& inPolicyClass) const;
}

class PolicyInfo : virtual public Info
{
public:
     /// Primary constructor
     PolicyInfo(const std::string& inClassName,
                  const int32_t&     inNumberOfCopies,
                  const int32_t&     inMaxInactiveVersions);

     /// Method to return the policy classname.
```

```
    const std::string& getPolicyClassName() const;

    /// Method to return the number of copies to make.
    const int32_t& getNumberOfCopies() const;

    /// Method to return the maximum inactive versions to keep.
    const int32_t& getMaxInactiveVersions() const;

    /// Method to deflate this object for client-server transmision
    std::string deflate() const;

    /// Method to restore this object from a deflated string
    void inflate(std::string& inDeflatedObject);
}
```

GetPortList

This API retrieves the import/export port IDs for a specified archive. An archive can have multiple ports. (Vaults do not have a port, so the ID is always 0.)

Input

archiveID: The ID of the archive whose import/export port information you want to retrieve.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

portID: The port number.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Return IE port information on a specific archive. -->
<COMMAND name="GetPortList">
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <!-- archiveID : valid archive name -->
```

```
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- The archiveID argument is required exactly once. -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <ARGUMENT name="archiveID" value="scsi_archive1"/>
</COMMAND>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="GetPortList" statusCode="0" status="SUCCESS"
                        statusDescription="Command Successful">
    <!-- The PORTINFO will be repeated to specify multiple ports -->
    <PORTINFO statusCode="0" status="SUCCESS"
                    statusDescription="Command Successful">
        <INFO name="portID" value="16"/>
    </PORTINFO>
</RESPONSE>
```

C++ Class Declaration

```
class GetPortList : public Request
{
public:
     /// Primary Constructor
     GetPortList(const std::string& inArchiveId);

     /// Method to return the port IDs
     const std::vector<PortInfo>& getPortInfo() const;

     /// Method to return the local status
     const Status& getLocalStatus() const;
}

class PortInfo : virtual public Info
{
public:
     /// Default and Primary Constructor
     PortInfo(const int16_t& inPortID=0);

     /// Method to return the port ID for the archive.
```

```
    int16_t getPortID() const;

    /// Method to set the port ID for the archive
    void setPortID(const int16_t& inPortID);

    /// Method to deflate this object for client-server transmision
    std::string deflate() const;

    /// Method to restore this object from a deflated string
    void inflate(std::string& inDeflatedObject);
}
```

GetSchedule

This API provides information about previously scheduled events such as backups.

Input

scheduleType: BACKUP

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

scheduleType: The type of schedule (e.g., BACKUP)

scheduleTime: Time the scheduled event is set to run, in 24-hour format: HH:MM.

scheduleLastAttempt: Time the scheduled event last attempted to run.

scheduleLastStatus: Status of the last scheduled event attempt: SUCCESSFULL, RUNNING, LOCKED, TERMINATED, or FAILED.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Get the time for the default full backup and default -->
<!-- partial backup schedules. -->
```

```
<COMMAND name="GetSchedule">
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <!-- scheduleType : backup -->
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <!-- Each argument above is required exactly once. -->
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <ARGUMENT name="scheduleType" value="backup"/>
</COMMAND>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="GetSchedule" statusCode="0" status="SUCCESS"
                              statusDescription="Command Successful">
    <SCHEDULEINFO statusCode="0" status="SUCCESS"
                            statusDescription="Command Successful">
       <!-- scheduleTime: (in 24-hour HH:MM format) -->
       <!-- scheduleLastStatus: {Successful, Running, Locked, Terminated, Failed} -->
       <INFO name="scheduleType" value="backup"/>
       <INFO name="scheduleTime" value="23:30"/>
       <INFO name="scheduleLastAttempt" value="Jul 9,2005 00:30"/>
       <INFO name="scheduleLastStatus" value="Successful"/>
    </SCHEDULEINFO>
</RESPONSE>
```

C++ Class Declaration

```
class GetSchedule : public Request
{
public:
     /// Default and Primary Constructor
     GetSchedule(const std::string& inScheduleType="backup");

     /// Method to return the ScheduleInfo
     const ScheduleInfo& getScheduleInfo() const;

     /// Method to return the local status.
     const Status& getLocalStatus() const;
}

class ScheduleInfo : virtual public Info
{
public:
     /// State Values
```

```
static const std::string unknown;

/// Default and Primary Constructor
ScheduleInfo(const std::string& inType=unknown,
             const std::string& inRunTime=unknown,
             const std::string& inLastAttempt=unknown,
             const std::string& inLastStatus=unknown,
             const std::string& inPeriod=unknown,
             const std::string& inDay=unknown);

/// Method to return the type of the schedule.
const std::string& getType() const;

/// Method to return the run time of the schedule.
const std::string& getRunTime() const;

/// Method to return the last attempted run time of the schedule.
const std::string& getLastAttempt() const;

/// Method to return the status of last attempted run of the schedule.
const std::string& getLastStatus() const;

/// Method to return the period of the schedule.
const std::string& getPeriod() const;

/// Method to return the day of the schedule.
const std::string& getDay() const;

/// Method to set the type of the schedule.
void setType(const std::string& inType);

/// Method to set the run time of the schedule.
void setRunTime(const std::string& inRunTime);

/// Method to set the last attempted run time of the schedule.
void setLastAttempt(const std::string& inLastAttempt);

/// Method to set the status of last attempted run of the schedule.
void setLastStatus(const std::string& inLastStatus);

/// Method to set period (daily/weekly/monthly) the of the schedule.
void setPeriod(const std::string& inPeriod);
```

```
    /// Method to set the day of the schedule.
    void setDay(const std::string& inWeekDay);

    /// Method to deflate this object for client-server transmision
    std::string deflate() const;

    /// Method to restore this object from a deflated string
    void inflate(std::string& inDeflatedObject);
}
```

GetSystemStatus

This API provides the current system status.

Input

NA. This API has no command arguments.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

version: The current StorNext software version number.

systemState: The overall system status.

tsm: The current TSM state (ONLINE or OFFLINE).

msm: The current MSM state (ONLINE or OFFLINE).

dsm: The current DSM state (ONLINE or OFFLINE).

database: The current database state (ONLINE or OFFLINE).

svclog: The current service log state (ONLINE or OFFLINE).

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Returns the current system status. -->
<COMMAND name="GetSystemStatus"/>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="GetSystemStatus" statusCode="0" status="SUCCESS"
                              statusDescription="Command Successful">
    <SYSTEMINFO statusCode="0" status="SUCCESS"
                         statusDescription="Command Successful">
        <INFO name="version" value="2.7.0(18)"/>
        <!-- The systemState is the overall system status -->
        <!-- The values below will be "online" or "offline" -->
        <INFO name="systemState" value="online"/>
        <INFO name="tsm" value="online"/>
        <INFO name="msm" value="online"/>
        <INFO name="dsm" value="online"/>
        <INFO name="database" value="online"/>
        <INFO name="svclog" value="online"/>
    </SYSTEMINFO>
</RESPONSE>
```

C++ Class Declaration

```
class GetSystemStatus : public Request
{
public:
     /// Default and Primary Constructor
     GetSystemStatus();

     /// Method to return the system information
     const SystemInfo& getSystemInfo() const;

     /// Method to return the local status
     const Status& getLocalStatus() const;
}

class SystemInfo : virtual public Info
{
public:
     /// Component Values
     enum Component
     {
        system=0,
        database,
        dsm,
        msm,
        svclog,
        tsm,
     };

     /// State Values
     static const std::string unknown;
     static const std::string online;
     static const std::string offline;

     /// Default and Primary Constructor
     SystemInfo(const std::string& inSystemVersion=unknown,
                const std::string& inSystemState=unknown,
                const std::string& inTsmState=unknown,
                const std::string& inMsmState=unknown,
                const std::string& inDsmState=unknown,
                const std::string& inDatabaseState=unknown,
                const std::string& inSvclogState=unknown);

     /// Method to return the version of the system
     const std::string& getSystemVersion() const;

     /// Method to return the state of the specified component
     const std::string& getState(const Component& inComponent) const;

     /// Method to set the system version.
     void setSystemVersion(const std::string& inSystemVersion);

     /// Method to set the state of the specified component.
     void setState(const Component& inComponent,
              const std::string& inState);

     /// Method to deflate this object for client-server transmision
     std::string deflate() const;

     /// Method to restore this object from a deflated string
     void inflate(std::string& inDeflatedObject);
}
```

MoveMedia

This API logically marks a media to move from one archive to another. It does not do the physical moving. In order to both physically and logically move a media between two archives, you must take three steps:

1 Call MoveMedia.

2. Call EjectMedia to physically move the media to the mailbox at the source archive.
3. Call EnterMedia to physically move the media from mailbox into the destination archive.

Note: If the media is entered or ejected from a vault, EjectMedia and EnterMedia are logical operations.

Input

mediaID: The media ID to move.

destArchiveID: The destination archive ID to move to.

Note: Both of these arguments are required.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Move a specific media to another archive. -->
<COMMAND name="MoveMedia">
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <!-- mediaID : media to move -->
   <!-- destArchiveID : destination archive name -->
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <!-- Exactly one each of the above arguments are required. -->
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <ARGUMENT name="mediaID" value="000455"/>
   <ARGUMENT name="destArchiveID" value="vault1"/>
</COMMAND>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="MoveMedia" statusCode="0" status="SUCCESS"
                    statusDescription="Command Successful">
    <STATUSDETAIL name="mediaID" value="000455"
                    statusCode="0" status="SUCCESS"
                    statusDescription="Command Successful"/>
</RESPONSE>
```

C++ Class Declaration

```
class MoveMedia : public Request
{
public:
    /// Primary Constructor
    /// This constructor is intended for primary instantiation of the object,
    /// given a destination archive ID and a single media ID.
    MoveMedia(const std::string& inMediaID,
                const std::string& inDestArchiveID);

    /// Method to return the local status pair
    const StatusPair& getLocalStatus() const;
}
```

PassThru

This API executes the specified command line argument. The PassThru API supports running only one CLI command at a time. You cannot string together multiple concatenated commands (e.g. ls /tmp/; ls /var/tmp/; cd /tmp/).

Input

commandString: The command you want to execute from the command line (e.g., ls -laF /tmp /)

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

Any response for the executed command.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Executes a specific shell command. -->
<COMMAND name="PassThru">
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- commandString : valid shell command -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- The commandString argument is required exactly once. -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <ARGUMENT name="commandString" value="ls -laF /tmp"/>
</COMMAND>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="PassThru" statusCode="0" status="SUCCESS"
                                statusDescription="Command Successful">
    <STRINGINFO statusCode="0" status="SUCCESS"
                        statusDescription="Command Successful">
        <!-- An INFO tag will surround each output line. -->
        <INFO value="Response text goes here."/>
        <INFO value="Response text goes here."/>
        <INFO value="Response text goes here."/>
        <INFO value="Response text goes here."/>
    </STRINGINFO>
</RESPONSE>
```

C++ Class Declaration

```
class PassThru : public Request
{
```

```
public:
    /// Primary Constructor
    /// This constructor is intended for primary instantiation of the object,
    /// given a command string.
    PassThru(const std::string& inCommand);

    /// Method to return the passthru command string
    const std::string& getCommandString() const;

    /// Method to return the results of the passthru command
    const std::string& getCommandResults() const;

    /// Method to return the local status pair
    const StatusPair& getLocalStatus() const;
}
```

RmDiskCopy

This API removes the disk copy of the specified file through explicit truncation.

Input

filename: The pathname of the file whose on-disk copy will be removed.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Truncate file data from disk. -->
<COMMAND name="RmDiskCopy">
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- fileName : name of file -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- The fileName argument is required exactly once. -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <ARGUMENT name="fileName" value="/snfs/myDirectory/myFile.dat"/>
</COMMAND>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="RmDiskCopy" statusCode="0" status="SUCCESS"
                     statusDescription="Command Successful">
    <STATUSDETAIL name="filename" value="/snfs/myDirectory/myFile.dat"
                     statusCode="0" status="SUCCESS"
                     statusDescription="Command Successful"/>
</RESPONSE>
```

C++ Class Declaration

```
class RmDiskCopy : public Request
{
public:
    /// Primary Constructor
    /// This constructor is intended for primary instantiation of the object,
    /// given a single filename.
    RmDiskCopy(const std::string& filename);

    /// Method to return the local status pair
    const StatusPair& getLocalStatus() const;
}
```

SetArchiveState

This API allows you to set the archive state to ON or OFF.

Input

archiveName: The name of the archive for which you want to set the state.

state: ON or OFF.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Set the operating state of a specific archive. -->
<COMMAND name="SetArchiveState">
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <!-- archiveName : archive name -->
   <!-- state : ON, OFF -->
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <!-- Each argument above is required exactly once. -->
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <ARGUMENT name="archiveName" value="Andromeda"/>
   <ARGUMENT name="state" value="ON"/>
</COMMAND>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="SetArchiveState" statusCode="0" status="SUCCESS"
                         statusDescription="Command Successful">
    <STATUSDETAIL name="archiveName" value="Andromeda"
                         statusCode="0" status="SUCCESS"
                         statusDescription="Command Successful"/>
</RESPONSE>
```

C++ Class Declaration

```
class SetArchiveState : public Request
{
public:
    /// State values
    enum StateType
    {
      ONLINE=0,
      OFFLINE
    };

    /// Primary Constructor
    /// This constructor is intended for primary instantiation of the
    /// object, providing it the archive name and state.
    SetArchiveState(const std::string& inArchiveName,
                    const StateType&   inArchiveState);

    /// Method to return the archive name
    const std::string& getArchiveName() const;

    /// Method to return the archive state
    const StateType& getArchiveState() const;

    /// Method to return the archive state as a string
    const std::string& getArchiveStateAsString() const;

    /// method to return the local status
    const Status& getLocalStatus() const;
}
```

SetDirAttributes

This API allows you to set the following directory attributes:

- store (enable or disable)
- truncate (enable or disable)
- policy class name for the directory

Input

directoryName: The name of the directory for which you want to set attributes.

noTruncate: TRUE or FALSE.

noStore: TRUE or FALSE.

policyClass: The name of the policy class you want to apply to the directory.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Set the attributes for a specific directory. -->
<COMMAND name="SetDirAttributes">
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- directoryName : name of directory -->
    <!-- noTruncate : TRUE, FALSE -->
    <!-- noStore : TRUE, FALSE -->
    <!-- policyClass : valid policy class name -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- The directoryName argument is required exactly once. -->
    <!-- At least one of the following arguments are required, but -->
    <!-- no more than one of each: -->
    <!-- noTruncate -->
    <!-- noStore -->
    <!-- policyClass -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <ARGUMENT name="directoryName" value="/snfs/mydirectory"/>
    <ARGUMENT name="noTruncate" value="TRUE"/>
    <ARGUMENT name="noStore" value="TRUE"/>
    <ARGUMENT name="policyClass" value="mypolicyclass"/>
</COMMAND>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="SetDirAttributes" statusCode="0" status="SUCCESS"
                    statusDescription="Command Successful">
    <STATUSDETAIL name="directoryName" value="/snfs/mydirectory"
                    statusCode="0" status="SUCCESS"
                    statusDescription="Command Successful"/>
</RESPONSE>
```

C++ Class Declaration

```
class SetDirAttributes : public Request
{
public:
    /// Primary Constructor
    /// This constructor is intended for primary instantiation of the object,
    /// providing it the directory name, policy class, truncate, and store
    /// settings.
    /// the state
    SetDirAttributes(const std::string& inDirectoryName,
                     const std::string& inPolicyClass,
                     const bool& inNoTruncate,
                     const bool& inNoStore);

    /// Secondary Constructor
    /// This constructor is a secondary means to instantiate the object,
    /// the state
    SetDirAttributes(const std::string& inDirectoryName);

    /// Method to return the directory name
    const std::string& getDirectoryName() const;

    /// Method to return the policy class
    const std::string& getPolicyClass() const;

    /// Method to return the noTruncate flag
    bool getNoTruncateFlag() const;

    /// Method to return the noStore flag
    bool getNoStoreFlag() const;

    /// Method to return the local status
    const StatusPair& getLocalStatus() const;
}
```

SetDriveState

This API allows you to set the drive state to ON or OFF.

Input

drivename: The name of the drive whose state you want to set.

state: ON or OFF.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Set the operating state of a specific drive. -->
<COMMAND name="SetDriveState">
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <!-- drivename : valid drive name -->
   <!-- state : ON, OFF -->
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <!-- Exactly one each of the above arguments is required. -->
   <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
   <ARGUMENT name="drivename" value="lto2-001"/>
   <ARGUMENT name="state" value="ON"/>
</COMMAND>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="SetDriveState" statusCode="0" status="SUCCESS"
                     statusDescription="Command Successful">
   <STATUSDETAIL name="drivename" value="Andromeda_LTO_Drive1"
                     statusCode="0" status="SUCCESS"
                     statusDescription="Command Successful"/>
</RESPONSE>
```

C++ Class Declaration

```
class SetDriveState : public Request
{
public:
    /// Drive state values.
    enum DriveState
    {
      ON = 0,
      OFF
    };

    /// Primary Constructor
    /// This constructor is intended for primary instantiation
    /// of the object, providing it the drive name and state.
    SetDriveState(const std::string& inDriveName,
                  const DriveState&  inDriveState);

    /// Method to return the drivename
    std::string getDrivename() const;

    /// Method to return the drive state
    DriveState getDriveState() const;

    /// Method to return the drive state as a string
    std::string getDriveStateAsString() const;

    /// Method to return the local status pair
    const StatusPair& getLocalStatus() const;
}
```

SetFileAttributes

This API allows you to set the following attributes for a file:

- store (enable or disable)
- truncate (enable or disable)

Input

fileName: The name of the file for which you want to set attributes.

noStore: TRUE or FALSE.

noTruncate: TRUE or FALSE.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Set the attributes for a specific file. -->
<COMMAND name="SetFileAttributes">
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- fileName : name of file -->
    <!-- noTruncate : TRUE, FALSE -->
    <!-- noStore : TRUE, FALSE -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- The fileName argument is required exactly once. -->
    <!-- At least one of the following arguments are required, but -->
    <!-- no more than one of each: -->
    <!-- noTruncate -->
    <!-- noStore -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <ARGUMENT name="fileName" value="/snfs/myDirectory/myFile.dat"/>
    <ARGUMENT name="noTruncate" value="TRUE"/>
    <ARGUMENT name="noStore" value="TRUE"/>
</COMMAND>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="SetFileAttributes" statusCode="0" status="SUCCESS"
                            statusDescription="Command Successful">
     <STATUSDETAIL name="filename" value="/snfs/myDirectory/myFile.dat"
                            statusCode="0" status="SUCCESS"
                            statusDescription="Command Successful"/>
</RESPONSE>
```

C++ Class Declaration

```
class SetFileAttributes : public Request
{
public:
     /// Primary Constructor
     /// This constructor is intended for primary instantiation of the object,
     /// providing it the file name, notruncate, and nostore attributes to be
     /// set for the file.
     SetFileAttributes(const std::string& inFileName,
                       const bool&        inNoTruncate,
                       const bool&      inNoStore);

     /// Secondary Constructor
     /// This constructor is a secondary means to instantiate the object,
     SetFileAttributes(const std::string& inFileName);

     /// Method to return the fileName
     const std::string& getFileName() const;

     /// Method to return the noTruncate flag
     bool getNoTruncateFlag() const;

     /// Method to return the noStore flag
     bool getNoStoreFlag() const;

     /// Method to return the local status
     const StatusPair& getLocalStatus() const;
}
```

SetMediaState

This API allows you to set the state for one or more piece of media.

Input

state: AVAIL (available), UNAVAIL (not available), PROTECT (write protected), UNPROTECT (not write protected), UNMARK (not marked), or UNSUSP (not suspect).

mediaID: The ID of the media whose state you want to set. When entering multiple media IDs, enter one media ID per line.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code for each media ID entered.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
```

```
<!-- Set the operating state for specific media. -->
<COMMAND name="SetMediaState">
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- state : avail, unavail, protect, unprotect, unmark, unsusp -->
    <!-- (the state will be set on all specified media) -->
    <!-- mediaID : valid media ID -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- The state argument is required exactly once. -->
    <!-- One or more mediaID arguments are required. -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <ARGUMENT name="state" value="avail"/>
    <ARGUMENT name="mediaID" value="025311"/>
    <ARGUMENT name="mediaID" value="025312"/>
</COMMAND>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="SetMediaState" statusCode="0" status="SUCCESS"
                        statusDescription="Command Successful">
    <!-- A status detail element will be returned for each media -->
     <STATUSDETAIL name="mediaID" value="025311"
                        statusCode="0" status="SUCCESS"
                        statusDescription="Command Successful"/>
     <STATUSDETAIL name="mediaID" value="025312"
                        statusCode="0" status="SUCCESS"
                        statusDescription="Command Successful"/>
</RESPONSE>
```

C++ Class Declaration

```
class SetMediaState : public Request
{
public:
    /// Media State values
    enum MediaState
    {
        MEDIA_STATE_TYPE_START=0,
        AVAIL=MEDIA_STATE_TYPE_START,
        UNAVAIL,
        PROTECT,
        UNPROTECT,
        UNMARK,
        UNSUSP,
        MEDIA_STATE_TYPE_END
    };

    /// Primary Constructor
    /// This constructor is intended for primary instantiation of the object,
    /// providing it the media name and state.
    SetMediaState(const std::string& inMediaID,
                  const MediaState& inState);

    /// Secondary Constructor
    /// This constructor is intended for instantiation of the object from a
    /// vector of media ids
    SetMediaState(const MediaList& inMedia,
                  const MediaState& inState);

    /// Method to retrieve the map of mediaIDs to their status'
    std::map<std::string, Status> getMediaMap() const;

    /// Method to return the media state
    const MediaState& getMediaState() const;

    /// Method to return the media state as a string
    const std::string& getMediaStateAsString() const;

    /// Method to return the list of media
    MediaList getMediaList() const;

    /// Method to return the list of local status pairs
    const StatusPairList& getLocalStatus() const;

    /// Method to return the specific local status pair for a given mediaID
    const StatusPair& getLocalStatus(const std::string& mediaID) const;
}
```

SetPolicy

This API allows you to set the number of copies and the maximum number of inactive versions for a policy class.

Input

policyClass: The name of the policy class for which you want to set arguments, or enter "**all**" to apply arguments to all policy classes.

numberOfCopies: The number of copies to maintain for the policy class.

maxInactiveVersions: The maximum of versions to maintain for the policy class.

> **Note:** You must enter one policy class name. You can enter either the number of copies, the maximum number of versions, or both arguments.

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Configure one or all policy classes. -->
<COMMAND name="SetPolicy">
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- policyClass : valid policy class name or "all" -->
    <!-- numberOfCopies : 1 .. 8 -->
    <!-- maxInactiveVersions : 1 .. 10 -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- The policyClass argument is required exactly once. -->
    <!-- At least one of the following arguments are required, but -->
    <!-- no more than one of each: -->
    <!-- numberOfCopies -->
    <!-- maxInactiveVersions -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <ARGUMENT name="policyClass" value="mypolicyclass"/>
    <ARGUMENT name="numberOfCopies" value="2"/>
    <ARGUMENT name="maxInactiveVersions" value="10"/>
```

```
</COMMAND>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="SetPolicy" statusCode="0" status="SUCCESS"
                              statusDescription="Command Successful">
     <STATUSDETAIL name="policyClass" value="all"
                              statusCode="0" status="SUCCESS"
                              statusDescription="Command Successful"/>
</RESPONSE>
```

C++ Class Declaration

```
class SetPolicy : public Request
{
public:
    /// Primary Constructor
    /// This constructor is intended for primary instantiation of the object.
    SetPolicy(const std::string& inPolicyClass,
              const uint32_t    inNumberOfCopies,
              const uint32_t    inMaxInactiveVersions);

    /// Secondary Constructor
    /// This constructor is a secondary means to instantiate the object.
    SetPolicy(const std::string& inPolicyClass);

    /// Get the policy class name
    const std::string& getPolicyClass() const;

    /// Get the number of copies
    uint32_t getNumberOfCopies() const;

    /// Get the maximum inactive versions
    uint32_t getMaxInactiveVersions() const;

    /// Return the list of local status pairs.
    const StatusPairList& getLocalStatus() const;

    /// Return a specific local status pair for a given policy class
    const StatusPair& getLocalStatus(const std::string& inPolicyClass) const;
}
```

SetSchedule

This API allows you to specify a schedule for a backup. Running this API affects only system default full backups and default partial backups, not any user-configured backup events.

Input

scheduleType: BACKUP or other scheduled event type.

timeOfDay: The time the scheduled event begins, in 24-hour format (HH:MM).

Output

status: SUCCESS, FAILURE, SUBFAILURE, or SYNTAXERROR status code.

XML Example

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- Set the time for the default full backup and default -->
<!-- partial backup schedules. -->
<COMMAND name="SetSchedule">
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- scheduleType : backup -->
    <!-- timeOfDay : time in 24-hour HH:MM format -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- Each argument above is required exactly once. -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <ARGUMENT name="scheduleType" value="backup"/>
    <ARGUMENT name="timeOfDay" value="23:59"/>
</COMMAND>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RESPONSE name="SetSchedule" statusCode="0" status="SUCCESS"
                          statusDescription="Command Successful">
    <STATUSDETAIL name="scheduleType" value="backup"
                          statusCode="0" status="SUCCESS"
```

```
                    statusDescription="Command Successful"/>
</RESPONSE>
```

C++ Class Declaration

```
class SetSchedule : public Request
{
public:
    /// Enumerator for the schedule types
    enum ScheduleType {backup=0};

    /// Primary Constructor
    ///                 schedule type.
    SetSchedule(const ScheduleType& inScheduleType,
                const std::string&  inTimeOfDay);

    /// Return the string associated to a ScheduleType
    static std::string getTypeString(const ScheduleType& inType);

    /// Return the schedule type
    const ScheduleType& getScheduleType() const;

    /// Return the scheduled time-of-day
    const std::string& getScheduleTimeOfDay() const;

    /// Return the local status
    const Status& getLocalStatus() const;
}
```

Chapter 3

StorNext File System APIs

This chapter describes the Application Programming Interfaces (APIs) that are available for StorNext File System. These file system APIs are automatically installed with the StorNext software and do not need to be installed separately.

The file system APIs are grouped into the following categories:

- Allocation and Space Management APIs
- Quality of Service and Real Time I/O APIs
- File System Configuration and Location Management APIs
- Access Management APIs

Appendix A at the end of this guide provides a test API sample program that illustrates how to use many of the StorNext File System APIs described in this guide.

Most of these APIs take two parameters: a request structure and a reply structure. On Windows, these correspond to the *inbuffer* and *outbuffer* parameters of DeviceIoControl(). On UNIX, the two structures (request and response) are consolidated into a single union so that we can still use the ioctl() and fcntl() interface. The functionality remains the same.

For example, the UNIX definition of the call to allocate space would be:

```
typedef union allocspacereqreply {
    AllocSpaceReq_treq;
    AllocSpaceReply_treply;
} AllocSpaceReqReply_t;
```

Field names for request structures are prefaced with xq_ where <x> is representative of the structure's name. Field names for reply structures are similarly prefaced with xr_.

All structures are 64-bit aligned and use the following typedefs on Windows:

```
typedef UCHARuint8_t;
typedef USHORTuint16_t;
typedef ULONGuint32_t;
typedef ULONGLONGuint64_t;
typedef CHARint8_t;
typedef SHORTint16_t;
typedef LONGint32_t;
typedef LONGLONGint64_t;
```

The control code definitions are platform dependent, and are defined in a separate file for each platform. The version number is encoded in each control code so that individual calls can be modified without affecting the rest of the interface. The macro name (for example, CvApi_PunchHole,) is the same on all platforms.

Platform	Filename	Interface Routine
Linux	cv_linux_ioctl.h	ioctl(2)
Solaris	cv_sol_ioctl.h	ioctl(2)
Irix	cv_irix_ioctl.h	fcntl(2)
Windows	cv_nt_ioctl.h	DeviceIoControl()

Except where noted, all calls return 0 (zero) on success, and a standard platform-specific error code on error. On UNIX, it is a standard errno. On Windows, it is one of the status codes listed in ddk\inc Windows Driver Development Kit. Internally, StorNext maps all error codes to a platform-independent value and only maps to platform error codes just before returning to the user. In this document the following error codes are defined:

VOP_ENOENT	2
VOP_EACCESS	5
VOP_EXIST	7
VOP_EINVAL	11

VOP_ENOSPC	12
VOP_EFAULT	19

These error codes are mapped to their closest platform-specific error. For example, the error VOP_ENOENT maps to ENOENT on most UNIX platforms, and the Windows error code STATUS_OBJECT_NAME_NOT_FOUND on Windows (which may map to a Win32 definition such as ERROR_NOT_FOUND).

All offsets and sizes are given in bytes and are rounded up, if necessary, to the next file system block size.

For calls that return variable length buffers (such as extent lists), the call will return the total number of items available, as well as the number returned in this particular call. If more data is available than can be returned in the user’s buffer, as much as possible is copied into the buffer, and no error is returned. For subsequent calls, the user can specify a different starting location (ordinal for stripe groups, starting offset for extents). This is similar to the getdents/getdirentries semantic on UNIX. Note that if the list is changing while the user is attempting to retrieve it, inconsistent results may be returned. When there are no more entries available, ENOENT is returned.

In this document, the word *handle* is synonymous with a *file descriptor* in UNIX.

Note: StorNext file system API names are preceded with “CvApi” because they were inherited from CVFS.

Allocation and Space Management APIs

These APIs allow you to control how data is written to StorNext, resulting in faster writes and more efficient allocation of capacity.

CvApi_AllocSpace

This API allocates extent space in a file. (This API is scheduled for deprecation and will not be supported in future StorNext releases.)

Handle

Target file.

Notes

This call attempts to allocate disk space of the requested size, starting at the requested file-relative offset. Commonly, this results in a single extent being allocated. However, if the file system free space is fragmented, up to 24 extents may be allocated. In addition, if the entire requested space cannot be allocated by adding 24 new extents, the API performs only a partial allocation and still returns a successful status. Therefore, to reliably determine the actual amount of space allocated by CvApi_AllocSpace, applications must track the number of file blocks using the UNIX fstat(2) system call or through the CvApi_CvFstat API.

If the caller specifies an offset to begin allocation, the call allocates space at the offset, rounded up to a file system block size. If any allocation exists that maps even a portion of <offset + size>, the call returns EXISTS. If no offset is specified, the call allocates the requested size beginning at the next file system block boundary beyond the current end of file. The number of bytes is rounded up to a file system block size. In both cases, the file size is updated if the allocation causes an increase in the current end of file.

If the affinity is specified, this sets the affinity for this and all future allocations. In this way, setting the affinity is “sticky.” If the affinity is already set in the file, setting the affinity in this call has no affect. To get/ set the affinity, see the CvApi_GetAffinity / CvApi_SetAffinity calls. The allocation will be made *exclusively* only from stripe groups that have the matching affinity.

The caller can also specify that the extent information be loaded into the client extent mapping tables. This eliminates a subsequent trip to the FSM to retrieve extent information for the range mapped by this call.

All byte sizes and offsets are rounded up to the nearest file system block size. The call returns the actual allocated size and offset.

CvApi_AllocSpace is scheduled for deprecation in a future release. You should use the new function CvApi_VerifyAlloc instead.

Structure

```
typedef struct _AllocSpaceReq {
    uint64_taq_size;
    uint64_taq_offset;
    uint64_taq_affinitykey;

    uint32_taq_flags;
        #define ALLOC_OFFSETzx01
        #define ALLOC_LOAD_EXT0x02
        #define ALLOC_STRIPE_ALIGN0x04
        #define ALLOC_AFFINITY0x08
        #define ALLOC_KEEPSIZE0x10
        #define ALLOC_PERFECTFIT0x20
        #define ALLOC_SETSIZE0x40


    uint32_taq_pad1;

} AllocSpaceReq_t;

typedef struct _AllocSpaceReply {
    uint64_tar_size;
    uint64_tar_offset;
} AllocSpaceReply_t;
```

UNIX `ioctl` structure:

```
typedef union _allocspacereqreply {
    AllocSpaceReq_treq;
    AllocSpaceReply_treply
} AllocSpaceReqReply_t;
```

reQuest Fields

aq_size	Size in bytes to allocate. Must not be zero. If not a multiple of the file system block size, it is rounded up to the nearest file system block size.
aq_offset	If ALLOC_OFFSET is set, the aq_size bytes (rounded up to the nearest file system block size) is allocated starting at file byte offset aq_offset (rounded up to the nearest file system block size). If ALLOC_OFFSET is clear, the offset is ignored when allocating space. The space is allocated at the end of the file.
aq_affinitykey	64 bit affinity “key” identifying the affinity for the allocation. Valid only if ALLOC_AFFINITY is set. This forces the space to be allocated exclusively from stripe groups with a matching affinity key.
aq_flags	Control Flags:

ALLOC_OFFSET	aq_offset is valid.
ALLOC_LOAD_EXT	Load the extent struct mapping on the client and add it to the client file system. Note: in case where multiple extents are allocated, only the first extent is loaded.
ALLOC_STRIPE_ALIGN	Allocate space starting at a stripe boundary.
ALLOC_AFFINITY	aq_affinitykey is valid.
ALLOC_KEEPSIZE	Do not update the size of the file, even if extending it.
ALLOC_PERFECTFIT	Follow the ‘perfect fit’ rules for allocation.
ALLOC_SETSIZE	Update the file size, and broadcast the size change to other clients.

The default is to only broadcast the new number of blocks and the fact the extent list has changed (which causes clients to flush their extent lists).

Reply Fields

ar_size	Actual size of the allocation that was attempted.
ar_offset	File relative offset of allocated space.

Error Returns

VOP_ENOSPC	Insufficient space in the file system to satisfy the request.
VOP_EINVAL	Invalid affinity key.
VOP_EXISTS	An allocation already exists that maps some or all of the specified offset and length.

CvApi_GetPerfectFitStatu

This API determines whether a file has been marked for PerfectFit allocations.

Handle

Handle for the file being queried.

Notes

This API is used by the `snfsdefrag` application to ensure that files with PerfectFit allocations continue to have PerfectFit allocations after the files are defragmented.

Structure

typedef struct _GetPerfectFitStatusReply {

```
        uint32_t    pr_status;
        uint32_t    pr_pad1;
} GetPerfectFitStatusReply_t;
```

UNIX ioctl structure:

None. Use GetPerfectFitStatusReply_t directly.

Reply Fields:

pr_status	If 1, the file has the PerfectFit bit set. If 0, the file does not.

Error Returns

VOP_ENOENT	The file doesn't exist.
VOP_EINVAL	The file is not a “regular” file.
Other	Communications failure with the FSM.

CvApi_PunchHole

This API punches a hole in the file.

Handle

Target file.

Notes

This call punches a hole in the file, adjusting the allocation map of the file to indicate that no data blocks are allocated for the indicated range. The granularity of access is the file system block size. The byte offsets are rounded down to the beginning of the block containing the specified byte offset. The actual offsets used are returned; the starting offset will always be file system block aligned.

If zero is specified as the ending offset, a hole is punched to the end of the file.

Structure

```
typedef struct _PunchHoleReq {
      uint64_tpq_start;
      uint64_tpq_end;     /* Inclusive */
} PunchHoleReq_t;

typedef struct _PunchHoleReply {
      uint64_tpr_start;
      uint64_tpr_end;
      uint64_tpr_nblocks;
      uint64_tpr_blksfreed;
} PunchHoleReply_t;
```

UNIX `ioctl` structure:

```
typedef union _punchholereqrep {
      PunchHoleReq_treq;
      PunchHoleReply_treply;
} PunchHoleReqReply_t;
```

reQuest fields

pq_start	Starting byte offset of hole.
pq_end	Inclusive ending byte offset. A value of zero means punch to EOF.

Reply Fields

pr_start	Starting byte offset where hole was created.
pr_end	Inclusive ending byte offset where hole was created.
pr_nblocks	Number of blocks currently allocated to the file.
pr_blksfreed	Number of blocks freed.

CvApi_SetFileSize

This API sets the size of a file without zeroing pre-allocated data.

Handle

Target file.

Notes

The effect of this call is very similar to making the ftruncate(2) system call, except that when the file size is being extended, any existing blocks between the old EOF and the new EOF are not zeroed regardless of whether the SNFS "sparse" mount option is enabled or not. CvApi_SetFileSize is not currently supported on the Apple Xsan clients.

Structure

```
typedef struct _setfilesizereq {
    uint64_t sq_size;
} SetFileSizeReq_t;
```

No reply structure

UNIX `ioctl` structure.
None. Use SetFileSizeReq_t directly.

reQuest Fields

sq_size	New file size.

Error Returns

VOP_EFAULT	Bad buffer offset.
VOP_EPERM	File is not writable by the caller.
VOP_EPERM	The user is not superuser and the "protect alloc" mount option is enabled.
Other	Communications failure with the FSM.

CvAPI_VerifyAlloc

This API allocates blocks within a file at the given offset.

Handle

The target file.

Notes

This call allocates all extents needed to fill the given range within a file. If any portion of the given range is already allocated, that portion is skipped. All additional space needed to form a completely allocated range is then filled in and a successful return status is set. If the given range is completely allocated, no allocations will be done and a successful status will be returned.

In order to ensure that only the requested allocation and no more is provided, use the ALLOC_NOMORETHAN flag. If this flag is not used the allocation might be rounded to optimize allocations.

In the case of an allocation failure, no space will be allocated and an error will be returned.

CvAPI_VerifyAlloc does not modify the file's size.

Structure

```
typedef struct  _VerifyAllocReq {
     uint64_tvq_size;

     uint64_tvq_offset;

     uint32_tvq_flags;

#define ALLOC_STRIPE_ALIGN0x04
#define ALLOC_NOMORETHAN0x20

     uint32_tvq_pad1;

} VerifyAllocReq_t;
```

No reply structure

Unix ioctl structure:
None. Use VerifyAllocReq_t directly.

reQuest Fields

vq_size	Size, in bytes, of the allocation request.
vq_offset	Offset, in bytes, of the start of the allocation range.
vq_flags	Flags to affect allocation behavior.

Error Returns

VOP_ENOSPC	Not enough free space from which to allocate.
VOP_EINVAL	Invalid arguments.
Other	Communications failure with the FSM.

Quality of Service and Real Time I/O APIs

This section describes the file system APIs that pertain to quality of service and real time IO.

CvApi_DisableRtio

This API disables (clears) the file's real-time attribute, making all further I/Os gated (if the stripe group is still in real-time mode).

Handle

Target file.

Notes

Files are also ungated by closing.

Structure

None.

CvApi_EnableRtio

This API puts a file handle into real time (ungated) mode.

Handle

Target file.

Notes

Ungated file handles are allowed full, unfettered (ungated) access to the SAN. Handles remain ungated until explicitly disabled or closed.

It is important to note that gating occurs on a handle basis, not a file basis. It is therefore possible for multiple threads with multiple handles to be sharing a file, and for some of them to receive ungated (real time) access, while the remainder is gated.

It is not necessary to explicitly enable RTIO on a handle via this call if the handle refers to a regular file and the handle was specified in the call to CvApi_SetRtio.

Structure

Optional: If the RtReq_t structure is provided as an argument, the rq_flags field is queried to determine if extents should be pre-loaded. See CvApi_SetRtio for more information.

Reply Fields

pr_extent	Extent information for the target offset.
pr_breadth	Stripe breadth. This is the amount of data written on each disk.
pr_depth	Stripe depth. This is the number of disks in a stripe.
pr_voloffset	Amount to skip in basic blocks from start of volume (size of disk label info).
pr_blkoffset	Device block offset from beginning of disk.
pr_edev	Unit object pointer. Fairly useless outside of the kernel.

CvApi_GetRtio

This API retrieves the real time parameters for a stripe group.

Handle

Any file in the file system.

Notes

This API returns the real time parameters for a stripe group. The parameters are in I/Os/sec.

Structure

This uses the RtReq_t structure with the RT_GET flag to request the real-time parameters for a stripe group.

```
typedef struct _rtqueryreply {
    uint32_trrq_sgid;
    uint32_trrq_pad;

    int32_trrq_limit;
    int32_trrq_cur;

    int32_trrq_nrtio_hint;
    uint32_trrq_nrtio_clients;
} RtQueryReply_t;
```

UNIX ioctl structure

```
typedef union rtqueryreqrep {
    RtReq_treq;
    RtQueryReply_treply;
} RtQueryReqReply_t;
```

reQuest Fields

See SETRTIO

Reply Fields

rrq_sgid	Stripe group number.
rrq_limit	Configured real time I/O limit, in I/Os/sec.
rrq_cur	Current amount of real time I/O committed to clients.
rrq_nrtio_hint	Amount of non-real time I/O a client is most likely to obtain when requesting a non-real time I/O token.
rrq_nrtio_clients	Number of clients with outstanding non-real time I/O tokens.

CvApi_QosClientStats

This API gets the QOS statistics for all connected clients.

Handle

Target file.

Notes

This call obtains the amount of real time and non-real time currently allocated for each client in the SAN.

The number of client connections the FSM can support is fixed by the configuration file. To retrieve the number of client connections available, the caller should first specify zero as the maximum number of clients. This returns the size of the connection table on the FSM. The caller can then allocate an array large enough to hold the QOS information for all clients, and pass the array in a second call.

Structure

Following is the structure that returns information about each connected client. For QOS purposes, only clients that have the QSTAT_CLIENT flag set will have any valid QOS information. The QSTAT_VALID flag means the entry in the connection table is valid, and the QSTAT_ADM flag means the connection is an “administrative tap.”

```
typedef struct rtclientstat {
    uint32_tc_ipaddr;
    uint32_tc_rtios;
    uint32_tc_nonrtios;
    uint32_tc_flags;
          #define QSTAT_VALID0x1/* entry valid */
          #define QSTAT_ADM0x2/* adm connection */
          #define QSTAT_CLIENT0x4/* client connection */
} RtClientStat_t;

typedef struct _rtclientstatreq {
    uint32_tcq_nclients;
    uint32_tcq_sg;
    CvUserAddr_tcq_buf;/* RtClientStat_t */
} RtClientStatReq_t;

typedef struct _rtclientstatreply {
    uint32_tcr_maxclients;/* max entries on fsm */
    uint32_tcr_pad;
} RtClientStatReply_t;

typedef union rtclientstatreqrep {
    RtClientStatReq_treq;
    RtClientStatReply_treply;
} RtClientStatReqReply_t;
```

reQuest Fields

cq_nclients	Number of RtClientStat_t entries in the .cq_buf field. If this field is zero, then the call will return the max number of clients in the FSM connection table. This information can be used on subsequent calls to size the request buffer.
cq_sg	Stripe group to query.
cq_buf	Address of RtClientStat_t buffer where information will be deposited.

Reply Fields

Cr_maxclients	Maximum number of entries in the FSM connection table. Callers can use this information to size their buffer to the largest possible size.

Error Returns

VOP_EFAULT	Bad buffer offset.
Other	Communications failure with the FSM.

CvApi_SetRtio

This API requests real time IO on an individual stripe group or file. See also the Platform Dependencies section for this API.

Handle

Root directory. Affects all files on stripe group.

or

Handle to target file. Affects only specific target handle/file descriptor.

Notes

This call enables real time I/O (RTIO) on a stripe group basis. RTIO is on an individual stripe group, not file system basis, since stripe groups can have very different access characteristics and can be used for very different file types (for example, audio versus video).

The caller specifies the maximum number of IOs per second or MB/sec. that they expect to utilize on the stripe group. Either one, but not both, may be specified. The number of IOs per second can be converted into MB/sec. by dividing the MB/sec. rate by the block size, stripe width, and stripe depth.

For example:Given an 8 disk stripe group with a *StripeBreadth* of 16 and an *FsBlockSize* of 4k, a requested rate of 50 MB/sec. would be (50mb/sec. * 1024k) / (8 * 16 *4k) = 100/sec.

The FSM may grant some amount less than requested unless the RT_MUST flag is set.

Once the FSM has returned a non-zero value in the reply structure, the partition group is in real-time mode. If the caller chooses to not accept the values returned by the FSM, it is the caller's responsibility to disable RTIO on the stripe group.

If the handle/file descriptor is the handle to the root of the file system ("\/"),the call affects all files on the designated stripe group. Closing the root handle will not disable RTIO on the stripe group. It must be implicitly cleared by specifying zero in either the rq_rtios or the rq_rtmb fields and setting the RT_CLEAR flag.

Setting the RT_CLEAR flag with a non-zero value in either rq_rtios or rq_rtmb only releases the amount specified; specifying zero in those fields completely disables real time IO on the stripe group.

If the handle/file descriptor is for a regular file in the stripe group, the call has slightly different semantics. Using a target handle is equivalent to specifying RTIO on the root directory, followed by a call to put the handle into real time mode (CvApi_EnableRtio). All other non-real time IO handles on the stripe group will be gated, as when specifying RTIO on the root directory. RTIO will be disabled when the handle is closed, either explicitly or implicitly, and the bandwidth returned to the system.

Note: If a file is opened with multiple handles, each specifying a different amount of real time IO, the RTIO be released only when the last handle has been closed. No RTIO will be released until the last handle has been closed.

The mode of specifying a target file allows non-cooperating applications to request differing amounts of real time IO on the same stripe group. Upon successful return from the call, the target handle is in real time (ungated) mode; no further calls need be made. All accesses to other non-real time handles will be gated. The additional semantic difference is that RTIO is returned to the system when the handle is closed.

If the handle/file descriptor is for a regular file and the RT_NOLOAD flag is not set, all extents for the file are preloaded into the file system. This cuts down on cold-start overhead. However, if the file has many extents, this operation can take a long time to complete.

For both modes, if the FSM is rebooted or is reset, the client file system will attempt to re-negotiate the real time IO requirements with the FSM. This may introduce a period of instability. It may not be possible to guarantee the same bandwidth requests as before, due to request ordering during the recovery period. If the same amount of real-time bandwidth cannot be obtained during recovery processing, the next access to a handle that is real time mode will fail and an event will be logged in the system log.

If the handle/file descriptor is for the root of the file system and the RT_ABSOLUTE flag is set, the system adjusts the amount of currently allocated RTIO to bring it in line with the request. This is useful for systems that are continually adjusting the amount of RTIO available based on external criteria (such as a video stream bit rate).

Note: The following extent crossing functionality will be implemented in a future release.

File-based RTIO encompasses the entire file. Since the file can have multiple extents, it is possible that it will cross stripe groups. If the RT_SEQ bit is set in rq_flags field, the client FSD assumes that access will be sequential through the file. The client FSD ensures that when crossing from one stripe group to the next, the new stripe group is put into the appropriate real time mode before any access occurs. The real time requirements of the previous stripe group will be released.

Structure

```
typedef struct _rtreq {
    union {
        int32_tru_rtios;/* ios per sec */
        int32_tru_rtmb;/* mb per sec */
    } rq_un;

    #define rq_rtiosrq_un.ru_rtios
    #define rq_rtmbrq_un.ru_rtmb

    uint32_t    rq_flags;
        #define RT_IO0x01/* r_rtios valid */
        #define RT_MB0x02/* r_rtmb valid */
        #define RT_CLEAR0x04/* clear RT */
        #define RT_SET0x08/* set RT */
        #define RT_MUST0x10/* fail if can't satisfy */
        #define RT_SEQ0x20/* sequential IO */
        #define RT_GET0x40/* get rt params */
        #define RT_NOGATE0x80/* ungated IO */
        #define RT_NOLOAD0x100/* don't load extents */
        #define RT_ABSOLUTE 0x200 /* absolute req not incr */


    uint32_t    rq_sgid;/* stripe group ID */

} RtReq_t;

typedef struct _rtreply {
    union {
        int32_tru_rtios;/* ios per sec */
        int32_tru_rtmb;/* mb per sec */
    } rr_un;

    #define rr_rtiorr_un.ru_rtios
    #define rr_rtmbrr_un.ru_rtmb

    uint32_t    rr_flags;
        #define RT_IO0x01/* r_rtios valid */
        #define RT_MB0x02/* r_rtmb valid */
    uint32_t    rr_pad1;
} RtReply;
```

UNIX ioctl structure

```
typedef union rtreqrep {
    RtReq_treq;
    RtReply_treply;
} RtReqReply_t;
```

reQuest Fields

rq_rtios	Requested real time I/Os per sec. Valid only if RT_IO is set. RT_MB may not also be set.
rq_rtmb	Requested megabytes per sec. Valid only if RT_MB is set. RT_IO may not also be set.
rq_flags	Control flags

RT_IO	rr_rtios is valid.
RT_MB	rr_rtmb is valid.
RT_CLEAR	Clear real time from stripegroup. Either RT_CLEAR or RT_SET must be set.
RT_SET	Set real time parameters for stripegroup.
RT_MUST	Fail the request if the requested amount can't be satisfied. This prevents the FSM from returning a lesser value than what was requested.
RT_SEQ	The application will be performing sequential I/O and the client FSD should handle crossing of stripe groups. (This functionality will be implemented in a future release.)
RT_NOGATE	If RT_SET is specified put the handle into ungated mode. In this mode, the I/O using this handle consumes no RTIO and is not gated. If RT_CLEAR is specified, the handle is removed from this mode.
RT_NOLOAD	Do not pre-load extents for the file.
RT_ABSOLUTE	The amount specified in `the` rq_rtios field is an *absolute* value for the stripe group

specified in rq_sgid. The system adjusts the request depending on the current state of the stripe group and the current amount of RTIO already allocated. This flag is only valid when used on the root directory (that is, not on a regular file).

rq_sgid	Stripe group ordinal, identifying stripe group. See the API for retrieving the stripe group name to match names with ordinals.

Reply Fields

rr_rtios	Allowed real time I/Os per second. Valid only if RT_IO is set. May be less than or greater than the amount requested if RT_MUST flag is clear.
rr_rtmb	Allowed real time MB per second. Valid only if RT_MB is set. May be less than or greater than the amount requested if RT_MUST flag is clear.

Platform Dependencies

If the same file is shared in real time and non-real time mode on a Unix platform (that is, anything other than NT4, Win2k, or XP), the caller must use the fcntl(2) system call to differentiate between the real time and non-real time accesses. This is because UNIX platforms do not typically export file descriptor flags down to the file system. The value to use is different, depending on the platform. On Irix and Solaris, the caller must do a fcntl (fd, F_SETFL, O_SYNC) to identify the real time file descriptor.

Note: The file system cannot distinguish between the use of O_SYNC for identifying real time file descriptors and its use for specifying synchronous writes. Therefore, on Irix and Solaris platforms, when a file is opened O_SYNC (or if the O_SYNC is set on the file via fcntl), all writes will be synchronous and all I/O performed on the file will be non-gated.

On Linux, the caller must do a fcntl (fd, F_SETFL, O_NONBLOCK). All other file descriptors will be gated.

File System Configuration and Location Management APIs

These APIs are used primarily for reporting purposes and provide details on file parameters as well as configuration of the underlying disk volumes that make up a StorNext file system.

CvApi_GetAffinity

This API gets the affinity for a file.

Handle

Target file.

Notes

Affinities can direct allocations to specific stripe groups. This call will return the current affinity for the file.

Structure

No request structure

```
typedef struct getaffinityreply {
    uint64_tar_affinity;
} GetAffinityReply_t;
```

UNIX ioctl structure.

None. Use GetAffinityReply_t directly.

Reply Fields

ar_affinity	Current affinity identifier for file.

CvApi_GetExtList

This API gets the list of extents for a file.

Handle

Target file.

Notes

This call does not load the extents in the client file system extent map for a file. It returns as many extents as it can in a single call directly from the FSM.

This is an iterative call. It is the responsibility of the caller to allocate any free buffers. Since there may be many extents for a file, the caller can iterate over the list, specifying a different starting offset in gq_startfrbase. When there are no more extents available the call returns the platform equivalent of ENOENT.

The caller must allocate the buffer for the reply data, and initialize the gq_buf field to point to it. The buffer should be aligned on a minimum of an 8 byte boundary.

Structure

```
typedef sruct _cvexternalextent {
    uint64_tex_frbase;/* file relative offset */

    uint64_tex_base;/* fs starting offset */
    uint64_tex_end;/* fs ending offset, inclusive*/
    uint32_tex_sg;/* stripe group number */
    uint32_tex_depth;/* sg depth for this extent */
    uint32_tex_pad1;

} CvExternalExtent_t;

typedef struct _getextlistreq {
    uint64_t gq_startfrbase;
    uint32_t gq_numbufs;
    uint32_t gq_pad1;
    void*qb_buf;
} GetExtListReq_t;
```

```
typedef struct _getextlistreply {
    uint32_t gr_numreturned;
    uint32_t gr_pad;
} GetExtListReply_t;
```

UNIX ioctl structure

```
typedef union _getextlistreqrep {
    GetExtListReq_treq;
    GetExtListReply_treply;
} GetExtListReqReply_t;
```

Fields, CvExternalExtent_t

ex_frbase	File relative starting byte offset. This offset can be anywhere in the extent; it does not have to correspond exactly to the starting offset of an extent. Any extent that contains ex_frbase will be returned.
ex_base	File system starting byte offset.
ex_end	File system ending byte, inclusive. Since this byte offset is inclusive and specifies the last valid byte in the extent, this value will not be a multiple of the file system block size. To obtain the next extent, add one to `e_end` for the starting offset of the next extent.
ex_sg	ID of stripe group.
ex_depth	Depth of stripe group for this extent. Stripe groups can grow and shrink dynamically, so the depth of the extent may not match the current depth of the stripe group.

reQuest Fields

gq_startfrbase	Starting file relative byte offset.
gq_numbufs	Number of CvExtent_t sized buffers in response.
gq_buf	User allocated buffer where the call will place an array of CvExternalExtent_t structs.

Reply Fields

gr_numreturned	Number of CvExternalExtent_t elements in gq_buf.

Error Returns

VOP_ENOENT	No more entries available.
VOP_EFAULT	Buffer is invalid.
VOP_EINVAL	Invalid starting offset. The extent list has probably changed.

CvApi_GetPhysLoc

This API determines the physical location of any byte offset in a file.

Handle

Target file.

Notes

This call returns extent information about the target offset, as well as the location in the file system and on the target disk.

All offsets are rounded up to the nearest file system block size. All values are specified in bytes.

Structure

```
typedef struct _physlocreq {
      uint64_tpq_offset;
} PhysLocReq_t;

typedef struct _physlocreply {
      CvExtentpr_extent;

      uint64_tpr_breadth;
      uint64_tpr_depth;

      uint64_tpr_voloffset;/* VolHder sz in bytes    */
      uint64_tpr_blkoffset;/* Block offset in bytes   */

      uint32_tpr_edev;
      uint32_tpr_pad1;
} PhysLocReply_t;
```

Unix ioctl structure

```
typedef union _phylocreqreply {
      PhysLocReq_treq;
      PhysLocReply_treply;
} PhysLocReqReply_t;
```

reQuest Fields

pq_offset	Desired byte offset.

Reply Fields

pr_extent	Extent information for the target offset.
pr_breadth	Stripe breadth. This is the amount of data written on each disk.
pr_depth	Stripe depth. This is the number of disks in a stripe.
pr_voloffset	Amount to skip in basic blocks from start of volume (size of disk label info).
pr_blkoffset	Device block offset from beginning of disk.
pr_edev	Unit object pointer. Fairly useless outside of the kernel.

CvApi_GetSgInfo

This API gets the parameters associated with a stripe group.

Handle

Any, but usually handle to root directory.

Notes

This call retrieves all the kernel states associated with the specified stripe group, including the affinity identifiers.

This call is local to the client and only queries the information on partition groups that are valid on the client. If a stripe group is not being accessed by the client and is therefore not currently active on the client, then the call cannot return information about the stripe group. It will return ENOENT.

Structure

```
#define SG_NAMELEN256

typedef struct _sginforeq {
    uint32_tsqi_id;
    uint32_tsqi_pad0;
    CvUserAddr_tsqi_keys;
    CvUserAddr_tsqi_keycnt;
} SgInfoReq_t;

typedef struct _sginforeply {
    uint64_tsri_totblks;
    uint64_tsri_freeblks;

    uint32_tsri_breadth;
    uint32_tsri_depth;

    uint32_tsri_flags;
        #define SG_PART_VALID0x1
        #define SG_PART_ONLINE0x2
        #define SG_PART_METADATA0x4
        #define SG_PART_JOURNAL0x8
        #define SG_PART_EXCLUSIVE0x10
    uint32_tsri_bsize;

    charsri_name[SG_NAMELEN];

} SgInfoReply_t;
```

UNIX ioctl structure:

```
typedef union _sginfoqrep {
    SgInfoReq_treq;
    SgInfoReply_treply;
} SgInfoReqReply_t;
```

reQuest Fields

sqi_id	ID of stripe group to search for.
sqi_keys	A user supplied uint64_t array to be filled with affinity identifiers. If sqi_keys is NULL, no identifiers will be returned.
sqi_keycnt	Pointer to a uint32_t. On input, this integer should be the maximum number of affinity identifiers to be returned (i.e. the size of the array sqi_keys). On output, the integer will contain the number of indentifiers actually returned. If sqi_keycnt is NULL, no keys will be returned.

Reply Fields

sri_totblks	Total blocks on stripe group.	
sri_freeblks	Free blocks available on stripe group.	
sri_breadth	Stripe breadth in file system blocks (number of blocks written to each disk in a single chunk).	
sri_depth	Stripe breadth (number of disks in stripe group).	
sri_flags	State flags	
	SG_PART_ONLINE	Stripe group is valid and online.
	SG_PART_VALID	Stripe group is valid; may be offline.
	SG_PART_METADATA	Stripe group is used for metadata.
	SG_PART_JOURNAL	Stripe group is used for journal.
	SG_PART_EXCLUSIVE	Stripe group is exclusive.
sri_bsize	Block size for stripe group.	
sri_name	Name of stripe group.	

Error Returns

VOP_ENOENT	No more entries available.

VOP_EFAULT	Buffer is invalid.
VOP_E2BIG	Stripe group has more affinity IDs than the user-supplied sqi_keycnt.

CvApi_GetSgName

This API gets the ASCII name for a stripe group or the ordinal for a specified ASCII name.

Handle

Any, but usually handle to root directory.

Notes

Many of the requests use an ordinal to identify a stripe group. This API converts an ordinal into its user-visible name. In this manner, a user can iterate through the stripe groups and match up the names in the FSM configuration file with the current ordinal. The names returned will match the [StripeGroup xxx] directive in the FSM config file. The call returns the name of the stripe group matching the ordinal specified in the sq_id field, or the ordinal of the stripe group with the matching name in sq_name .

This call is local to the client, and only queries the information on partition groups that are valid on the client. If a stripe group is not being accessed by the client and is therefore not currently active on the client, the call cannot return information about the stripe group. It will return ENOENT.

Each stripe group name is 256 bytes long, including NULL. This is the minimum buffer size. Buffers that are shorter than the minimum will result in an exception, returning EFAULT.

When there are no more entries available, the call returns ENOENT. If no stripe group corresponds to the specified ordinal, it means that the list changed between calls, and the call returns EINVAL.

Structure

```
#define SG_NAMELEN256
```

```
typedef struct _sgnamereq {

    uint32_tsq_flags;
        #define SG_GETNAME1
        #define SG_GETNUM2

    uint32_tsq_pad1;

    union {
        uint32_tsqu_id;
        charsqu_buf[SG_NAMELEN];
    } sq_un;

} SgNameReq_t;
        #define sq_idsq_un.squ_id
        #define sq_namesq_un.squ_buf

typedef struct _sgnamereply {
    union {
        uint32_tsru_id;
        charsru_buf[SG_NAMELEN];
    } sr_un;

} SgNameReply_t;
```

UNIX ioctl structure

```
typedef union _sgnamereqrep {
    SgNameReq_treq;
    SgNameReply_treply
} SgNameReqReply_t;
```

reQuest Fields

sg_flags	If SG_GETNAME is set, sq_id contains the ordinal of the stripe group to search for. If SG_GETNUM is set, then sq_name contains the ASCII name of the stripe group to search for.
sr_un.sru_id	ID of stripe group to search for.

sr_un.sru_buf	Name of stripe group to search for.

Reply Fields

sr_un.sru_buf	Buffer with name of stripe group.
sr_un.sru_id	Ordinal of stripe group.

Error Returns

VOP_ENOENT	No more entries available.
VOP_EFAULT	Buffer is invalid.
VOP_EINVAL	Invalid stripe group ordinal.

CvApi_SetAffinity

This API sets the affinity for a file.

Handle

Target file.

Notes

Affinities can direct allocations to specific stripe groups. This call sets the current affinity for the file and affects all future allocations.

Structure

```
typedef struct setaffinityreq {
    uint64_tsq_affinity;
} SetAffinityReq_t;
```

No reply structure

UNIX `ioctl` structure.

None. Use SetAffinityReq_t directly.

Request Fields

sq_affinity	Affinity identifier for file>

Access Management APIs

These APIs allow you to control concurrent file operations and quotas, and they provide additional reporting utilities.

CvApi_ClearConcWrite

This API disables concurrent writes to a file.

Handle

Target file.

Notes

This API disables concurrent writes for all users of the file. This works on a file, not handle basis.

Structure

None.

CvApi_ClearRdHoleFail

This API causes reads from a hole in the file to return zero (default behavior).

Handle

Target file.

Notes

This restores the default behavior of returning zero for a read from non-allocated space in a file.

Structure

None.

CvApi_CvFstat

This API gets the UNIX-like stat struct from a file.

Handle

Target file.

Notes

This call performs much the same as the UNIX stat(2) call. See also CvApi_StatPlus.

Structure

No request structure.

```
typedef struct _statreply {
    int32_tsr_dev;
    uint32_tsr_mode;

    uint64_tsr_ino;

    uint64_tsr_size;

    uint64_tsr_nblocks;

    int32_tsr_nlink;
    uint32_tsr_bsize;

    int32_tsr_uid;
    int32_tsr_gid;

    int32_tsr_rdev;
    int32_tsr_atim;
    int32_tsr_mtim;
    int32_tsr_ctim;
} StatReply_t;
```

UNIX `ioctl` structure:

None. Use StatReply_t directly.

Reply Fields:

sr_dev	Device identifier, unique per mounted file system.
sr_mode	Unix mode.
sr_ino	File handle, unique per file system.
sr_size	Size of file in bytes.
sr_nblocks	Number of "basic" (512 byte) blocks allocated to file.
sr_nlink	Number of links to the file.
sr_bsize	Blocksize of file system.
sr_uid	Unix User Identifier.
sr_gid	Unix Group Identifier.
sr_rdev	Device node for devices. 0 on the file system.
sr_atim	Last access time in seconds since January 1, 1970.
sr_mtim	Last modify time in seconds since January 1, 1970.
sr_cim	Last "change" time in seconds since January 1, 1970.

CvApi_CvOpenStat

This API retrieves open status on the given file.

Handle

Target file.

Notes

This method returns status regarding the open state of a file. These status objects are useful in finding the open state across the cluster. The os_sharedwrite and os_sharedread bits may be used to indicate whether a file is opened on another client. The os_opencount and os_refcount values indicate how the local client is using the given file.

Structure

```
typedef struct _openstatreply {
    uint32_tos_sharedwrite;
    uint32_tos_sharedread;
    uint32_tos_opencount;
    uint32_tos_refcount;
} OpenStatReply_t;
```

CvApi_GetDiskInfo

This API gets disk information for a stripe group.

Handle

Any, but usually handle to root directory.

Notes

The information returned by this call is somewhat limited. More information may be returned in future releases.

Structure

```
#define MAXPATHS 4
typedef struct _cvdiskinfo {
        char     di_name[256]; /* CVFS disk name */
        uint32_t  di_nameloc;   /* disk byte offset to disk name */
        uint32_t  di_vhsize;    /* Volume header size */
        char      di_serialnum[64]; /* WWN or disk serial number */
        uint32_t  di_sectorsize;    /* disk sector size in bytes */
        uint32_t  di_npaths;      /* number of active paths */
        struct di_dev_info {
                char d_bdev[256];     /* block device name */
                char d_rdev[256];     /* character device name */
        } di_paths[MAXPATHS];
} CvDiskInfo_t;

typedef struct _diskinforeq {
        uint32_tsq_sg;
        uint32_tsq_pad0;
        CvUserAddr_tsq_dinfobuf;
        CvUserAddr_tsq_dcnt;
} DiskInfoReq_t;
```

No reply structure

UNIX `ioctl` structure.

None. Use DiskInfoReq_t directly.

Fields, CvDiskInfo_t

di_name	Disk name. For example, "CvfsDisk_0".
di_nameloc	Disk offset (in bytes) where the disk name resides.
di_vhsize	The size of the disk volume header, in bytes.
di_serialnum	Either the disk's WWN (for fibre-attached devices) or its serial number.
di_sectorsize	Sector size of the disk, in bytes.
di_npaths	Number of paths to the disk.
di_paths	A structure containing the block and character device names for active paths to the disk. A single disk may have up to MAXPATHS (4) paths. Therefore, there may be up to 4 block and raw device names for a single disk.

reQuest Fields

sq_id	ID of stripe group.
sq_dinfobuf	A user supplied CvDiskInfo_t array to be filled in.
sq_dcnt	Pointer to a uint32_t. On input, this integer should be the maximum number of disk info structures to be returned (that is, the size of the array sq_dinfobuf). On output, the

integer will contain the number of disk info structures actually returned.

Error Returns

VOP_ENOENT	sq_id is not a valid stripe group number.
VOP_E2BIG	The stripe group contains more disks than the user supplied sq_dcnt.
VOP_EFAULT	sq_dinfobuf or sq_dcnt is an invalid pointer.

CvApi_GetQuota

This API gets the current quota usage and limits for a given user or group.

Handle

Any, but usually handle to root directory.

Notes

The source handle may refer to any open file or directory that resides on the file system. It is not necessary that the file be owned by the user or group whose quota values are being queried.

Structure

```
#define MAX_QUOTA_NAME_LENGTH 256

typedef struct getquotareq {
    uint32_t  gq_type;
        #define QUOTA_TYPE_USER(uint32_t)'U'
        #define QUOTA_TYPE_GROUP(uint32_t)'G'

    uint32_t  gq_pad;
    char      gq_quotaname[MAX_QUOTA_NAME_LENGTH];
} GetQuotaReq_t;

typedef struct getquotareply {
    uint64_t  gr_hardlimit;  /* in bytes */
    uint64_t  gr_softlimit;  /* in bytes */
    uint64_t  gr_cursize;    /* in bytes */
    uint32_t  gr_timelimit;  /* in minutes */
    uint32_t  gr_timeout;    /* in seconds since January 1, 1970 */
} GetQuotaReply_t;

typedef union _getquotareqreply {
    GetQuotaReq_t    req;
    GetQuotaReply_t  reply;
} GetQuotaReqReply_t;
```

reQuest Fields

gq_type	Either QUOTA_TYPE_USER or QUOTA_TYPE_GROUP.
gq_quotaname	The null-terminated name of the user or group whose quota is being queried.

Reply Fields

gr_hardlimit	The hard quota limit in bytes, rounded up to the nearest file system block.
gr_softlimit	The soft quota limit in bytes, rounded up.
gr_cursize	The current usage in bytes, rounded up.
gr_timelimit	When the soft limit is exceeded, the amount of time (in minutes) before the soft limit will be treated as a hard limit.
gr_timeout	If the soft quota has been exceeded, gr_timeout will contain the time (in seconds since January 1, 1970) when the soft limit will be treated as a hard limit.

Error Returns

VOP_ENOTSUP	The quota system is not enabled on the FSM and/or client.
VOP_INVAL	gq_type is not QUOTA_TYPE_USER or QUOTA_TYPE_GROUP.\

VOP_ENOENT	gq_quotaname is not a valid user or group name.

CvApi_GetVerInfo

This API retrieves version information.

Handle

Any.

Notes

This returns the version, build, and creation date of the kernel, as well as the version of the external API. Note that since individual calls can be up-revved independently, the version number that is returned may not be the same for all calls.

Structure

No request structure.

```
#define CVVERINFO_MAX   64

typedef struct _verinfo_reply {
    charvr_version[CVVERINFO_MAX];
    charvr_build[CVVERINFO_MAX];
    charvr_creationdate[CVVERINFO_MAX];
    uint32_tvr_apiversion;
    uint32_tvr_pad1;
} VerInfoReply_t;
```

Reply Fields

vr_version	A string containing the release and build number, such as #!@$ CVFS Client Revision 2.1.1 Build 60.
vr_build	A string containing platform information, such as #!@$ Built for Windows 2000 i386.
vr_creationdate	A string containing the creation date, such as #!@$ Created on Thu Dec 19 14:15:53 PST 2002.

vr_apiverion	The current version of the external API.

CvApi_LoadExtents

This API pre-loads a range of extents for a file.

Handle

Target file.

Notes

This call requests that the extent information from the FSM be pre-loaded into the file system. This cuts down on first access (cold-start) time. If there are any holes in the file, they will be preserved. No space is allocated by this call.

Structure

```
typedef struct _LoadExtReq {
    uint64_tlq_size;
    uint64_tlq_offset;
    uint32_tlq_pad1;
    uint32_tlq_pad2;
} LoadExtReq_t;
```

reQuest Fields

lq_size	The number of bytes to load. Specifying zero (0) will cause the extents for the entire file to be loaded. If lq_size + lq_offset extends beyond the end of the file, extents will be loaded up to the end of the file.
lq_offset	Starting offset to begin loading. The offset does not have to be on an existing extent boundary; any offset will suffice.

Error Returns

VOP_EINVAL	Offset is greater than end of file.

Other	Communications failure with the FSM.

CvApi_SetConcWrite

This API allows concurrent writes to a file.

Handle

Target file.

Notes

This call allows multiple handles to write concurrently to a file without being serialized. It is important to note that this `ioctl` call operates on the file, not the handle. Once a file is in concurrent write mode, all users of the file will be able to write concurrently. The concurrent write feature is reset to the default (non-concurrent) when the last user closes the file, or when it is explicitly disabled.

No data buffering is done. Malformed I/O returns the platform equivalent of EINVAL. It is the responsibility of those using this feature to maintain separate handles and separate offsets. The primary users of this feature are drivers that are layered directly on top of the FSD and use IRPs to communicate.

Structure

None.

CvApi_SetQuota

This API sets quota limits for a given user or group.

Handle

Any, but usually handle to root directory.

Notes

The source handle may refer to any open file or directory that resides on the file system. It is not necessary that the file be owned by the user whose quota values are being queried.

The specified hard and soft limits are automatically rounded up to the nearest file system block by the quota system.

Structure

```
#define MAX_QUOTA_NAME_LENGTH 256

typedef struct _setquotareq {
    char      sq_quotaname[MAX_QUOTA_NAME_LENGTH];
    uint32_t  sq_type;
        #define QUOTA_TYPE_USER(uint32_t)'U'
        #define QUOTA_TYPE_GROUP(uint32_t)'G'

    uint32_t  sq_timelimit;      /* in minutes */
    uint64_t  sq_hardlimit;      /* in bytes */
    uint64_t  sq_softlimit;      /* in bytes */
} SetQuotaReq_t;
```

No reply structure

UNIX `ioctl` structure.

None. Use SetQuotaReq_t directly.

reQuest Fields

sq_quotaname	The name of the user or group whose quota limits are being set.
sq_type	Either QUOTA_TYPE_USER or QUOTA_TYPE_GROUP.
sq_timelimit	The soft quota grace period, in minutes.
sq_hardlimit	The hard limit, in bytes.
sq_softlimit	The soft limit, in bytes.

Error Returns

VOP_ENOTSUP	The quota system is not enabled on the FSM and/or client.
VOP_INVAL	sq_type is not QUOTA_TYPE_USER or QUOTA_TYPE_GROUP.
VOP_ENOENT	sq_quotaname is not a valid user or group name.
VOP_IO	sq_softlimit is greater than sq_hardlimit (or an internal error occurred).

CvApi_SetRdHoleFail

This API causes reads from a hole in a file to fail.

Handle

Target file.

Notes

During recovery processing after a client or FSM crash, it can be desirable to have reads from non-allocated space return an error rather than the default of zeros. This call affects all handles of a file. If an attempt is made to read from non-allocated space, the platform equivalent of EACCESS will be returned. On Windows, this translates into STATUS_ACCESS_DENIED. This behavior remains in affect until all handles to the file have been closed or the feature has been explicitly cleared.

Structure

None.

CvApi_StatFs

This API gets file system information.

Handle

Any, but usually handle to root directory.

Notes

Gets basic file system information.

Structure

No request structure.

```
typedef struct __statfs {
    uint32_tfr_options
        #define FSOPTION_DMIG(1<<0)
        #define FSOPTION_QUOTAS(1<<1)
        #define FSOPTION_BRLS(1<<2)
        #define FSOPTION_GLOBALSU(1<<3)
        #define FSOPTION_WINSEC(1<<4)

    uint32_tfr_blocksize;

    uint64_tfr_epoch;

    uint64_tfr_total_blocks;

    uint64_tfr_blocks_free;

    uint32_tfr_reserved[8];

} StatFsReply_t;
```

UNIX `ioctl` structure:

None. Use StatFsReply_t directly.

Reply Fields

fr_options	Mask containing enabled file system options.
fr_blocksize	File system basic block size in bytes.
fr_epoch	File system creation date in microseconds since January 1, 1970.
fr_total_blocks	Total capacity of the file system in blocks.
fr_blocks_free	Number of unallocated blocks.

fr_reserved	Fields reserved for future use.

Error Returns

VOP_EFAULT	Bad buffer offset.
VOP_ENOMEM	Out of memory.
Other	Communications failure with the FSM.

CvApi_StatPlus

This API gets the UNIX-like stat struct from a file, plus additional file information

Handle

Target File

Notes

This call performs much the same as the UNIX stat(2) call except that additional information is returned such as the storage state. Also see CVApi_CVFStat.

Structure

No request struct.

```
typedef struct _statplusreply {
    int32_tsr_dev;
    uint32_tsr_mode;

    uint64_tsr_ino;

    uint64_tsr_size;

    uint64_tsr_nblocks;

    int32_tsr_nlink;
    uint32_tsr_bsize;

    int32_tsr_uid;
    int32_tsr_gid;

    int32_tsr_storagestate;
        #define STORESTATE_ON_DISK_ONLY(1)
        #define STORESTATE_ON_TAPE_ONLY(2)
        #define STORESTATE_ON_DISK_AND_TAPE(3)

    int32_tsr_atim;

    int32_tsr_mtim;
    int32_tsr_ctim;

    uint32_tsr_reserved[8];
} StatPlusReply_t;
```

UNIX `ioctl` structure:

None. Use StatPlusReply_t directly.

Reply Fields

sr_dev	Device identifier, unique per mounted file system.
sr_mode	Unix mode.
sr_ino	File handle, unique per file system.
sr_size	Size of file in bytes.
sr_nblocks	Number of “basic” (512 byte) blocks allocated to file.
sr_nlink	Number of links to the file.
sr_bsize	Blocksize of file system.
sr_uid	Unix User Identifier.
sr_gid	Unix Group Identifier.
sr_storestate	Data migration: current file storage state.
sr_atim	Last access time in seconds since January 1, 1970.
sr_mtim	Last modify time in seconds since January 1, 1970.

sr_cim	Last “change” time in seconds since January 1, 1970.
sr_reserved	Fields reserved for future use.

Error Returns

VOP_EFAULT	Bad buffer offset.
VOP_ENOMEM	Out of memory.
Other	Communications failure with the FSM.

CvApi_SwapExtents

This API swaps all the extents for a file.

Handle

Source handle.

Notes

During defrag, the defrag utility allocates an extent and copy into it the data for a file. When the copy is complete, it attempts to swap the extents for the file, replacing the prior highly fragmented extent list in a file with, hopefully, a much smaller one.

This API swaps all the extents for one file (the file identified by the calling handle) with the file identified in the API structure. After this operation, the caller can unlink the source handle (if desired).

Structure

```
typedef struct _swapextreq {
    uint64_tsq_targhandle;
    uint32_tsq_msec;
    uint32_tsq_pad1;
} SwapExtReq_t;
```

No reply structure

UNIX `ioctl` structure.

None. Use SwapExtReq_t directly.

reQuest Fields

sq_handle	Handle to target file that will have its extents replaced by the source. This handle should be acquired by calling CVApi_CVFStat and using the sr_ino field in the returned StatReply_t object. **NOTE:** Do not attempt to use the st_ino returned by UNIXstat(2), as this will often cause CvApi_SwapExtents to fail.
sq_msec	A date in UNIX time(2) format. If non-zero, the value of sq_msec is checked against the modification time of the source file. If they are not the same, VOP_EBUSY is returned. So, sq_msec can be used as an additional sanity check to prevent attempts to defragment files that are actively being written.

Error Returns

VOP_EBUSY	The file is in use.
VOP_EPERM	The user does not have permission to defrag the file. Under UNIX, the user must be superuser or the owner of the file. Under Windows, the user must have write access to the file.
VOP_ENOENT	The file doesn't exist.
VOP_EINVAL	The file is not a “regular” file

Appendix A

File System API Example

Following is a test API sample program that illustrates how to use many of the StorNext File System APIs described in this document. This example applies only to the File System APIs, not the Storage Manager APIs.

```
/*
Copyright (c) 1997-2006
    All Rights Reserved.
    StorNext File System
Provided AS-IS, with no warranties expressed or implied.
*/
/*
 * tapi.c -- Test API
 *
 * Sample application to illustrate how to use the SNFS external API.
 *
 * This requires a 'getopt' routine that is available on most unix boxes,
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>
#include <time.h>            /* localtime() */

#if defined (_WIN32)
  #include <io.h>            /* for posix open(2) */
#endif

/* Local Headers */
#if defined(_WIN32)
```

```
  #include <cvinttypes.h>       /* for integer typedefs */
  #include <bsd_getopt.c>       /* need a getopt routine */
  #include <bsd_strtoll.c>
#endif

extern char *optarg;
extern int optind;

#if defined(__linux__)
  #include <stdint.h>
   #ifndef NULL
      #define NULL 0
   #endif /* NULL */
#endif /* linux */

#include <extapi.h>
#include <cvapi.h>
```

```
/* Macros */
#if defined(_WIN32)
  #define ARG64X    "0x%I64x"
  #define ARG64D    "%I64d"
#else
  #define ARG64X    "0x%llx"
  #define ARG64D    "%lld"
#endif /* _WIN32 */

#define EXTMAX24
#define MAXDISKS 32

/* File scope variables */
char   *Progname;
int    Verbose;
int    ErrorFlag;

/* External variables */
extern int optind;

/* External functions */
/* Structures and Unions */

/* Signal Catching Functions */
    /* NONE */
void
Usage()
{
    fprintf(stderr, "Usage: %s <args> filename \n", Progname);
    fprintf(stderr, "\t -A alloc space\n");
    fprintf(stderr, "\t -a affinity\n");
    fprintf(stderr, "\t -B stat fs\n");
    fprintf(stderr, "\t -b broadcast size on alloc\n");
    fprintf(stderr, "\t -C set concurrent write\n");
    fprintf(stderr, "\t -c clear concurrent write\n");
    fprintf(stderr, "\t -D get disk info for stripe group <n>\n");
    fprintf(stderr, "\t -E print extent list\n");
    fprintf(stderr, "\t -F set affinity\n");
    fprintf(stderr, "\t -f get affinity \n");
    fprintf(stderr, "\t -G get stripe group name from Ordinal\n");
    fprintf(stderr, "\t -g get stripe group ordinal from Name\n");
    fprintf(stderr, "\t -I Info for stripe group <n> \n");
    fprintf(stderr, "\t -L get physical Location for offset in file\n");
    fprintf(stderr, "\t -l (alloc flag) load extent in client fsd\n");
    fprintf(stderr, "\t -N keep size on allocation \n");
    fprintf(stderr, "\t -n nbytes [kmg] \n");
    fprintf(stderr, "\t -O Open stat file\n");
    fprintf(stderr, "\t -o offset [kmg] \n");
    fprintf(stderr, "\t -P punch hole \n");
    fprintf(stderr, "\t -p get PerfectFit status\n");
    fprintf(stderr, "\t -Q (user) set the quota limits for (user)\n");
    fprintf(stderr, "\t -q (user) get the quota usage and limits for (user)\n");
    fprintf(stderr, "\t -R set read hole fail \n");
    fprintf(stderr, "\t -r clear read hole fail \n");
    fprintf(stderr, "\t -S stat file (old version)\n");
    fprintf(stderr, "\t -s (alloc flag) stripe align allocation \n");
    fprintf(stderr, "\t -T stat file (new \"plus\" version)\n");
    fprintf(stderr, "\t -t (alloc flag) set perfect fit status\n");
    fprintf(stderr, "\t -v verbose\n");
    fprintf(stderr, "\t -V get version info\n");
    fprintf(stderr, "\t -w wait before exiting\n");
    fprintf(stderr, "\t -x load extents\n");
    fprintf(stderr, "\t -Y VerifyAlloc allocation\n");
    fprintf(stderr, "\t -y Toggle PerfectFit status\n");
    fprintf(stderr, "\t -Z set file size\n");
    fprintf(stderr, "\t -z size [kmg]\n");

}
```

```
/*
 * PURPOSE
 *  Print out the time for an 'ls' style listing
 */
void
PrintTime(
    int32_tf_secs)
{
    struct tm*tp;
    tp = localtime((time_t *)&f_secs);
    switch (tp->tm_mon) {
    case 0:
       printf("Jan");
       break;
           case 1:
       printf("Feb");
       break;
    case 2:
       printf("Mar");
       break;
    case 3:
       printf("Apr");
       break;
    case 4:
       printf("May");
       break;
    case 5:
       printf("Jun");
       break;
    case 6:
       printf("Jul");
       break;
    case 7:
       printf("Aug");
       break;
    case 8:
       printf("Sep");
       break;
    case 9:
       printf("Oct");
       break;
    case 10:
       printf("Nov");
       break;
    case 11:
       printf("Dec");
       break;
    }
    printf(" %d ", tp->tm_mday);
    printf("%02d:%02d", tp->tm_hour, tp->tm_min);
}
/*
 * PURPOSE
 *  Stat file, old version
 */
int
StatFile(
    int     f_fd,
    char    *f_filename)
{
    int         error = 0;
    StatReply_t     sb;
    /*
     * Get the stats
     */
    error = CvApi_CvFstat(f_fd, &sb);
    if (error) {
      return error;
    }
    /*
     * Print out the stats in no particular order
     */
    printf("File stats for file '%s'\n", f_filename);
    printf("Dev %d rdev %d nlink %d bsize %d\n",
       sb.sr_dev, sb.sr_rdev, sb.sr_nlink, sb.sr_bsize);
    printf("Size " ARG64D " nblocks " ARG64D "\n", sb.sr_size, sb.sr_nblocks);
    printf("Inode " ARG64D " uid %d gid %d mode 0%o \n",
           sb.sr_ino, sb.sr_uid, sb.sr_gid, sb.sr_mode);
    printf("atime: ");
    PrintTime(sb.sr_atim);
    printf(" mtime: ");
    PrintTime(sb.sr_mtim);
    printf(" ctime: ");
    PrintTime(sb.sr_ctim);
    printf("\n");
    return error;
}
/*
 * PURPOSE
 *  Stat file, new "plus" version
 */
int
StatFilePlus(
    int           f_fd,
    char    *f_filename)
{
    int                     error = 0;
    StatPlusReply_t         sb;
    char            *storestate;
    /*
     * Get the stats
     */
    error = CvApi_StatPlus(f_fd, &sb);
    if (error) {
       return error;
    }
    /*
     * Print out the stats in no particular order
     */
    printf("File stats for file '%s'\n", f_filename);
    printf("Dev %d nlink %d bsize %d\n",
       sb.sr_dev, sb.sr_nlink, sb.sr_bsize);

    switch(sb.sr_storestate) {

    case STORESTATE_ON_DISK_ONLY:
       storestate = "on disk only";
       break;
    case STORESTATE_ON_TAPE_ONLY:
       storestate = "on tape only";
       break;
    case STORESTATE_ON_DISK_AND_TAPE:
       storestate = "on disk and tape";
       break;
    default:
       storestate = "unknown";
       break;
    }
    printf("Store state: %d (%s)\n", sb.sr_storestate, storestate);
    printf("Size " ARG64D " nblocks " ARG64D "\n", sb.sr_size, sb.sr_nblocks);
    printf("Inode " ARG64D " uid %d gid %d mode 0%o \n",
           sb.sr_ino, sb.sr_uid, sb.sr_gid, sb.sr_mode);
    printf("atime: ");
    PrintTime(sb.sr_atim);
    printf(" mtime: ");
    PrintTime(sb.sr_mtim);
    printf(" ctime: ");
    PrintTime(sb.sr_ctim);
    printf("\n");
    return error;
}
```

```
/*
 * PURPOSE
 *  Stat VFS
 */
int
StatFs(
    int           f_fd,
    char    *f_filename)
{
    int                     error = 0;
    StatFsReply_t           sb;
    time_t           esecs;
    /*
     * Get the stats
     */
    error = CvApi_StatFs(f_fd, &sb);
    if (error) {
      return error;
    }
    /*
     * Print out the stats in no particular order
     */
    printf("FS stats for '%s'\n", f_filename);
    printf("options: 0x%x\n", sb.fr_options);
    if (sb.fr_options & FSOPTION_DMIG)
      printf("\tFSOPTION_DMIG\n");
    if (sb.fr_options & FSOPTION_QUOTAS)
      printf("\tFSOPTION_QUOTAS\n");
    if (sb.fr_options & FSOPTION_BRLS)
      printf("\tFSOPTION_BRLS\n");
    if (sb.fr_options & FSOPTION_GLOBALSU)
      printf("\tFSOPTION_GLOBALSU\n");
    if (sb.fr_options & FSOPTION_WINSEC)
      printf("\tFSOPTION_WINSEC\n");
    esecs = (time_t)(sb.fr_epoch / (uint64_t)1000000);
    printf("epoch: " ARG64X " - %s", sb.fr_epoch, ctime(&esecs));
    printf("block size: %d\n", sb.fr_blocksize);
    printf("total blocks: " ARG64D "\n", sb.fr_total_blocks);
    printf("blocks free: " ARG64D "\n", sb.fr_blocks_free);
    printf("inode stripe breadth: " ARG64D "\n", sb.fr_inode_stripe_width);
    printf("\n");
    return error;
}
/*
 * PURPOSE
 *  OpenStat file
 */
int
OpenStatFile(
    int      f_fd,
    char    *f_filename)
{
    int                  error = 0;
    OpenStatReply_t      sb;
    /*
     * Get the stats
     */
    error = CvApi_CvOpenStat(f_fd, &sb);
    if (error) {
        return error;
    }
    /*
     * Print out the stats in no particular order
     */
    printf("File Open_stats for file '%s'\n", f_filename);
    printf("RefCount: %u\n", sb.os_refcount);
    printf("OpenCount: %u\n", sb.os_opencount);
    if(sb.os_sharedread)
      printf("SharedRead TRUE\n");
    else
      printf("SharedRead FALSE\n");
    if(sb.os_sharedwrite)
        printf("SharedWrite TRUE\n");
    else
        printf("SharedWrite FALSE\n");
    printf("\n");
    return error;
}
/*
 * PURPOSE
 *  Get the physical location for a file given an offset
 */
int
GetPhysLoc(
    int               f_fd,
    char       *f_filename,
    uint64_t    f_offset)
{
    int                      error = 0;
    CvExternalExtent_t      *exp;
    PhysLocReply_t loc;
    /*
     * Get the location
     */
    error = CvApi_GetPhysLoc(f_fd, f_offset, &loc);
    if (error) {
      return error;
    }
    printf("Physical location for offset " ARG64X " in file '%s' \n",
          f_offset, f_filename);
    exp = &loc.pr_extent;
    printf("Extent: sg %d file relative base " ARG64X "\n" , exp->ex_sg,
          exp->ex_frbase);
    printf("         filesystem base " ARG64X, exp->ex_base);
    printf(" filesystem end " ARG64X , exp->ex_end);
    printf("\n");
    printf("SG breadth " ARG64X " depth " ARG64X "\n", loc.pr_breadth,
          loc.pr_depth);
    printf("volume offset " ARG64X " device relative blkoffset " ARG64X "\n",
          loc.pr_voloffset, loc.pr_blkoffset);
    printf("Device pseduo id %x\n", loc.pr_edev);
    return error;
}
/*
 * PURPOSE
 *  Dump out the extents for a file
 */
int
GetExtList(
    intf_fd)
{
    CvExternalExtent_t     *buf = NULL, *junkbuf = NULL, *exp;
    int                     allocsize;
    uint32_t         cnt, i, numbufs;
    int                     error = 0;
    uint64_t         offset;
    StatReply_t             sb;
    /*
     * Get the stats
     */
    error = CvApi_CvFstat(f_fd, &sb);
    if (error) {
       fprintf(stderr, "Can not stat file, error %d\n", error);
       return error;
    }
    /*
     * Alloc space for the extent buffer. We swag
     * and get 24 extents at a time.
     */
    numbufs = EXTMAX;
    allocsize = sizeof(CvExternalExtent_t) * numbufs;
    buf = malloc(allocsize);
    if (buf == NULL) {
       fprintf(stderr, "Can not alloc space for extent buffers\n");
       return 1;
    }
    memset(buf, 0, allocsize);
    if (ErrorFlag == 3) {   /* EINVAL */
       offset = 99999;
    }
    else  {
       offset = 0;
    }
    cnt = 0;
    do {
        if (ErrorFlag == 2)   /* EFAULT */
        {
            error = CvApi_GetExtList(f_fd, offset, &numbufs, junkbuf);
        }
        else {
            error = CvApi_GetExtList(f_fd, offset, &numbufs, buf);
        }
    if (error) {
        if (error != ENOENT) {
          fprintf(stderr,
               "Can not get extent list, offset " ARG64X " error %d\n",
               offset, error);
    }
    /*
     * ENOENT means no more extents
     */
            if ((error == ENOENT) && (ErrorFlag != 1))
            {
               error = 0;
```

```
                break;
            }
        }
        exp = buf;
        for (i=0; i<numbufs; i++) {

                printf("Extent %d frbase " ARG64X " sg 0x%x fsbase " ARG64X
                       " fsend " ARG64X " depth %x\n",
                  cnt++, exp->ex_frbase, exp->ex_sg, exp->ex_base, exp->ex_end,
                  exp->ex_depth);
                 /* Look for the next extent. */
                offset = exp->ex_frbase + ((exp->ex_end + 1) - exp->ex_base);
                exp++;
            }
        } while (error == 0);
        printf("%d total extents\n", cnt);
        return error;
}
/*
 * PURPOSE
 *  Punch a hole in a file
 */
int
PunchHole(
    int     f_fd,
    char    *f_filename,
    uint64_t f_offset,
    uint64_t f_nbytes)
{
    uint64_toffset, end, nblks, freed;
    int    error = 0;
    offset = f_offset;
    end = f_offset + f_nbytes;
    end--;            /* last byte, inclusive */
    if (Verbose) {
       printf("Punching a hole in '%s' from " ARG64X " to " ARG64X "\n",
           f_filename, f_offset, end);
    }
    error = CvApi_PunchHole(f_fd, &offset, &end, &nblks, &freed);
    if (error) {
       return error;
    }
    if (Verbose) {
       printf("Hole punched in '%s' from " ARG64X " to " ARG64X
              "blks freed " ARG64X " , blocks now in file " ARG64X "\n",
              f_filename, offset, end, freed, nblks);
    }
    return error;
}
```

```
/*
 * PURPOSE
 *  Load extents space
 */
int
LoadExtents(
    int     f_fd,
    char    *f_filename,
    uint64_t f_offset,
    uint64_t f_nbytes)
{
    uint64_toffset, nbytes;
    int    error = 0;
    offset = f_offset;
    nbytes = f_nbytes;
    if (Verbose) {
      printf("Loading extents in '%s' " ARG64D " bytes at offset "
             ARG64D "\n", f_filename, nbytes, offset);
    }
    error = CvApi_LoadExtents(f_fd, nbytes, offset);
    if (error) {
           return error;
    }
    if (Verbose) {
       printf("Loaded " ARG64D " bytes starting at offset " ARG64D "\n",
              nbytes, offset);
    }
    return error;
}
/*
 * PURPOSE
 *  Alloc space
 */
int
AllocSpace(
    int     f_fd,
    char    *f_filename,
    uint64_t f_offset,
    uint64_t f_nbytes,
    uint64_t f_affinity,
    uint32_t f_flags)
{
    uint64_toffset, nbytes;
    int    error = 0;
    offset = f_offset;
    nbytes = f_nbytes;
    if (Verbose) {
       printf("Allocating space in '%s' " ARG64D " bytes at offset "
              ARG64D "\n", f_filename, nbytes, offset);
    }
    error = CvApi_AllocSpace(f_fd, &nbytes, &offset, f_affinity,  f_flags);
    if (error) {
          return error;
    }
    if (Verbose) {
       printf("Allocated " ARG64D " bytes starting at offset " ARG64D "\n",
              nbytes, offset);
    }
    return error;
}
/*
 * PURPOSE
 *  Get PerfectFit status
 */
int
PerfectFitStatus(
    int     f_fd,
    char    *f_filename)
{
    int isperfectfit;
    int error;
    error = CvApi_GetPerfectFitStatus(f_fd, &isperfectfit);
    if (error) {
           return error;
    }
    if (Verbose) {
       printf("File %s does%s have the PerfectFit bit set.\n",
              f_filename, isperfectfit ? "" : " not");
    }
    return error;
}
/*
 * PURPOSE
 *  Get PerfectFit status
 */
int
TogglePerfectFitStatus(
    int          f_fd,
    char        *f_filename)
{
    int isperfectfit;
    int setperfectfit;
    int error;
    error = CvApi_GetPerfectFitStatus(f_fd, &isperfectfit);
    if (error) {
            return error;
    }
    if (Verbose) {
        printf("%s PerfectFit bit on file %s.\n",
                isperfectfit ? "Clearing" : "Setting", f_filename);
    }
    if(isperfectfit)
       setperfectfit = 0;
    else
       setperfectfit = 1;
    error = CvApi_SetPerfectFitStatus(f_fd, setperfectfit);
    return error;
}
/*
 * PURPOSE
 *  Get the affinity for a file
 */
int
GetAffinity(
    int     f_fd,
    char    *f_filename)
```

```
{
    uint8_t *cp;
    int     error = 0;
    uint64_t affinity;
    int     i;
      error = CvApi_GetAffinity(f_fd, &affinity);
    if (error == 0) {
       printf("Affinity for file '%s' is : ", f_filename);
       cp = (uint8_t *)&affinity;
       for (i=0; i<8; i++) {
           printf("%c", *cp++);
       }
       printf("\n");
    }
    return error;
}
/*
 * PURPOSE
 *  Set a file into concurrent write mode
 */
int
ConcWrite(
    int     f_fd,
    char    *f_filename)
{
    int   error = 0;
    error = CvApi_SetConcWrite(f_fd);
    if (error == 0) {
       printf("File '%s' is now in concurrent write mode. ", f_filename);
    }
    return error;
}
```

```
/*
 * PURPOSE
 *  Unset a file from concurrent write mode
 */
int
NoConcWrite(
    int      f_fd,
    char    *f_filename)
{
    int   error = 0;
    error = CvApi_ClearConcWrite(f_fd);
    if (error == 0) {
        printf("File '%s' is now out of concurrent write mode. ", f_filename);
    }
    return error;
}
/*
 * PURPOSE
 *   Print out info about a stripe group
 */
int
SgInfo(
    int     f_fd,
    int      f_sg)
{
    uint64_t    totblks;
    uint64_t    freeblks;
    uint32_t    breadth;
    uint32_t    depth;
    uint32_t    flags;
    uint32_t    bsize;
    char        *junkbuf = NULL;
    char        sgbuf[SG_NAMELEN];
    uint64_t    nativekeys[32];
    uint32_tkeycnt;
    int         error = 0;
    uint32_ti;

    if (ErrorFlag == 1) { /* ENOENT */
        f_sg = -1;
    }
    keycnt = 32;

    if (ErrorFlag == 2) { /* EFAULT */
         error = CvApi_GetSgInfo(f_fd, f_sg,
                        &totblks, &freeblks, &breadth,
                        &depth, &flags, &bsize, junkbuf,
                        nativekeys, &keycnt);
                            }
    else {
        error = CvApi_GetSgInfo(f_fd, f_sg,
                        &totblks, &freeblks, &breadth,
                        &depth, &flags, &bsize, sgbuf,
                        nativekeys, &keycnt);
    }
    if (error)
       return error;
    printf("Stripe group info for '%s' <%d>\n", sgbuf, f_sg);
    printf("Total blocks " ARG64D " (" ARG64X ")\n", totblks, totblks);
    printf("free blocks " ARG64D " (" ARG64X ")\n", freeblks, freeblks);
    printf("Breadth %x depth %x bsize %d (0x%x)\n",
       breadth, depth, bsize, bsize);
    printf("Flags (0x%x) ", flags);
    if (flags & SG_PART_VALID)
       printf(" valid " );
    if (flags & SG_PART_ONLINE)
       printf(" online " );
    if (flags & SG_PART_METADATA)
       printf(" metadata " );
    if (flags & SG_PART_JOURNAL)
       printf(" journal " );
    if (flags & SG_PART_EXCLUSIVE)
       printf(" exclusive " );
    printf("\n");
    for(i = 0; i < keycnt; i++) {
       char buf[9];
       memcpy(&buf, &nativekeys[i], sizeof(nativekeys[i]));
       buf[8] = '\0';
       printf("NativeKey[%d] = %s (%llx)\n", i, buf, nativekeys[i]);
    }
    return error;
}
/*
 * PURPOSE
 *   Print out disk info given a stripe group ordinal
 */
int
DiskInfo(
    int     f_fd,
    int      f_sg)
{
    CvDiskInfo_t dinfo[MAXDISKS];
    uint32_t ndisks;
    int error;
    ndisks = MAXDISKS;
    error = CvApi_GetDiskInfo(f_fd, f_sg, dinfo, (int *)&ndisks);
    if (!error) {
       int i, j;
       printf("Disk Info for Stripe group #%d:\n", f_sg);
       for(i = 0; i < (int)ndisks; i++) {
           printf("\tDisk #%d: name=\"%s\" vhsize=%u nameloc=%u secsize=%u 
serialnum=%s\n",
              i, dinfo[i].di_name, dinfo[i].di_vhsize,
              dinfo[i].di_nameloc, dinfo[i].di_sectorsize,
              dinfo[i].di_serialnum);
           for(j = 0; j < dinfo[i].di_npaths; j++) {
              printf("\t\tpath[%d]: blkdev=%s, rawdev=%s\n",
                   j,
                   dinfo[i].di_paths[j].d_bdev,
                   dinfo[i].di_paths[j].d_rdev);
           }
       }
    }
    return error;
}
/*
```

```
 * PURPOSE
 *   Print out ordinal of stripe group
 */
int
SgOrdinal(
    int  fd,
    char *sgname)
{
    uint32_t    sg;
    char        sgbuf[SG_NAMELEN];
    char        *junkbuf = NULL;
    int         error = 0;
    if ((ErrorFlag == 1) || (ErrorFlag == 3)) { /* ENOENT */
       sgbuf[0] = '\0';
    } else {
       strncpy(sgbuf, sgname, SG_NAMELEN);
       sgbuf[SG_NAMELEN - 1] = '\0';
    }
    if (ErrorFlag == 2) { /* EFAULT */
        error = CvApi_GetSgName(fd, SG_GETNUM, &sg, junkbuf);
    }
    else {
        error = CvApi_GetSgName(fd, SG_GETNUM, &sg, sgbuf);
    }
    if (error)
        return error;
    printf("Stripe group ordinal for '%s' : <%d>\n", sgname, sg);
    return error;
}
```

```
/*
 * PURPOSE
 *   Print out name of stripe group
 */
int
SgName(
    int       fd,
    uint32_t  sg)
{
    char        sgbuf[SG_NAMELEN];
    char        *junkbuf = NULL;
    int         error = 0;

    if (ErrorFlag == 1) { /* ENOENT */
       sgbuf[0] = '\0';
    }
    if ((ErrorFlag == 1) || (ErrorFlag == 3)) { /* ENOENT */
       sg = -1;                                /*   or   */
    }                                          /* EINVAL */
    if (ErrorFlag == 2) { /* EFAULT */
         error = CvApi_GetSgName(fd, SG_GETNAME, &sg, junkbuf);
    }
    else {
         error = CvApi_GetSgName(fd, SG_GETNAME, &sg, sgbuf);
    }
    if (error)
        return error;
    printf("Stripe group name for sg '%d' : <%s>\n", sg, sgbuf);
    return error;
}
/*
 * PURPOSE
 *   Set the read hole fail option for a file
 */
int
SetRDHoleFail(int  f_fd)
{
    int error = 0;
    error = CvApi_SetRdHoleFail(f_fd);
    if (error) {
        return error;
    }
    printf("File now in read hole fail mode\n");
    return error;
}
/*
 * PURPOSE
 *   Clear the read hole fail option for a file
 */
int
ClearRDHoleFail(int  f_fd)
{
    int error = 0;
    error = CvApi_ClearRdHoleFail(f_fd);
    if (error) {
        return error;
    }
    printf("File cleared from read hole fail mode\n");
    return error;
}
 /*
 * PURPOSE
 *   Retrieve the version info
 */
int
GetVersionInfo(int f_fd)
{
    int error = 0;
    VerInfoReply_tverinfo;
    error = CvApi_GetVerInfo(f_fd, &verinfo);
    if (error) {
        return error;
    }
    printf("Version string:\n %s \n", verinfo.vr_version);
    printf("Build string:\n %s \n", verinfo.vr_build);
    printf("Date string:\n %s \n", verinfo.vr_creationdate);
    printf("External API version: %d\n", verinfo.vr_apiversion);
    return error;
}
int
SetQuota(int f_fd, char *f_quotaname)
{
    int error;
    error = CvApi_SetQuota(f_fd, QUOTA_TYPE_USER, f_quotaname,
       (uint64_t)12000000, (uint64_t)10000000, 60);
    return(error);
}
int
GetQuota(int f_fd, char *f_quotaname)
{
    int error;
    GetQuotaReply_t qrep;
    error = CvApi_GetQuota(f_fd, QUOTA_TYPE_USER, f_quotaname, &qrep);
    if (!error) {
       printf("GetQuota results for %s:\n", f_quotaname);
       printf("hardlimit = %lld\n", qrep.gr_hardlimit);
       printf("softlimit = %lld\n", qrep.gr_softlimit);
       printf("cursize   = %lld\n", qrep.gr_cursize);
       printf("timelimit = %u\n", qrep.gr_timelimit);
       printf("timeout   = %u\n", qrep.gr_timeout);
    }
    return(error);
}
int
SetFileSize(int f_fd, uint64_t f_len)
{
    int error;
    error = CvApi_SetFileSize(f_fd, f_len);
    return(error);
}
/*
 * PURPOSE
 *  New Alloc space call
 */
int
VerifyAlloc(
    int          f_fd,
    char        *f_filename,
    uint64_t     f_offset,
    uint64_t     f_nbytes,
    uint32_t     f_flags)
{
    uint64_t    offset, nbytes;
    int         error = 0;
    offset = f_offset;
    nbytes = f_nbytes;
    if (Verbose) {
        printf("Allocating space in '%s' " ARG64D " bytes at offset "
                ARG64D " (flags 0x%X)\n", f_filename, nbytes, offset, f_flags);
    }
    error = CvApi_VerifyAlloc(f_fd, offset, nbytes, f_flags);
    if (error) {
            return error;
    }
    if (Verbose) {
        printf( ARG64D " bytes starting at offset " ARG64D " are allocated\n",
                nbytes, offset);
    }
    return error;
}

/*
 * PURPOSE
 *  Extract a value from the command line
 */
uint64_t
GetVal(
    char    *f_arg)
{
    uint64_tval = 0;

    val = strtoll(f_arg, NULL, 0);

    if (strrchr(f_arg, 'k'))
       val *= 1024;
    else if (strrchr(f_arg, 'm'))
       val *= (1024 * 1024);
    else if (strrchr(f_arg, 'g'))
       val *= (1024 * 1024 * 1024);
    return val;
}
int
main(argc, argv)
int    argc;
char   *argv[];
{
    char     *filename, *cp;
    char        *sgname = NULL;
    int     c, error = 0;
    int     fd;
    uint32_t flags = 0;
    uint32_t     sg = 0;
    uint64_t offset, nbytes, affinity, size;
    int     AllocFlag, ExtentFlag, PunchFlag, StatFlag, InfoFlag;
    int     GetAffFlag, SetAffFlag, PhysFlag, ConcFlag, NoConcFlag;
    int     SgNameFlag, SgOrdinalFlag, SetRDHoleFlag, ClearRDHoleFlag;
    int     GetQuotaFlag, SetQuotaFlag, DiskFlag, PerfectFlag;
    int     StatPlusFlag, SetFileSizeFlag, StatFsFlag;
    int     VerifyAllocFlag, SetPerfectFitFlag;
    char    *quotauser = NULL;
    int     offset_set, nbytes_set, size_set;
    int     openflags, VersFlag, LoadExtFlag, WaitFlag, OpenStatFlag;
    char    waitbuf[4];

    Progname = argv[0];
    if ((cp = strrchr(Progname, '/')) != NULL)
Progname = cp + 1;

    AllocFlag =  ExtentFlag = PunchFlag = StatFlag = InfoFlag = 0;
    PhysFlag = GetAffFlag = SetAffFlag = ConcFlag = NoConcFlag = 0;
    SgNameFlag =  SgOrdinalFlag = SetRDHoleFlag = ClearRDHoleFlag = 0;
    GetQuotaFlag = SetQuotaFlag = DiskFlag = PerfectFlag = SetFileSizeFlag = 0;
    VersFlag = LoadExtFlag = WaitFlag = OpenStatFlag = StatPlusFlag = 0;
    StatFsFlag = VerifyAllocFlag = SetPerfectFitFlag = 0;
    size_set = offset_set = nbytes_set = 0;
    affinity = offset = nbytes = 0;

    while ((c = getopt(argc, argv, "ABbCcD:EFfG:g:I:LOPpQ:q:RrSTa:lN:n:o:svVwxYyZz:"))
           != EOF)
    {
       switch (c) {
       case 'A':
           AllocFlag = 1;
           break;
       case 'B':
           StatFsFlag = 1;
           break;
       case 'b':
           flags |= ALLOC_SETSIZE;
           break;
       case 'C':
           ConcFlag = 1;
           break;
       case 'c':
           NoConcFlag = 1;
           break;
       case 'D':
           DiskFlag = 1;
           sg = strtoul(optarg, NULL, 0);
           break;
       case 'E':
           ExtentFlag = 1;
           break;
       case 'F':
           SetAffFlag = 1;
           break;
       case 'f':
           GetAffFlag = 1;
           break;
       case 'G':
           SgOrdinalFlag = 1;
           sg = strtoul(optarg, NULL, 0);
```

```
            break;
        case 'g':
            SgNameFlag = 1;
            sgname = optarg;
            break;
        case 'I':
            InfoFlag = 1;
            sg = strtoul(optarg, NULL, 0);
            break;
        case 'L':
            PhysFlag = 1;
            break;
        case 'O':
            OpenStatFlag = 1;
            break;
        case 'P':
            PunchFlag = 1;
            break;
        case 'p':
            PerfectFlag = 1;
            break;
        case 'R':
            SetRDHoleFlag = 1;
            break;
        case 'r':
            ClearRDHoleFlag = 1;
            break;
        case 'S':
            StatFlag = 1;
            break;
        case 'T':
            StatPlusFlag = 1;
            break;
        case 'a':
            /*
             * It is important to remember that affinities are
             * ASCII strings, so we have to special case '0'
             */
                if (strlen(optarg) > 8) {

                    fprintf(stderr, "The affinity type (-a) argument must be "
                    "eight characters or less.");
                Usage();
        exit(2);
            }
            if ((strlen(optarg) == 1) && (optarg[0] == '0')) {
        affinity = 0;
            } else {
                    strncpy((char*)&affinity, optarg, strlen(optarg));
            }
            flags |= ALLOC_AFFINITY;
            break;
        case 'l':
            flags |= ALLOC_LOAD_EXT;
            break;
        case 'n':
                nbytes = GetVal(optarg);
            nbytes_set = 1;
            break;
        case 'N':
            flags |= ALLOC_KEEPSIZE;
            nbytes = GetVal(optarg);
            nbytes_set = 1;
            break;
        case 'o':
                offset = GetVal(optarg);
            offset_set = 1;
            flags |= ALLOC_OFFSET;
            break;
        case 'Q':
            quotauser = optarg;
            SetQuotaFlag++;
            break;
        case 'q':
            quotauser = optarg;
            GetQuotaFlag++;
            break;
        case 's':
            flags |= ALLOC_STRIPE_ALIGN;
            break;
        case 't':
            flags |= ALLOC_PERFECTFIT;
            break;
        case 'v':
            Verbose = 1;
            break;
        case 'V':
            VersFlag = 1;
            break;
        case 'w':
                    WaitFlag = 1;
                    break;
                case 'x':
                    LoadExtFlag = 1;
                    break;
        case 'Y':
            VerifyAllocFlag = 1;
            break;
                case 'y':
                    SetPerfectFitFlag = 1;
                    break;
        case 'z':
                size = GetVal(optarg);
            size_set = 1;
            break;
        case 'Z':
            SetFileSizeFlag = 1;
            break;
        case '?':
        default:
            Usage();
            exit(2);
        }
    }

    if ((argc - optind) < 1) {
      fprintf(stderr, "Must supply filename\n");
      Usage();
      exit(3);
   }
   filename = argv[optind];

   if (AllocFlag || VerifyAllocFlag ||
       PunchFlag || SetAffFlag || ConcFlag) {
       openflags = O_RDWR | O_CREAT;
   } else {
       openflags = O_RDONLY;
   }
   if ((fd = open(filename, openflags, 0777)) < 0) {
       error = errno;
       fprintf(stderr, "Can not open filename %s, error '%s' (%d)\n",
            filename, strerror(error), error);
       exit( error);
   }
   if (AllocFlag) {
      /*
       * We used to check for 0 offset and size, but
       * now we default to 0 for the offset, and
       * if the user says alloc zero bytes, we need
       * to verify we return an error
       */
      error = AllocSpace(fd, filename, offset, nbytes, affinity,  flags);
      if (error) {
         fprintf(stderr, "Can not alloc " ARG64D " bytes in filename %s, "
                 " error '%s' (%d)\n", nbytes, filename,
                 strerror(error), error);
         exit(error);
   }
   }
   if (VerifyAllocFlag) {
       error = VerifyAlloc(fd, filename, offset, nbytes, flags);
       if (error) {
           fprintf(stderr, "Can not alloc " ARG64D " bytes in filename %s, "
                   " error '%s' (%d)\n", nbytes, filename,
                   strerror(error), error);
           exit(error);
       }
   }
   if (SetPerfectFitFlag) {
       error = TogglePerfectFitStatus(fd, filename);
       if (error) {
           fprintf(stderr, "Can not get PerfectFit status for filename %s, "
                   " error '%s' (%d)\n", filename,
                   strerror(error), error);
           exit(error);
       }
   }
   if (PerfectFlag) {
      error = PerfectFitStatus(fd, filename);
      if (error) {
          fprintf(stderr, "Can not get PerfectFit status for filename %s, "
               " error '%s' (%d)\n", filename,
                strerror(error), error);
          exit(error);
```

```
        }
    }
    if (PunchFlag) {
       if ((offset_set == 0) || (nbytes_set == 0)) {
           fprintf(stderr, "Must supply offset and number of bytes\n");
           Usage();
           exit(3);
       }
       error = PunchHole(fd, filename, offset, nbytes);
    }
    if (ExtentFlag) {
       error = GetExtList(fd);
       if (error) {
         fprintf(stderr, "Can not get extent list for file '%s'"
               " error '%s' (%d)\n", filename,
               strerror(error), error);
         exit(error);
       }
    }
    if (SetFileSizeFlag) {
       if (size_set == 0) {
           fprintf(stderr, "Must supply size\n");
           Usage();
           exit(3);
       }
       error = SetFileSize(fd, size);
       if (error) {
          fprintf(stderr, "Can not set size of "ARG64D" for file '%s'"
                " error '%s' (%d)\n", size, filename,
                    strerror(error), error);
          exit(error);
       }
    }
    if (StatFlag) {
       error = StatFile(fd, filename);
       if (error) {
           fprintf(stderr, "Can not stat file '%s'"
                " error '%s' (%d)\n", filename,
                    strerror(error), error);
           exit(error);
       }
    }
    if (StatPlusFlag) {
       error = StatFilePlus(fd, filename);
       if (error) {
           fprintf(stderr, "Can not do \"stat plus\" on file '%s'"
                 " error '%s' (%d)\n", filename,
                 strerror(error), error);
           exit(error);
       }
    }
    if (StatFsFlag) {
       error = StatFs(fd, filename);
       if (error) {
           fprintf(stderr, "Can not do StatFs on file '%s'"
                 " error '%s' (%d)\n", filename,
                 strerror(error), error);
           exit(error);
        }
    }
    if (OpenStatFlag) {
        error = OpenStatFile(fd, filename);
        if (error) {
            fprintf(stderr, "Can not open_stat file '%s'"
                    " error '%s' (%d)\n", filename,
                    strerror(error), error);
            exit(error);
        }
    }
    if (SetAffFlag) {
       if (Verbose) {
          printf("Setting affinity in '%s' to " ARG64X "\n",
             filename, affinity);
       }
       error = CvApi_SetAffinity(fd, affinity);
       if (error) {
          fprintf(stderr, "Can not set affinity for file '%s'"
                 " error '%s' (%d)\n", filename,
                 strerror(error), error);
          exit(error);
      }
    }
    if (GetAffFlag) {
       error = GetAffinity(fd, filename);
       if (error) {
           fprintf(stderr, "Can not get affinity for file '%s'"
                " error '%s' (%d)\n", filename,
                strerror(error), error);
           exit(error);
       }
    }
    if (PhysFlag) {
       if (offset_set == 0) {
           fprintf(stderr, "Must supply offset \n");
           Usage();
           exit(3);
       }
       error = GetPhysLoc(fd, filename, offset);
       if (error) {
           fprintf(stderr, "Can not get physical location for file '%s'"
                " error '%s' (%d)\n", filename,
                strerror(error), error);
           exit(error);
       }
    }
    if (ConcFlag) {
       error = ConcWrite(fd, filename);
       if (error) {
           fprintf(stderr, "Can not perform concurrent writes for file '%s'"
                " error '%s' (%d)\n", filename,
                strerror(error), error);
           exit(error);
       }
    }
    if (NoConcFlag) {
        error = NoConcWrite(fd, filename);
```

```
            if (error) {
                fprintf(stderr, "Can not clear concurrent writes for file '%s'"
                        " error '%s' (%d)\n", filename,
                        strerror(error), error);
                exit(error);
            }
        }
        if (SetQuotaFlag) {
           error = SetQuota(fd, quotauser);
           if (error) {
              fprintf(stderr, "Cannot set quota for user %s, error = %d\n",
                   quotauser, error);
              exit(error);
           } else {
              printf("Quota for %s successfully set\n", quotauser);
           }
        }
        if (GetQuotaFlag) {
           error = GetQuota(fd, quotauser);
           if (error) {
              fprintf(stderr, "Cannot get quota for user %s, error = %d\n",
                   quotauser, error);
              exit(error);
           }
        }
        if (InfoFlag) {
           error = SgInfo(fd, sg);
           if (error) {
              fprintf(stderr, "Can not get stripe group info for %d"
                   " error '%s' (%d)\n", sg,
                   strerror(error), error);
              exit(error);
           }
        }
        if (DiskFlag) {
           error = DiskInfo(fd, sg);
           if (error) {
              fprintf(stderr, "Can not get disk info for stripe group %d"
                   " error '%s' (%d)\n", sg,
                   strerror(error), error);
              exit(error);
           }
        }
        if (SgNameFlag) {
           error = SgOrdinal(fd, sgname);
           if (error) {
              fprintf(stderr, "Can not get stripe group ordinal for %s"
                   " error '%s' (%d)\n", sgname,
                   strerror(error), error);
              exit(error);
           }
        }
        if (SgOrdinalFlag) {
           error = SgName(fd, sg);
           if (error) {
              fprintf(stderr, "Can not get stripe group name for %d"
                   " error '%s' (%d)\n", sg,
                   strerror(error), error);
            exit(error);
        }
    }
    if (SetRDHoleFlag) {
        error = SetRDHoleFail(fd);
        if (error) {
            fprintf(stderr, "Can not set RD Hole Fail for file '%s'"
                    " error '%s' (%d)\n", filename,
                    strerror(error), error);
            exit(error);
        }
    }
    if (ClearRDHoleFlag) {
        error = ClearRDHoleFail(fd);
        if (error) {
            fprintf(stderr, "Can not clear RD Hole fail for file '%s'"
                    " error '%s' (%d)\n", filename,
                    strerror(error), error);
            exit(error);
        }
    }
    if (VersFlag) {
        error = GetVersionInfo(fd);
        if (error) {
            fprintf(stderr, "Can not retrieve version info "
                    " error '%s' (%d)\n",
                    strerror(error), error);
            exit(error);
        }
    }
    if (LoadExtFlag) {
       error = LoadExtents(fd, filename, offset, nbytes);
       if (error) {
           fprintf(stderr, "Can not load " ARG64D " bytes in filename %s, "
                " error '%s' (%d)\n", nbytes, filename,
                strerror(error), error);
           exit(error);
       }
    }
    if (WaitFlag) {
       printf("Waiting, press return to continue...\n");
       fflush(stdout);
       (void)fgets(waitbuf, sizeof(waitbuf), stdin);
    }
    return 0 ;
}
```

Appendix B

Storage Manager API Example

This appendix contains examples showing how to run a test program for the GetFileAttribute API. Included are two C++ files and two makefiles (one file each for Linux and Solaris platforms). The difference between the two C++ files is that one file is for running the test program via the XML interface. All examples in this appendix apply only to the Storage Manger APIs, not the File System APIs.

The makefiles are designed to work with the C++ files. Put the C++ source files and the makefile appropriate to your operating system into the same directory and then run the make command to create executable binaries. (The makefiles and C++ files can be found in the directory /usr/adic/SNAPI/examples.)

C++ Test Program Example

Following is a test program for the GetFileAttribute API.

```
///////////////////////////////////////////////////////////////////
// Copyright 2007 Quantum, Inc.
//
// NAME: GetFileAttribute.cc
//
#include <API.hh>

#include <sys/types.h>
#include <unistd.h>
#include <iostream>
#include <sstream>
#include <string>
```

```
using namespace std;
using namespace Quantum::SNAPI;

int
main()
{
   Status status;

   try
   {
      // Create object for requesting file attributes.
      GetFileAttribute getFileAttrReq("/snfs/testFile.dat");

      // Send request to the server and get overall request status code.
      // Note: This method may throw exceptions (see catch block below).
      status = getFileAttrReq.process();

      // Check the returned status code.
      if (status.getCode() != SUCCESS)
      {
         cout << "StatusCode: " << status.getCodeAsString() << endl;
         cout << "Description: " << status.getDescription() << endl;
         cout << "LocalStatus.StatusCode: " <<
                 getFileAttrReq.getLocalStatus().getCodeAsString() << endl;
         cout << "LocalStatus.Description: " <<
                 getFileAttrReq.getLocalStatus().getDescription() << endl;
      }
      else
      {
         // Get the requested data.
         FileInfo fileInfo = getFileAttrReq.getFileInfo();

         cout << "FileName:        " << fileInfo.getFileName() << endl;
         cout << "Location:        " << fileInfo.getLocationAsString() << endl;
        cout << "Existing Copies: " << fileInfo.getNumberOfExistingCopies() << endl;
         cout << "Target Copies:   " << fileInfo.getNumberOfTargetCopies() << endl;

         MediaList media = fileInfo.getMedia();

         for (int i=0; i<media.size(); i++)
         {
            cout << "Media ID:        " << media[i] << endl;
         }
      }
   }
   catch (SnException& exception)
   {
      switch (exception.getCode())
      {
         case SUBFAILURE:
         case FAILURE:
         case SYNTAXERROR:
         default:
         {
            cout << "Exception code:   " << exception.what() << endl;
            cout << "Exception detail: " << exception.getDetail() << endl;
            }
            break;
         }
      }
   }
   catch (...)
   {
      cout << "Caught unknown exception." << endl;
   }

   return status.getCode();
}
```

C++ XML Interface Test Program Example

Following is a test program for the GetFileAttribute API run from the XML interface.

```
/////////////////////////////////////////////////////////////////
// Copyright 2007 Quantum, Inc.
//
// NAME: GetFileAttributeXML.cc
//

#include <API.hh>

#include <sys/types.h>
#include <unistd.h>
#include <iostream>
#include <sstream>
#include <string>

using namespace std;
using namespace Quantum::SNAPI;

int
main()
{
   Status status;

   // Initialize input command in XML format.
   XML xmlIn("<?xml version=\"1.0\"?>"
             "<COMMAND name=\"GetFileAttribute\">"
             "<ARGUMENT name=\"fileName\" value=\"/snfs/testFile.dat\"/>"
             "</COMMAND>");

   XML xmlOut;

   try
   {
      // Perform the request.
      status = doXML(xmlIn, xmlOut);
   }
   catch (SnException& excptn)
   {
      cerr << "Exception code:   " << excptn.what() << endl;
```

```
    cerr << "Exception detail: " << excptn.getDetail() << endl;
    status = excptn.getCode();
  }

  // Stream the results to standard out.
  cout << xmlOut << endl;

  return status.getCode();
}
```

Makefile Example for Linux Platforms

Following is a makefile example for Linux platforms.

```
##################################################################
# Copyright 2005-2007 Quantum Corporation.
#
# This makefile works on Linux platforms. It looks for libraries and
# headers under /usr/adic/SNAPI/. Use it with the two source files
# listed below.
#
#   GetFileAttribute.cc
#   GetFileAttributeXML.cc
#
# Just rename this file as "makefile" and run "make all".
#
##################################################################
BASEPATH = /usr/adic/SNAPI

# Libraries for linking
LIBSSHARED = -L${BASEPATH}/lib -lsnapi -lpthread
LIBSSTATIC = -L${BASEPATH}/lib/static -lsnapi -lpthread

# Set this to the location of your g++ compiler.
CC = /usr/bin/g++

all: getFileAttribute.dynamic getFileAttribute.static\
     getFileAttributeXML.dynamic getFileAttributeXML.static

getFileAttribute.dynamic: getFileAttribute.o
     $(CC) -Wl,-rpath,${BASEPATH}/lib -o $@ getFileAttribute.o ${LIBSSHARED}

getFileAttribute.static: getFileAttribute.o
     $(CC) -o $@ getFileAttribute.o ${LIBSSTATIC}

getFileAttribute.o: GetFileAttribute.cc
     ${CC} -I${BASEPATH}/inc -c $< -o $@

getFileAttributeXML.dynamic: getFileAttributeXML.o
     $(CC) -Wl,-rpath,${BASEPATH}/lib -o $@ getFileAttributeXML.o ${LIBSSHARED}
```

```
getFileAttributeXML.static: getFileAttributeXML.o
     $(CC) -o $@ getFileAttributeXML.o ${LIBSSTATIC}

getFileAttributeXML.o: GetFileAttributeXML.cc
     ${CC} -I${BASEPATH}/inc -c $< -o $@

clean:
     /bin/rm -f getFileAttribute.o getFileAttribute.dynamic \
  getFileAttribute.static getFileAttributeXML.o getFileAttributeXML.dynamic \
  getFileAttributeXML.static
```

Makefile Example for Solaris Platforms

Following is a makefile example for Solaris platforms.

```
###################################################################
# Copyright 2005-2007 Quantum Corp.
#
# This makefile works on Solaris platforms. It looks for libraries and
# headers under /usr/adic/SNAPI/. Use it with the two source files
# listed below.
#
#   GetFileAttribute.cc
#   GetFileAttributeXML.cc
#
# Just rename this file as "makefile" and run "make all".
#
###################################################################
BASEPATH = /usr/adic/SNAPI

# Libraries for linking
LIBSSHARED = -L${BASEPATH}/lib -lsnapi -lxnet -lrt
LIBSSTATIC = -L${BASEPATH}/lib/static -lsnapi -lxnet -lrt

# Set this to the location of your g++ compiler.
CC = g++

all: getFileAttribute.dynamic getFileAttribute.static\
     getFileAttributeXML.dynamic getFileAttributeXML.static

getFileAttribute.dynamic: getFileAttribute.o
     $(CC) -R ${BASEPATH}/lib -o $@ getFileAttribute.o ${LIBSSHARED}

getFileAttribute.static: getFileAttribute.o
     $(CC) -R ${BASEPATH}/lib -o $@ getFileAttribute.o ${LIBSSTATIC}

getFileAttribute.o: GetFileAttribute.cc
     ${CC} -I${BASEPATH}/inc -c $? -o $@

getFileAttributeXML.dynamic: getFileAttributeXML.o
     $(CC) -R ${BASEPATH}/lib -o $@ getFileAttributeXML.o ${LIBSSHARED}

getFileAttributeXML.static: getFileAttributeXML.o
```

```
	$(CC) -R ${BASEPATH}/lib -o $@ getFileAttributeXML.o ${LIBSSTATIC}

getFileAttributeXML.o: GetFileAttributeXML.cc
	${CC} -I${BASEPATH}/inc -c $? -o $@

clean:
	/bin/rm -f getFileAttribute.o getFileAttribute.dynamic \
    getFileAttribute.static getFileAttributeXML.o getFileAttributeXML.dynamic \
    getFileAttributeXML.static
```