



Telestream



**C++ API**

# **Programmer's Guide**

**Release: 8.2**

**Revision: 1.2**

## Copyrights and Trademark Notices

Specifications subject to change without notice. Copyright © 2022 Telestream, LLC and its Affiliates. Telestream, CaptionMaker, Cerify, DIVA, Episode, Flip4Mac, FlipFactory, Flip Player, Gameshow, GraphicsFactory, Kumulate, Lightspeed, MetaFlip, Post Producer, Prism, ScreenFlow, Split-and-Stitch, Switch, Tempo, TrafficManager, Vantage, VOD Producer, and Wirecast are registered trademarks and Aurora, ContentAgent, Cricket, e-Captioning, Inspector, iQ, iVMS, iVMS ASM, MacCaption, Pipeline, Sentry, Surveyor, Vantage Cloud Port, CaptureVU, Cerify, FlexVU, PRISM, Sentry, Stay Genlock, Aurora, and Vidchecker are trademarks of Telestream, LLC and its Affiliates. All other trademarks are the property of their respective owners.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

# Contents

## Telestream Contact Information 8

## Preface 9

- Audience 9
- Documentation Accessibility 9
  - Access to Telestream Support 9
- Related Documents 9
- Document Updates 10

## Overview 11

- C++ API Overview 12
- DIVA Core Release Compatibility 13
- Alternate APIs 13
- New and Enhanced Features and Functionality 14
- New Terminology 15
- Managing Connections 16
  - Securing the API 16
    - Java API 16
    - C++ API 16
  - SSL (Secure Sockets Layer) and Authentication 16
- Compilers 18
  - Visual C++ Compiler on Windows 18
    - Supported Platforms 18
    - Supported Compilers 18
    - API Library Options 19
    - API Compilation 19
    - Initiator Sample Program API Usage 20
  - C++ Compiler on Linux 20
    - Supported Platforms 20
    - API Compilation 21
- Using the API in Multithreaded Applications 22
- Using Unicode Strings in the API 23

## Use and Operations 24

Session Management Commands	25
DIVA_getApiVersion	25
Synopsis	25
DIVA_SSL_initialize	25
Synopsis	25
DIVA_connect	25
Synopsis	25
Return Values	27
DIVA_disconnect	27
Synopsis	28
Multithreaded Applications	28
Return Values	28
Requests and Commands	29
DIVA_addGroup	29
Synopsis	29
Return Values	30
DIVA_archiveObject	31
Synopsis	31
Return Values	34
DIVA_associativeCopy	35
Synopsis	35
Return Values	36
DIVA_cancelRequest	38
Synopsis	38
Return Values	38
DIVA_changeRequestPriority	39
Synopsis	39
Return Values	40
DIVA_copyToGroup and DIVA_copy	41
Synopsis	41
Return Values	43
DIVA_copyToNewObject	44
Synopsis	44
Return Values	48
DIVA_deleteGroup	49
Synopsis	49
Return Values	50
DIVA_deleteInstance	51
Synopsis	51
Return Values	52
DIVA_deleteObject	53
Synopsis	54
Return Values	55
DIVA_ejectTape	56
Synopsis	56
Return Values	57

DIVA_enable_Automatic_Repack	58
Synopsis	58
Return Values	58
DIVA_getArchiveSystemInfo	59
Synopsis	59
Return Values	65
DIVA_getArrayList	66
Synopsis	66
Return Values	68
DIVA_getFinishedRequestList	69
Synopsis	69
Return Values	70
DIVA_getFilesAndFolders	71
Synopsis	71
Return Values	75
DIVA_getGroupsList	76
Synopsis	76
Return Values	77
DIVA_getObjectDetailsList	77
Synopsis	78
Return Values	87
Use with DIVA Connect	88
Use and Recommended Practices	89
Recommended Practices for Continuous Updates Notification Design Pattern (No Media Filter)	90
DIVA_getObjectInfo	93
Synopsis	93
Return Values	93
DIVA_getPartialRestoreRequestInfo	94
Synopsis	95
Return Values	95
DIVA_getRequestInfo	96
Synopsis	96
Return Values	100
Additional_Info	101
DIVA_getSourceDestinationList	102
Synopsis	102
Return Values	104
DIVA_getStoragePlanList	105
Synopsis	105
Return Values	105
DIVA_getTapeInfo	106
Synopsis	106
Return Values	107
DIVA_insertTape	108
Synopsis	108
Return Values	109
DIVA_linkObjects	110

Synopsis	110
Return Values	111
DIVA_lockObject	112
Synopsis	112
Return Values	112
DIVA_multipleRestoreObject	113
Synopsis	113
Return Values	115
DIVA_partialRestoreObject	117
Synopsis	121
Return Values	129
DIVA_release	131
Synopsis	131
Return Values	131
DIVA_require	132
Synopsis	133
Return Values	133
DIVA_restoreInstance	134
Synopsis	134
Return Values	137
DIVA_restoreObject	138
Synopsis	138
Return Values	141
DIVA_transcodeArchive	142
Synopsis	142
Return Values	145
DIVA_transferFiles	146
Synopsis	146
Return Values	148
DIVA_unlockObject	149
Synopsis	149
Return Values	149

## Using the API with DIVA Connect 151

What is DIVA Connect?	152
DIVA Core API Support	152
Input Parameters	153
Return Parameters	154
Return Codes	154
getObjectDetailsList Call	155

## Appendix 156

List of Authorized Special Characters in DIVA Core	157
Maximum Allowed Number of Characters	159
API Static Constant Values	160

## Glossary 166

# Telestream Contact Information

To obtain product information, technical support, or provide comments on this guide, contact us using our web site, email, or phone number as listed below.

Resource	Contact Information
DIVA Core Technical Support	<p>Web Site: <a href="https://www.telestream.net/telestream-support/diva/support.htm">https://www.telestream.net/telestream-support/diva/support.htm</a></p> <p>Depending on the problem severity, we will respond to your request within 24 business hours. For P1, we will respond within 1 hour. Please see the Maintenance &amp; Support Guide for these definitions.</p> <ul style="list-style-type: none"> <li>• Support hours for customers are Monday - Friday, 7am - 6pm local time.</li> <li>• P1 issues for customers are 24/7.</li> </ul>
Telestream, LLC	<p>Web Site: <a href="http://www.telestream.net">www.telestream.net</a></p> <p>Sales and Marketing Email: <a href="mailto:info@telestream.net">info@telestream.net</a></p> <p>Telestream, LLC 848 Gold Flat Road, Suite 1 Nevada City, CA USA 95959</p>
International Distributor Support	<p>Web Site: <a href="http://www.telestream.net">www.telestream.net</a></p> <p>See the Telestream Web site for your regional authorized Telestream distributor.</p>
Telestream Technical Writers	<p>Email: <a href="mailto:techwriter@telestream.net">techwriter@telestream.net</a></p> <p>Share comments about this or other Telestream documents.</p>

# Preface

This document contains a detailed description of the DIVA Core and DIVA Connect C++ API (Application Programmer's Interface).

## Audience

This document assists System Administrators and API Application Developers with development and deployment of applications interacting with DIVA Core and DIVA Connect.

## Documentation Accessibility

For information about Telestream's commitment to accessibility, visit the Telestream Support Portal located at:

<https://www.telestream.net/telestream-support/diva/support.htm>

## Access to Telestream Support

Telestream customers that have purchased support have access to electronic support through the Telestream Support Portal located at:

<https://www.telestream.net/telestream-support/diva/support.htm>

## Related Documents

For more information, see the DIVA Core documentation set for this release and the C++ Standard Template Library documentation located at:

<https://www.telestream.net/telestream-support/diva/support.htm>

# Document Updates

The following table identifies updates made to this document.

Date	Update
April 2022	Updated copyright information Updated book for 8.2 release Updated terminology to new standards (see the <a href="#">Overview</a> for updated terms)
July 2022	Migrated book to Telestream format.
September 2022	Updating terminology and new title page graphic.

# Overview

DIVA Core 8.2 supports interoperability among systems, helping to ensure long-term accessibility to valued content, and keeping up with evolving storage technologies.

The architecture of DIVA Core allows the integration of many different types of servers and technologies, for example Broadcast Video Servers, Storage Area Networks, and Enterprise Tape Managed Storage.

This chapter includes the following information:

## Topics:

- [C++ API Overview](#)
- [DIVA Core Release Compatibility](#)
- [Alternate APIs](#)
- [New and Enhanced Features and Functionality](#)
- [New Terminology](#)
- [Managing Connections](#)
- [Compilers](#)
- [Using the API in Multithreaded Applications](#)
- [Using Unicode Strings in the API](#)

# C++ API Overview

The main DIVA Core API is written in the C++ programming language. All of the definitions are contained in the include file named `DIVAapi.h`. In this document, parameters in function signatures are qualified by `IN` and `OUT` to specify whether the parameter is passed as an input or an output to the function. These qualifiers are not part of the C++ language and are only used for ease of readability. You must consider that these qualifiers are equivalent to the following macro definitions:

- `#define IN`
- `#define OUT`

In this document, the term structure identifies both C-like structures and classes which have only public data members and no function members<sup>1</sup>. Interfaces described in this document show only data members, not constructors or destructors.

The DIVA Core and DIVA Connect API use only standard data types provided directly by the C++ language, and the vector data type provided by the STL (Standard Template Library). For more information about the vector data type, refer to the STL documentation on the OTN.

---

**Note:** The API is not supported under the Solaris operating system.

---

DIVA Core 8.2 does not currently support the following API calls and features when used with complex Virtual Objects. Even if they are enabled, they will not be executed and no warnings will be generated.

- `VerifyFollowingArchive`
- `VerifyFollowingRestore`
- `DeleteOnSource`
- `DeleteFile`
- `getObjectListbyFileName`
- The `getObjectInfo` and `getObjectDetailsList` will only return a single file

When copying complex Virtual Objects to legacy-formatted media, the Copy request terminates returning a Can't write a complex object in Legacy format error, and an error code through the API.

---

1. The operators `new` and `delete` are not considered function members.

# DIVA Core Release Compatibility

DIVA Core and DIVA Connect are backward compatible with all earlier releases of the C++ API. Therefore, the C++ API 8.2.x is compatible with any DIVA Core release 8.0 and later.

Any new features added to DIVA Core after the release of the C++ API in use will not be available; the client system must be upgraded to the latest release to use all features.

## Alternate APIs

The API described in this document is for use with applications implemented in C++. However, the following additional APIs are available:

- **REST API:** DIVA Core exposes its functionality through a REST interface. It is self-contained in DIVA Core 8.0 and all future DIVA Core releases. In the 8.0 release, the API is used exclusively by the DIVA Core Web Application.

Telestream recommends using the REST API rather than the previous existing APIs. Although all previous APIs will remain available, the REST API offers new and enhance features.

See the DIVA Core REST API documentation set for more information.

- **Java API:** A set of libraries, samples and documentation for use with applications implemented in Java. See the Java API Readme for Java API document location information.
- **DIVA Enterprise Connect and Web Services API:** DIVA Enterprise Connect is a standards-based Web Service API implemented on the Oracle WebLogic Suite. DIVA Enterprise Connect interacts with the DIVA Core and DIVA Connect systems, acting as a web service binding for the API.

DIVA Enterprise Connect includes the DIVA Web Services API, which is a set of interface definition files and documentation for universal use by applications supporting Web Services communications.

See the DIVA Enterprise Connect documentation set for more information.

## New and Enhanced Features and Functionality

The following new and enhanced features and functionality are included in DIVA Core 8.2:

- The Source Media Priority is reported in the `getArrayList` and `getGroupsList` calls.
- The storage options are reported in the `getArrayList` call, and storage options for each disk instance is returned from the `getObjectInfo` and `getObjectDetailsList` calls.
- Secure Socket Layer authentication has been included in DIVA Core 8.2. See [SSL \(Secure Sockets Layer\) and Authentication](#) for more information.
- A new call named `DIVA_SSL_initialize` has been added to set the environment for secure communications with the Core Manager service. In DIVA Core 8.2 you must make this call before calling `DIVA_connect` or the connection will fail. See [DIVA\\_SSL\\_initialize](#) for more information on this call.

# New Terminology

The following terminology has been updated to reflect standardization efforts across all DIVA and Kumulate applications. There will be some variations in the documentation compared to the interface until everything is switched over to the new terminology; the documentation uses the new terms wherever possible.

- Running Requests are now called Jobs
- Request History is now called Job History
- Libraries are now called Managed Storage
- Datahub is now called Actor
- Proxyhub is now called Proxy Actor
- DIVA Core and DIVA Manager are now called DIVA Core / Core / Core Manager
- Category is now called Collection
- Source/Destination is now called Unmanaged Storage Repository
- Storage Repository is now called Managed Storage Repository
- Object is now called Virtual Object
- Group is now called Tape Group
- Link is now called Storage Link
- Storage Plan Manager is now called Storage Policy Manager
- Drop Folder Monitor (DFM) is now called Watch Folder Monitor (WFM)
- DIVA Command and Control Panel are now called System Management App
- DIVA Analytics and DIVAProtect are now called Analytics App

# Managing Connections

The number of connections to the Core Manager is limited by the Core Manager and set in the Core Manager configuration file. The default configuration is two hundred connections, which includes GUI connections and all API connections. Once the configured limit is reached, the API will not allow additional connections to be created. See the `manager.conf` file for additional information.

---

**Caution:** It is recommended that a new connection not be created for each request or command sent to the Core Manager. Whenever possible allow the connection to remain open for the lifetime of the session, or application.

---

## Securing the API

The following sections describe securing communications when using one of the available DIVA Core APIs. The JAVA and C++ Initiators use the default keys and certificates file in the `%DIVA_API_HOME%/Program/security` folder when connecting to the Core Manager.

The Core Manager Service is backward compatible with earlier versions of the JAVA, C++, Web Services APIs, DIVA Enterprise Connect 1.0, and DIVA Connect 2.2 establishing connections over regular sockets. The DIVA Core 8.2 (and later) Java and C++ API releases can establish Core Manager communications using secure, or insecure, sockets. Secure communications are only supported by the Core Manager.

The Core Manager Service supports both secure and insecure communication ports simultaneously. The default secure port is `tcp/8000`, and the default insecure port is `tcp/9000`.

### Java API

See the Java API documentation for information on the new methods added to the `SessionParameters` Class for secure communications. Also see the Java API Readme for the location of the full Java API documentation (delivered with the API).

### C++ API

The C++ API includes a new call named `DIVA_SSL_initialize` added to set the environment for secure communication with the Core Manager Service. You must call `DIVA_SSL_initialize` before calling `DIVA_connect` with DIVA Core 8.2, otherwise the `DIVA_connect` call will fail.

## SSL (Secure Sockets Layer) and Authentication

DIVA Core consist of services in Java and C++. The format in how certificates and keys are represented are different in each. DIVA Core has the keys and certificates for JAVA

services in a Java Keystore file, and in PEM (Privacy Enhanced Mail) format files for the C++ services.

The Core Manager can simultaneously support two communications ports - one secure, and one insecure. The default secure port number is 8000 and the insecure default port number is 9000.

All internal DIVA Core 8.2 services (System Management App, DBBackup, Migration Utility, Actor, SPM, WFM, SNMP, Robot Manager, RDTU, and Migration Services) can only connect to secure ports. The System Management App will report an SSL Handshake Timeout if you attempt to connect to the non-secure port. Clients using the Java or C++ API are allowed to connect to either port.

The following is a relative snippet from the Core Manager configuration file:

```
# Port number on which the DIVA Core is waiting for incoming
connections.
# Note: If you are using a Sony Managed Storage and plan to execute
the DIVA Core
# on the same machine as the PetaSite Controller (PSC) software, be
aware
# that the PSC server uses the 9000 port and that this cannot be
modified.
# In that situation, you have to use a different port for the DIVA
Core.
# This same warning applies to FlipFactory which uses ports 9000
and 9001.
# The default value is 9000.
DIVAMANAGER_PORT=9000

# Secure port number on which the DIVA Core is waiting for incoming
connections.
# The default value is 8000.
DIVAMANAGER_SECURE_PORT=8000
```

A new folder called %DIVA\_API\_HOME%/security is added to the API installation structure as follows:

```
%DIVA_API_HOME%
  security
    conf
```

The conf folder contains the SSLSettings.conf file that is used to configure the SSL handshake timeout.

# Compilers

The following sections cover the supported API compilers.

## Visual C++ Compiler on Windows

These section describe using the Visual C++ compiler on the Windows operating system.

### Supported Platforms

There are two separate variants of the API for Windows: 32-bit and 64-bit. The 32-bit model can be used on both x86 and x64 platforms. However, the 64-bit variant requires a 64-bit platform. The API for Windows is supported on the following Windows releases:

- Microsoft Windows Server 2012
- Microsoft Windows Server 2012 R2
- Microsoft Windows Server 2008
- Microsoft Windows Server 2008 x64
- Microsoft Windows Server 2008 R2

### Supported Compilers

The API is compiled and tested using the following compilers:

#### Microsoft Visual C++ 2010 (Release 10)

Including Microsoft Platform SDK 7.0a (April 2010)

#### Microsoft Visual C++ 2012 (Release 11)

Including Microsoft Platform SDK 7.1A (November 2012)

#### Microsoft Visual C++ 2013 (Release 13)

Including Microsoft Platform SDK 8.0A (October 2013)

## API Library Options

The API is delivered with both static and dynamic libraries. Each library is available in a standard format with debug support and Unicode compatibility. The different options may be found in the following build directories:

### Static Library

Static\_Release

### Static Library with Debug Support

Static\_Debug

### Dynamic Library

Dynamic\_Release

### Dynamic Library with Debug Support

Dynamic\_Debug

## API Compilation

Choose the 8 Bytes setting for the Strict Member Alignment option under C/C++ Code Generation in the project settings.

The following list identifies the library path that corresponds to each run time library. The run time library is normally changed automatically depending upon the selected build configuration.

### Multithreaded

Static\_Release

### Debug Multithreaded

Static\_Debug

### Multithreaded DLL

Dynamic\_Release

### Debug Multithreaded DLL

Dynamic\_Debug

You must include the DIVA Core API.lib file, or the path to this file, in the link settings (see [Initiator Sample Program API Usage](#)). The API can be included in an application compiled with either the IDE or a script using the command line compiler.

Once your application is built, you must either add the folder where the API.dll file is located to your PATH environment variable, or copy the API.dll file into the folder containing your executable file.

## Initiator Sample Program API Usage

The Initiator program is included with the API and is an example of the API usage. This is a command line program that uses the API to send requests and get data from DIVA Core. Use the following project files to view the compiler settings and build the program:

### Visual C++ .NET (Release 10)

doc\CpplInitiator\InitiatorVc100.vcxproj(64-bit API)

### Visual C++ .NET (Release 11)

doc\CpplInitiator\InitiatorVc110.vcxproj(64-bit API)

### Visual C++ .NET (Release 12)

doc\CpplInitiator\InitiatorVc120.vcxproj(64-bit API)

## C++ Compiler on Linux

These sections describe using the C++ compiler on the Linux operating system platform.

### Supported Platforms

The API for Linux is supported on Oracle Linux. The API was built with the C++ compiler and Oracle Solaris Studio library. The following list identifies the supported CC release and Oracle Solaris Studio library release.

- Oracle Linux 7 x86\_64 (64-bit) operating system
- Oracle Solaris Studio 12.4 library

The following command returns the CC release level:

```
[root@LinuxBuildVM /]# CC -V  
CC: Sun C++ 5.13 Linux_i386 2014/10/20
```

The API may work on other Linux platforms; however it is only officially validated in the environment described here. Support for the older release previously built on SuSe Linux 9.0 was discontinued starting with DIVA Core 8.0. For all development projects, use of the latest release is strongly recommended.

## API Compilation

The API is delivered with the `x86_64_Release_unicode` shared library for the Linux platform. The release is located in the `DIVA/api/lib` directory. The library is built in Release Mode and does not contain symbolic information.

Header files that may be required to compile an application with the API libraries are delivered in the `DIVA/api/include` directory.

For reference, a sample application is provided in the `DIVA/api/doc/CPPIInitiator` directory along with its source code. The Visual Studio project file for Microsoft Windows, and sample makefiles for Linux platforms are also provided. Refer to the sample makefiles provided in the `DIVA/api/doc/CPPIInitiator` directory for platform-specific compiler and linker options.

## Using the API in Multithreaded Applications

The API supports using multiple threads concurrently with the following restrictions (see the related function's specific documentation for additional information):

- The `DIVA_connect()` and `DIVA_disconnect()` functions share the same critical section. Although multiple simultaneous connections are supported, they must be opened and closed one at a time.
- The `init`, `get`, and `close` functions used to retrieve list information (Virtual Objects List or Virtual Objects Tape Information List) also use a [Critical Section](#) to prevent concurrent threads reinitializing the list while another thread is currently reading it. The critical section is entered when the list is initialized and left when the list is closed. There are two separate critical sections, one for each type of list.
- All of the other DIVA Core functions may be called simultaneously by different threads. For example, one thread can call the `DIVA_archiveObject()` function while another one is calling `DIVA_getArchiveSystemInfo()`.

# Using Unicode Strings in the API

The API (and other DIVA Core components) support wide character strings. Only 64-bit Unicode is delivered with the API. You must define the `_UNICODE` constant before including the `DIVAapi.h` header file to be able to use the `wchar_t` and `wstring`.

In addition, the application must be linked with one of the Unicode releases in the library (for example, in `lib/Release_Unicode`).

Defining, or not defining, the `_UNICODE` macro will change the implementation of the `DIVA_STRING` and `DIVA_CHAR` types.

The `_T` macro is recommended when working with static strings:

## Example:

```
_T("Hello")
```

Type	_UNICODE Not Defined	_UNICODE Defined
DIVA_STRING	string	wstring
DIVA_CHAR	char	wchar_t

# Use and Operations

This chapter discusses connection management, requests, and commands, and includes the following information:

**Topics:**

- [Session Management Commands](#)
- [Requests and Commands](#)

# Session Management Commands

The following three sections describe the commands used to control the session connection.

## DIVA\_getApiVersion

Returns the string pointed to by version of the major part of the release number.

### Synopsis

```
#include "DIVAapi.h"

void DIVA_getApiVersion (
    OUT DIVA_STRING    *version
);
```

### version

Points to a string that contains the major part of the release for this API.

## DIVA\_SSL\_initialize

The DIVA\_SSL\_initialize call sets the environment for secure communication with the Core Manager Service. You must call DIVA\_SSL\_initialize before calling DIVA\_connect with Core 8.2, otherwise the DIVA\_connect call will not establish a secure connection.

### Synopsis

```
DIVA_STATUS DIVA_SPEC DIVA_SSL_initialize(
    DIVA_STRING KeyPath, // [in] Full path of the Key file contain
the private key and certificate in PEM format.
    DIVA_STRING TrustStorePath, // [in] Full path of the file
containing Trust certificates in PEM format.
    DIVA_STRING KeyPassword // [in] Password for the private key
)
```

## DIVA\_connect

Opens a connection with the Core Manager. All of the other API functions are only available when a connection is open. A connection cannot be opened if another connection is already open. To open a new connection, the previous one must be explicitly closed by calling DIVA\_disconnect().

### Synopsis

```
#include "DIVAapi.h"

DIVA_STATUS DIVA_connect (
    IN string managerAddress,
    IN int portNumber
```

```
);
DIVA_STATUS DIVA_connect (
    IN string managerAddress,
    IN int portNumber,
    IN string userName,
    IN string password,
    IN string applicationName
);
DIVA_STATUS DIVA_connect (
    IN string managerAddress,
    IN int portNumber,
    IN string userName,
    IN string password,
    IN string applicationName
    IN string userInfo
);
```

### **managerAddress**

The IP address of the Core Manager.

### **portNumber**

The port on which the Core Manager is listening. The default port is pointed to by the constant value DIVA\_MGER\_DEFAULT\_PORT.

### **userName**

The user name.

### **password**

The password associated with the user name.

### **applicationName**

The name of the application.

### **userInfo**

User specific and specified information.

### **Multithreaded Applications:**

A critical section protects both the DIVA\_connect() and DIVA\_disconnect() functions. If a thread is already in the process of closing the connection to the Core Manager, other threads must wait until the running thread exits the DIVA\_connect() function before being able to open or close the connection.

## Return Values

One of the following `DIVA_STATUS` constants defined in `DIVAapi.h`:

### **DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

### **DIVA\_ERR\_SYSTEM\_IDLE**

The DIVA Core system is no longer able to accept connections and queries.

### **DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

### **DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the `DIVA_API_TIMEOUT` variable and equals one hundred-eighty (180) seconds by default.

### **DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

### **DIVA\_ERR\_INTERNAL**

An internal error was detected by the Core Manager or by the API.

### **DIVA\_ERR\_NO\_ARCHIVE\_SYSTEM**

There was a problem when establishing a connection with the specified DIVA Core system.

### **DIVA\_ERR\_WRONG\_VERSION**

The release levels of the API and the Core Manager are not compatible.

### **DIVA\_ERR\_ALREADY\_CONNECTED**

A connection is already open.

Also see [DIVA\\_connect](#).

## DIVA\_disconnect

Closes a connection with the Core Manager. When a connection is closed, only the `DIVA_connect()` function can be called. If no connection is currently open, this function has no effect and returns `DIVA_OK`.

## Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_disconnect ()
```

## Multithreaded Applications

A critical section protects both the `DIVA_connect()` and `DIVA_disconnect()` functions. If a thread is already in the process of closing the connection to the Core Manager, other threads must wait until the running thread exits the `DIVA_disconnect()` function before being able to open or close the connection.

## Return Values

One of the following `DIVA_STATUS` constants defined in `DIVAapi.h`:

### **DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

### **DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

### **DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the `DIVA_API_TIMEOUT` variable and equals one hundred-eighty (180) seconds by default.

### **DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

### **DIVA\_ERR\_INTERNAL**

An internal error was detected by the Core Manager or by the API.

### **DIVA\_ERR\_DISCONNECTING**

There was a problem when disconnecting. The connection is considered to still be open.

Also see [DIVA\\_connect](#).

# Requests and Commands

The following sections discuss all of the available API commands for use in your application.

## DIVA\_addGroup

This function adds a new Tape Group.

### Synopsis

```
#include "DIVAapi.h"

DIVA_STATUS DIVA_addGroup (
    IN DIVA_STRING      groupName,
    IN int               associatedSet,
    IN DIVA_STRING      comment,
    IN bool              toBeRepacked,
    IN bool              worstFitEnabled,
    IN int               worstFitRepackTapes,
    IN int               mediaFormatId
);
```

### groupName

The name of the Tape Group to be added.

### associatedSet

The set of tapes to associate with the new Tape Group. This value must be strictly greater than zero.

### comment

A text description of the new Tape Group.

### toBeRepacked

If true, tapes belonging to this Tape Group are eligible for automatic repacking.

### worstFitEnabled

If true, Worst Fit Policy (access speed optimization) will apply.

### worstFitRepackTapes

The number of tapes reserved for Worst Fit Repacking.

## **mediaFormatId**

The data format to be used by the tapes assigned to this Tape Group. The value can be `DIVA_MEDIA_FORMAT_LEGACY` or `DIVA_MEDIA_FORMAT_AXF`. See information on media formats in the [Glossary](#).

## **Return Values**

One of the following `DIVA_STATUS` constants defined in `DIVAapi.h`:

### **DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

### **DIVA\_ERR\_NOT\_CONNECTED**

No connection is open.

### **DIVA\_ERR\_SYSTEM\_IDLE**

The DIVA Core system can no longer accept connections and queries.

### **DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

### **DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. You set the timeout duration using the `DIVA_API_TIMEOUT` variable. The default value is one hundred-eighty (180) seconds.

### **DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

### **DIVA\_ERR\_INTERNAL**

An internal error was detected by the Core Manager or by the API.

### **DIVA\_ERR\_INVALID\_PARAMETER**

A parameter value was not understood by the Core Manager.

### **DIVA\_ERR\_GROUP\_ALREADY\_EXISTS**

The specified Tape Group already exists.

## DIVA\_archiveObject

Submits an archive request to the Core Manager. This function returns as soon as the Core Manager accepts the request. The application must call the function `DIVA_getRequestInfo()` to check that the operation completed successfully.

### Synopsis

```
#include "DIVAapi.h"

DIVA_STATUS DIVA_archiveObject (
    IN DIVA_STRING      objectName,
    IN DIVA_STRING      objectCollection,
    IN DIVA_STRING      source,
    IN DIVA_STRING      mediaName,
    IN DIVA_STRING      filePathRoot,
    IN vector<DIVA_STRING> filenamesList,
    IN DIVA_ARCHIVE_QOS qualityOfService,
    IN int              priorityLevel,
    IN DIVA_STRING      comments,
    IN DIVA_STRING      archiveOptions,
    OUT int             requestNumber
);
```

### objectName

The name of the Virtual Object to be archived.

### objectCollection

The Collection of the Virtual Object to be archived.

### source

The name of the Source Server (for example, the video server, browsing server, and so on). This name must be known to the DIVA Core configuration description.

### mediaName

The tape group or disk array where the Virtual Object is to be saved. The media may be defined as follows:

#### Name (of the Tape Group or Array)

Provide the tape group or disk array name as defined in the configuration. The Virtual Object is saved to the specified media and assigned to the default SP (Storage Plan).

#### SP Name

Provide a SP Name (Storage Plan Name) as defined in the configuration. The Virtual Object will be assigned to the specified Storage Plan and saved to the default media specified.

**Both of the above (Name and SP Name)**

The Virtual Object is saved to the specified media as in Name, and assigned to the specified Storage Plan as in SP Name. The Name and the SP Name must be separated by the & delimiter (this is configurable).

When this parameter is a null string, the default group of tapes called DEFAULT is used. Complex Virtual Objects can only be saved to AXF media types.

**filePathRoot**

The root folder for the files specified by the filenamesList parameter.

**filenamesList**

List of file path names relative to the folder specified by the filePathRoot parameter. Path names must be absolute names when the filePathRoot is null.

The following is for DIVA Core releases 7.1.2 and later only:

If the -gcinfilelist option is specified the Genuine Checksum is included with a colon separator between the file name and the GC value as follows:

```
test1.txt:a6f62b73f5a9bf380d32f062f2d71cbc  
test2.txt:96bf41e4600666ff69fc908575c0319
```

**qualityOfService**

One of the following codes executes the request using the specified QOS:

**DIVA\_QOS\_DEFAULT**

Archiving is performed according to the default Quality Of Service (currently direct and cache for archive operations).

**DIVA\_QOS\_CACHE\_ONLY**

Use cache archive only.

**DIVA\_QOS\_DIRECT\_ONLY**

Use direct archive only - no disk instance is created.

**DIVA\_QOS\_CACHE\_AND\_DIRECT**

Use cache archive if available, or direct archive if cache archive is not available.

**DIVA\_QOS\_DIRECT\_AND\_CACHE**

Use direct archive if available, or cache archive if direct archive is not available.

Additional and optional services are available. To request those services, use a logical OR between the previously documented Quality Of Service parameter and the following constant:

**DIVA\_ARCHIVE\_SERVICE\_DELETE\_ON\_SOURCE**

Delete source files when the tape migration is done. Available for local Source Servers, disk Source Servers, and standard FTP Source Servers. This feature is not available for complex Virtual Objects.

**priorityLevel**

The priority level for this request. The priorityLevel can be in the range zero to one hundred, or the value `DIVA_DEFAULT_REQUEST_PRIORITY`. The value zero is the lowest priority and one hundred the highest priority.

There are six predefined values as follows:

- `DIVA_REQUEST_PRIORITY_MIN`
- `DIVA_REQUEST_PRIORITY_LOW`
- `DIVA_REQUEST_PRIORITY_NORMAL`
- `DIVA_REQUEST_PRIORITY_HIGH`
- `DIVA_REQUEST_PRIORITY_MAX`
- `DIVA_DEFAULT_REQUEST_PRIORITY`

When the `DIVA_DEFAULT_REQUEST_PRIORITY` value is used, the Core Manager uses the default priority defined in the Core Manager configuration for the request.

Using a value either outside of the range of zero to one hundred, or predefined values yields a `DIVA_ERR_INVALID_PARAMETER` error.

**comments**

Optional information describing the Virtual Object. This can be a null string.

**archiveOptions**

Additional options for performing the transfer of data from the Source Server to DIVA Core. These options supersede any options specified in the DIVA Core configuration database. Currently the possible values for archiveOptions are as follows:

**Null string**

A null string specifies no options.

**-delete\_on\_source**

Executes a delete on the Source Server after an archive request completes.

**-r**

Using `-r` specifies that every name in `filenamesList` that refers to a folder must be scanned recursively. This also applies when `FilePathRoot` is specified and an asterisk designates the files to be archived. This option can be used when archiving from a local Source Server or from a standard FTP Server.

**-login**

A user name and password is required to log in to some Source Servers. This option obsoletes the -gateway option from earlier releases.

**-pass**

The password used with -login.

The following is for DIVA Core releases 7.1.2 and later only:

**-gcinfilelist [gcType]**

Specifies that GC (Genuine Checksum) values are included in the file names list. The value of gcType must match the Core Manager's default checksum type as specified in the DIVA Core configuration (MD5 by default). The GC values are then used to verify the transfer from the Source Server.

**requestNumber**

The request number assigned to this request. This number is used for querying the status or canceling the request.

**Return Values**

One of the following DIVA\_STATUS constants defined in DIVAapi.h:

**DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

**DIVA\_ERR\_NOT\_CONNECTED**

No connection is open.

**DIVA\_ERR\_SYSTEM\_IDLE**

DIVA Core can no longer accept connections and queries.

**DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

**DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. You set the timeout duration using the DIVA\_API\_TIMEOUT variable. The default value is one hundred-eighty (180) seconds.

**DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

## DIVA\_ERR\_INTERNAL

The Core Manager or API detected an internal error.

## DIVA\_ERR\_INVALID\_PARAMETER

The Core Manager did not understand a parameter value.

## DIVA\_ERR\_CANNOT\_ACCEPT\_MORE\_REQUESTS

The count of simultaneous requests reached the maximum allowed value. You set this variable in the manager.conf configuration file. The default value is three hundred.

## DIVA\_ERR\_GROUP\_DOESNT\_EXIST

The specified tape group or disk array does not exist.

## DIVA\_ERR\_SOURCE\_OR\_DESTINATION\_DOESNT\_EXIST

The specified Server is unknown by the DIVA Core system.

## DIVA\_associativeCopy

Submits a request for creating new instances in the Tape Group (specified by group). DIVA Core guarantees that these instances are stored sequentially on tapes:

- The request is completed only when every Virtual Object is copied to the same tape.
- In the case of drive or tape failure during a write operation, instances currently written are erased and the request is retried once.
- The choice of the tape to be used for the copy follows the policy used for the archive operation (written tapes with enough remaining size regardless of optimizations).
- Associative Copy does not span tapes - the request terminates (and is retried once) instead of spanning. The request terminates if the sum of the size of the Virtual Objects to copy exceeds the capacity of every individual tape present in the Managed Storage.

## Synopsis

```
#include "DIVAapi.h"
```

```

DIVA_STATUS DIVA_associativeCopy (
IN  vector<DIVA_OBJECT_SUMMARY>    *objectsInfo,
IN  DIVA_STRING                     groupName,
IN  int                             priorityLevel,
IN  DIVA_STRING                     options,
OUT int                             *requestNumber
);

```

## **objectsInfo**

A pointer to a list of Virtual Objects defined by a name and Collection pair.

## **groupName**

The name of the Tape Group where the new instance will be located. Complex Virtual Objects can only be saved to AXF media types. Associative Copy to a disk array is not available.

## **priorityLevel**

The level of priority for this request. The priorityLevel can be in the range zero to one hundred or the value `DIVA_DEFAULT_REQUEST_PRIORITY`. The value zero is the lowest priority and one hundred is the highest priority.

There are six predefined values as follows:

- `DIVA_REQUEST_PRIORITY_MIN`
- `DIVA_REQUEST_PRIORITY_LOW`
- `DIVA_REQUEST_PRIORITY_NORMAL`
- `DIVA_REQUEST_PRIORITY_HIGH`
- `DIVA_REQUEST_PRIORITY_MAX`
- `DIVA_DEFAULT_REQUEST_PRIORITY`

When the `DIVA_DEFAULT_REQUEST_PRIORITY` value is used, the Core Manager uses the default priority defined in the Core Manager configuration for the request.

Using a value either outside of the range of zero to one hundred or predefined values yields a `DIVA_ERR_INVALID_PARAMETER` error.

## **options**

An optional string attribute for specifying additional parameters to the request.

## **requestNumber**

A number identifying the request.

## **Return Values**

One of the following `DIVA_STATUS` constants defined in `DIVAapi.h`:

### **DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

### **DIVA\_ERR\_NOT\_CONNECTED**

No connection is open.

**DIVA\_ERR\_SYSTEM\_IDLE**

The DIVA Core system is no longer able to accept connections and queries.

**DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

**DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the `DIVA_API_TIMEOUT` variable and equals one hundred-eighty (180) seconds by default.

**DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

**DIVA\_ERR\_INTERNAL**

An internal error was detected by the Core Manager or by the API.

**DIVA\_ERR\_INVALID\_PARAMETER**

A parameter value was not understood by the Core Manager.

**DIVA\_ERR\_CANNOT\_ACCEPT\_MORE\_REQUESTS**

The count of simultaneous requests reached the maximum allowed value. This variable is set in the `manager.conf` configuration file and the default value is three hundred.

**DIVA\_ERR\_OBJECT\_DOESNT\_EXIST**

The specified Virtual Object does not exist in the DIVA Core database.

**DIVA\_ERR\_SEVERAL\_OBJECTS**

More than one Virtual Object with the specified name exists in the DIVA Core database.

**DIVA\_ERR\_OBJECT\_OFFLINE**

No available instance for this Virtual Object. Tape instances are ejected and no Actor could provide a disk instance.

**DIVA\_ERR\_GROUP\_DOESNT\_EXIST**

The specified tape group or disk array does not exist.

## DIVA\_ERR\_OBJECT\_IN\_USE

The Virtual Object is currently in use (*being archived, restored, deleted, and so on*).

## DIVA\_ERR\_OBJECT\_PARTIALLY\_DELETED

The specified Virtual Object has instances that are partially deleted.

Also see [DIVA\\_archiveObject](#) and [DIVA\\_copyToGroup and DIVA\\_copy](#).

# DIVA\_cancelRequest

Submits a Cancel operation to the Core Manager. This function returns as soon as the Core Manager accepts the operation. The application must call the function `DIVA_getRequestInfo()` to check that the operation was successful.

## Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_cancelRequest (  
IN int      requestNumber,  
IN DIVA_STRING options  
);
```

## requestNumber

A number identifying the request to be canceled. This parameter can be set to `DIVA_ALL_REQUESTS` to cancel all cancelable requests.

## options

An optional string attribute for specifying additional parameters to the request.

## Return Values

One of the following `DIVA_STATUS` constants defined in `DIVAapi.h`:

## DIVA\_OK

The request was correctly submitted and accepted by the Core Manager.

## DIVA\_ERR\_NOT\_CONNECTED

No connection is open.

## DIVA\_ERR\_SYSTEM\_IDLE

The DIVA Core system cannot accept connections and queries.

## DIVA\_ERR\_BROKEN\_CONNECTION

The connection with the Core Manager was broken.

## DIVA\_ERR\_TIMEOUT

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the `DIVA_API_TIMEOUT` variable and equals one hundred-eighty (180) seconds by default.

## DIVA\_ERR\_UNKNOWN

An unknown status was received from the Core Manager.

## DIVA\_ERR\_INTERNAL

An internal error was detected by the Core Manager or by the API.

## DIVA\_ERR\_NO\_SUCH\_REQUEST

The requestNumber identifies no request.

Also see [DIVA\\_getRequestInfo](#).

# DIVA\_changeRequestPriority

Submits a Change Request Priority request to the Core Manager. This function returns as soon as the Core Manager accepts the request. The application must call the `DIVA_getRequestInfo()` function to check that the operation was successful.

## Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_changeRequestPriority (  
IN int      requestNumber,  
IN int      priorityLevel,  
IN DIVA_STRING passThruOptions  
);
```

## requestNumber

A number identifying the request to be changed.

## priorityLevel

The level of priority for this request. The priorityLevel can be in the range zero to one hundred. The value zero is the lowest priority and one hundred is the highest priority.

There are five predefined values as follows:

- `DIVA_REQUEST_PRIORITY_MIN`

- `DIVA_REQUEST_PRIORITY_LOW`
- `DIVA_REQUEST_PRIORITY_NORMAL`
- `DIVA_REQUEST_PRIORITY_HIGH`
- `DIVA_REQUEST_PRIORITY_MAX`

The use of `DIVA_DEFAULT_REQUEST_PRIORITY` is not allowed with this function.

Using a value either outside of the range of zero to one hundred or predefined values yields a `DIVA_ERR_INVALID_PARAMETER` error.

## **passThruOptions**

An optional string attribute for specifying additional parameters to the request.

## **Return Values**

One of the following `DIVA_STATUS` constants defined in `DIVAapi.h`:

### **DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

### **DIVA\_ERR\_NOT\_CONNECTED**

No connection is open.

### **DIVA\_ERR\_SYSTEM\_IDLE**

The DIVA Core system cannot accept connections and queries.

### **DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

### **DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the `DIVA_API_TIMEOUT` variable and equals one hundred-eighty (180) seconds by default.

### **DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

### **DIVA\_ERR\_INTERNAL**

An internal error was detected by the Core Manager or by the API.

## DIVA\_ERR\_NO\_SUCH\_REQUEST

The requestNumber identifies no request.

## DIVA\_ERR\_INVALID\_PARAMETER

A parameter value has not been understood by the Core Manager.

Also see [DIVA\\_getRequestInfo](#).

## DIVA\_copyToGroup and DIVA\_copy

Submits a New Instance Creation request on the media specified by mediaName to the Core Manager, and the Core Manager chooses the appropriate instance to be created. This function returns as soon as the Core Manager accepts the request. The application must call the DIVA\_getRequestInfo() function to check that the operation was successful.

The request will fail if the requested Virtual Object is on media that is not available. The Media Names (tape barcodes and disk names) that contain instances of the Virtual Object will be included in the additionalInfo field of the DIVA\_getRequestInfo() response.

A tape group may contain two instances of the same Virtual Object. In this case, DIVA Core will terminate the request if both instances cannot be written on two different tapes. A disk array can contain two instances of the same Virtual Object; however DIVA Core will terminate the request if the new instance cannot be written on a different disk. There can be a maximum of only one instance of each Virtual Object per disk or tape.

## Synopsis

DIVA\_copyToGroup is a public alias to DIVA\_copy and performs the same functionality.

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_copy (
IN DIVA_STRING      objectName,
IN DIVA_STRING      CollectionName,
IN int              instanceID,
IN DIVA_STRING      mediaName,
IN int              priorityLevel,
IN DIVA_STRING      options,
OUT int             *requestNumber
);
```

```
DIVA_STATUS DIVA_copyToGroup (
IN DIVA_STRING      objectName,
IN DIVA_STRING      CollectionName,
IN int              instanceID,
IN DIVA_STRING      mediaName,
IN int              priorityLevel,
IN DIVA_STRING      options,
```

```
OUT int *requestNumber  
);
```

## objectName

The name of the Virtual Object to be copied.

## objectCollection

The Collection assigned to the Virtual Object when it was archived. This parameter can be a null string; however this may result in an error if several Virtual Objects have the same name.

## instanceID

The instance's identifier. DIVA\_ANY\_INSTANCE as the Instance ID means that DIVA Core will choose the appropriate instance.

## mediaName

The media (tape group or disk array) where the new instance will be located.

## priorityLevel

The level of priority for this request. The priorityLevel can be in the range zero to one hundred or the value DIVA\_DEFAULT\_REQUEST\_PRIORITY. The value zero is the lowest priority and one hundred is the highest priority.

There are six predefined values as follows:

- DIVA\_REQUEST\_PRIORITY\_MIN
- DIVA\_REQUEST\_PRIORITY\_LOW
- DIVA\_REQUEST\_PRIORITY\_NORMAL
- DIVA\_REQUEST\_PRIORITY\_HIGH
- DIVA\_REQUEST\_PRIORITY\_MAX
- DIVA\_DEFAULT\_REQUEST\_PRIORITY

When the DIVA\_DEFAULT\_REQUEST\_PRIORITY value is used, the Core Manager uses the default priority defined in the Core Manager configuration for the request.

Using a value either outside of the range of zero to one hundred or predefined values yields a DIVA\_ERR\_INVALID\_PARAMETER error.

## options

An optional string attribute for specifying additional parameters to the request.

## requestNumber

A number identifying the request to be changed.

## Return Values

One of the following DIVA\_STATUS constants defined in DIVAapi.h:

### **DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

### **DIVA\_ERR\_NOT\_CONNECTED**

No connection is open.

### **DIVA\_ERR\_SYSTEM\_IDLE**

The DIVA Core system cannot accept connections and queries.

### **DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

### **DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the DIVA\_API\_TIMEOUT variable and equals one hundred-eighty (180) seconds by default.

### **DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

### **DIVA\_ERR\_INTERNAL**

An internal error was detected by the Core Manager or by the API.

### **DIVA\_ERR\_INVALID\_PARAMETER**

A parameter value has not been understood by the Core Manager.

### **DIVA\_ERR\_CANNOT\_ACCEPT\_MORE\_REQUESTS**

The count of simultaneous requests has reached the maximum allowed value. This variable is set in the manager.conf configuration file. The default is three hundred.

### **DIVA\_ERR\_OBJECT\_DOESNT\_EXIST**

The specified Virtual Object does not exist in the DIVA Core database.

### **DIVA\_ERR\_INSTANCE\_DOESNT\_EXIST**

The instance specified for restoring this Virtual Object does not exist.

## DIVA\_ERR\_SEVERAL\_OBJECTS

More than one Virtual Object with the specified name exists in the DIVA Core database.

## DIVA\_ERR\_OBJECT\_OFFLINE

No available instance for this Virtual Object. Tape instances are ejected and no Actor could provide a disk instance.

## DIVA\_ERR\_INSTANCE\_OFFLINE

The instance specified for restoring this Virtual Object is ejected, or the Actor owning the specified disk instance is not available.

## DIVA\_ERR\_GROUP\_DOESNT\_EXIST

The specified Tape Group does not exist.

## DIVA\_ERR\_OBJECT\_IN\_USE

The Virtual Object is currently in use (being archived, restored, deleted, and so on).

## DIVA\_ERR\_OBJECT\_PARTIALLY\_DELETED

The specified Virtual Object has instances that are partially deleted.

Also see [DIVA\\_archiveObject](#).

# DIVA\_copyToNewObject

Submits a request for copying an archived Virtual Object to a new Virtual Object, with another name or Collection, to the Core Manager. The Core Manager chooses the appropriate instance as the source of the copy. This function returns as soon as the Core Manager accepts the request. The application must call the `DIVA_getRequestInfo()` function to check that the operation was successful.

The request will fail if the requested Virtual Object is on an unavailable media. The media names (tape barcodes and disk names) that contain instances of the Virtual Object will be included in the `additionalInfo` field of the `DIVA_getRequestInfo()` response.

All types of transfers (disk to disk, disk to tape, tape to disk, and tape to tape) are supported.

## Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_copyToNewObject (
IN const DIVA::ObjectInstanceDescriptor    &source,
IN const DIVA::ObjectInstanceDescriptor    &target,
```

```
IN const DIVA::RequestAttributes      &attrs,
IN DIVA STRING                       options,
OUT int                             *requestNumber
);
```

## source

The description of the Virtual Object or Virtual Object instance to be copied:

### source.objectName

The Source Server Virtual Object name (required).

### source.objectCollection

The Source Server Virtual Object Collection (required).

### source.group

The Source Server Virtual Object instance tape group or disk array. This is optional, however if specified DIVA Core will use this instance as the Source Server.

### source.instanceID

The Instance ID of the Source Server Virtual Object instance. This is optional, however if specified and not equal to DIVA\_ANY\_INSTANCE, DIVA Core will use this instance as the Source Server. The source.group parameter will be ignored if source.instanceID is specified.

If both source.group and source.instanceID are omitted, DIVA Core will use the most suitable instance (that provides the best performance) as a source.

## target

The description of the target Virtual Object:

### target.objectName

The target Virtual Object name (required).

### target.objectCollection

The target Virtual Object Collection (required).

### target.group

See the following paragraph.

### target.instanceID

This call ignores this value.

Either the Virtual Object name or Collection (or both) must be different from name or Collection of the Source Server Virtual Object. The request will fail if the target Virtual Object already exists in DIVA Core.

## attrs

The request attributes:

### attrs.priority

The request priority (optional). If this is not explicitly set the default value is used. Possible values are zero (lowest) to one hundred (highest).

### attrs.qos

QOS (Quality of Service) is not applicable to this request and this call ignores this value.

### attrs.comments

The target Virtual Object's comments (optional). If no value is specified the Source Server Virtual Object's comments are inherited.

### attrs.options

This request has no additional options and this call ignores this value.

## requestNumber

The number identifying the request that is returned by DIVA Core.

```
DIVA_STATUS DIVA_copyToNewObject (
IN const DIVA_STRING      &objectName,
IN const DIVA_STRING      &objectCollection,
IN const DIVA_STRING      &objectMedia,
IN int                    objectInstanceID,
IN const DIVA_STRING      &newObjectName,
IN const DIVA_STRING      &newObjectCollection,
IN const DIVA_STRING      &newObjectInstanceMedia,
IN const DIVA_STRING      &comments,
IN int                    priorityLevel,
IN DIVA_STRING            options,
OUT int                   *requestNumber
);
```

## objectName

The name of the Source Server Virtual Object.

## objectCollection

The Collection of the Source Server Virtual Object.

## objectMedia

The tape group or disk array of the Source Server Virtual Object instance (optional). If specified (not empty), DIVA Core will use this instance as a Source Server. Complex Virtual Objects can only be saved to AXF formatted media types.

## **objectInstanceID**

The Instance ID of the Source Server Virtual Object instance (optional). If specified and not equal to DIVA\_ANY\_INSTANCE, DIVA Core will use this instance as the Source Server. This call ignores the ObjectMedia parameter if an instanceID value is specified.

If both objectMedia and instanceID are not specified, DIVA Core will use the most suitable instance (providing the best performance) as the Source Server.

## **newObjectName**

The target Virtual Object name.

## **newObjectCollection**

The target Virtual Object Collection. Either the Virtual Object name or Collection (or both) must be different from name or Collection of the Source Server Virtual Object.

This request will fail if the target Virtual Object already exists in DIVA Core.

## **newObjectInstanceMedia**

The tape group or disk array where the Virtual Object will be saved. The media may be defined as follows:

### **Name (of the Tape Group or Array)**

Provide the tape group or disk array name as defined in the configuration. The Virtual Object is saved to the specified media and assigned to the default SP (Storage Plan).

### **SP Name**

Provide a SP Name (Storage Plan Name) as defined in the configuration. The Virtual Object will be saved to the default media specified in the Storage Plan and assigned to the specified Storage Plan.

### **Both of the above (Name and SP Name)**

The Virtual Object is saved to the specified media as in Name above. The Virtual Object is assigned to the specified SP as in SP Name above. The Name and the SP Name must be separated by the & delimiter (this is configurable).

## **comments**

Optional information describing the target Virtual Object. If left empty the Source Server Virtual Object comments are inherited.

## **priorityLevel**

Level of priority for this request. The possible values can be in the range zero to one hundred, and the DIVA\_DEFAULT\_REQUEST\_PRIORITY (use default request priority).

## **options**

Optional string attribute for specifying additional parameters to the request.

## **requestNumber**

The request number assigned to this request by DIVA Core.

## **Return Values**

One of these DIVA\_STATUS constants defined in DIVAapi.h:

### **DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

### **DIVA\_ERR\_NOT\_CONNECTED**

No connection is open.

### **DIVA\_ERR\_SYSTEM\_IDLE**

The DIVA Core system cannot accept connections and queries.

### **DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

### **DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the DIVA\_API\_TIMEOUT variable and equals one hundred-eighty (180) seconds by default.

### **DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

### **DIVA\_ERR\_INTERNAL**

An internal error was detected by the Core Manager or by the API.

### **DIVA\_ERR\_INVALID\_PARAMETER**

A parameter value was not understood by the Core Manager.

### **DIVA\_ERR\_CANNOT\_ACCEPT\_MORE\_REQUESTS**

The count of simultaneous requests has reached the maximum allowed value. This variable is set in the manager.conf configuration file. The default value is three hundred.

**DIVA\_ERR\_OBJECT\_DOESNT\_EXIST**

The specified Virtual Object does not exist in the DIVA Core database.

**DIVA\_ERR\_INSTANCE\_DOESNT\_EXIST**

The instance specified for restoring this Virtual Object does not exist.

**DIVA\_ERR\_SEVERAL\_OBJECTS**

More than one Virtual Object with the specified name exists in the DIVA Core database.

**DIVA\_ERR\_OBJECT\_OFFLINE**

No available instance for this Virtual Object. Tape instances are ejected and no Actor could provide a disk instance.

**DIVA\_ERR\_INSTANCE\_OFFLINE**

The instance specified for restoring this Virtual Object is ejected, or the Actor owning the specified disk instance is not available.

**DIVA\_ERR\_GROUP\_DOESNT\_EXIST**

The specified Tape Group does not exist.

**DIVA\_ERR\_OBJECT\_IN\_USE**

The Virtual Object is currently in use (being archived, restored, deleted, and so on).

**DIVA\_ERR\_OBJECT\_PARTIALLY\_DELETED**

The specified Virtual Object has instances that are partially deleted.

Also see [DIVA\\_copyToGroup](#) and [DIVA\\_copy](#).

## DIVA\_deleteGroup

Deletes the Tape Group passed as an argument. You can only delete a Tape Group when the Tape Group is empty.

**Synopsis**

```
#include "DIVAapi.h"
```

```
IN DIVA_STRING          groupName  
DIVA_STATUS DIVA_deleteGroup (  
);
```

**groupName**

The name of the Tape Group to be deleted.

**Return Values**

One of the following DIVA\_STATUS constants defined in DIVAapi.h.

**DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

**DIVA\_ERR\_NOT\_CONNECTED**

No connection is open.

**DIVA\_ERR\_SYSTEM\_IDLE**

The DIVA Core system cannot accept connections and queries.

**DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

**DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the DIVA\_API\_TIMEOUT variable and equals one hundred-eighty (180) seconds by default.

**DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

**DIVA\_ERR\_INTERNAL**

An internal error was detected by the Core Manager or by the API.

**DIVA\_ERR\_INVALID\_PARAMETER**

A parameter value was not understood by the Core Manager.

**DIVA\_ERR\_GROUP\_DOESNT\_EXIST**

The specified Tape Group does not exist.

**DIVA\_ERR\_GROUP\_IN\_USE**

The Tape Group contains at least one Virtual Object currently in use (being archived, restored, deleted, and so on).

## DIVA\_deleteInstance

Deletes a Virtual Object instance.

### Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_deleteInstance (  
IN DIVA_STRING      objectName,  
IN DIVA_STRING      CollectionName,  
IN int              instanceID,  
IN int              priorityLevel,  
IN DIVA_STRING      options,  
OUT int             *requestNumber  
);
```

```
DIVA_STATUS DIVA_deleteInstance (  
IN DIVA_STRING      objectName,  
IN DIVA_STRING      CollectionName,  
IN DIVA_STRING      mediaName,  
IN int              priorityLevel,  
IN DIVA_STRING      options,  
OUT int             *requestNumber  
);
```

### objectName

The name of the Virtual Object to be deleted.

### objectCollection

The Collection assigned to the Virtual Object when it was archived. This parameter can be a null string, however this may result in an error if several Virtual Objects have the same name.

### instanceID

The instance's identifier

### mediaName

Defines the media that contains the valid instance. If the instanceID is -1, the instance on the media will be deleted. If the media contains 2 or more instances, only one of the instances will be deleted.

### priorityLevel

The level of priority for this request. The priorityLevel can be in the range zero to one hundred or the value DIVA\_DEFAULT\_REQUEST\_PRIORITY. The value zero is the lowest priority and one hundred is the highest priority.

There are six predefined values as follows:

- `DIVA_REQUEST_PRIORITY_MIN`
- `DIVA_REQUEST_PRIORITY_LOW`
- `DIVA_REQUEST_PRIORITY_NORMAL`
- `DIVA_REQUEST_PRIORITY_HIGH`
- `DIVA_REQUEST_PRIORITY_MAX`
- `DIVA_DEFAULT_REQUEST_PRIORITY`

When the `DIVA_DEFAULT_REQUEST_PRIORITY` value is used, the Core Manager uses the default priority defined in the Core Manager configuration for the request.

Using a value either outside of the range of zero to one hundred or predefined values yields a `DIVA_ERR_INVALID_PARAMETER` error.

## options

An optional string attribute for specifying additional parameters to the request.

## requestNumber

A number identifying the request.

## Return Values

One of the following `DIVA_STATUS` constants defined in `DIVAapi.h`:

### **`DIVA_OK`**

The request was correctly submitted and accepted by the Core Manager.

### **`DIVA_ERR_NOT_CONNECTED`**

No connection is open.

### **`DIVA_ERR_SYSTEM_IDLE`**

The DIVA Core system cannot accept connections and queries.

### **`DIVA_ERR_BROKEN_CONNECTION`**

The connection with the Core Manager was broken.

### **`DIVA_ERR_TIMEOUT`**

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the `DIVA_API_TIMEOUT` variable and equals one hundred-eighty (180) seconds by default.

**DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

**DIVA\_ERR\_INTERNAL**

An internal error was detected by the Core Manager or by the API.

**DIVA\_ERR\_INVALID\_PARAMETER**

A parameter value was not understood by the Core Manager.

**DIVA\_ERR\_CANNOT\_ACCEPT\_MORE\_REQUESTS**

The count of simultaneous requests has reached the maximum allowed value. This variable is set in the `manager.conf` configuration file. The default value is three hundred.

**DIVA\_ERR\_OBJECT\_DOESNT\_EXIST**

The specified Virtual Object does not exist in the DIVA Core database.

**DIVA\_ERR\_SEVERAL\_OBJECTS**

More than one Virtual Object with the specified name exists in the DIVA Core database.

**DIVA\_ERR\_INSTANCE\_DOESNT\_EXIST**

The specified instance does not exist.

**DIVA\_ERR\_LAST\_INSTANCE**

`DIVA_deleteObject()` must be used to delete the last instance of an Virtual Object.

**DIVA\_ERR\_OBJECT\_IN\_USE**

The Virtual Object is currently in use (being archived, restored, deleted, and so on).

**DIVA\_ERR\_OBJECT\_PARTIALLY\_DELETED**

The specified Virtual Object has instances that are partially deleted.

See also [\*DIVA\\_getObjectInfo\*](#).

## **DIVA\_deleteObject**

Submits an Virtual Object Delete Request to the Core Manager. The Core Manager deletes every instance of the Virtual Object. This function returns as soon as the Core Manager accepts the request. To check that the operation was successful the application must call the function `DIVA_getRequestInfo()`.

## Synopsis

```
#include "DIVAapi.h"

DIVA_STATUS DIVA_deleteObject (
    IN DIVA_STRING      objectName,
    IN DIVA_STRING      objectCollection,
    IN int               priorityLevel,
    IN DIVA_STRING      options,
    OUT int              *requestNumber
);
```

### objectName

The name of the Virtual Object to be deleted.

### objectCollection

The Collection assigned to the Virtual Object when it was archived. This parameter can be a null string, however this may result in an error if several Virtual Objects have the same name.

### priorityLevel

The level of priority for this request. The priorityLevel can be in the range zero to one hundred or the value DIVA\_DEFAULT\_REQUEST\_PRIORITY. The value zero is the lowest priority and one hundred is the highest priority.

There are six predefined values as follows:

- DIVA\_REQUEST\_PRIORITY\_MIN
- DIVA\_REQUEST\_PRIORITY\_LOW
- DIVA\_REQUEST\_PRIORITY\_NORMAL
- DIVA\_REQUEST\_PRIORITY\_HIGH
- DIVA\_REQUEST\_PRIORITY\_MAX
- DIVA\_DEFAULT\_REQUEST\_PRIORITY

When the DIVA\_DEFAULT\_REQUEST\_PRIORITY value is used, the Core Manager uses the default priority defined in the Core Manager configuration for the request.

Using a value either outside of the range of zero to one hundred or predefined values yields a DIVA\_ERR\_INVALID\_PARAMETER error.

### options

An optional string attribute for specifying additional parameters to the request.

### requestNumber

A number identifying the request.

## Return Values

One of the following DIVA\_STATUS constants defined in DIVAapi.h:

### **DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

### **DIVA\_ERR\_NOT\_CONNECTED**

No connection is open.

### **DIVA\_ERR\_SYSTEM\_IDLE**

The DIVA Core system cannot accept connections and queries.

### **DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

### **DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the DIVA\_API\_TIMEOUT variable and equals one hundred-eighty (180) seconds by default.

### **DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

### **DIVA\_ERR\_INTERNAL**

An internal error was detected by the Core Manager or by the API.

### **DIVA\_ERR\_INVALID\_PARAMETER**

A parameter value was not understood by the Core Manager.

### **DIVA\_ERR\_CANNOT\_ACCEPT\_MORE\_REQUESTS**

The count of simultaneous requests has reached the maximum allowed value. This variable is set in the manager.conf configuration file. The default value is three hundred.

### **DIVA\_ERR\_OBJECT\_DOESNT\_EXIST**

The specified Virtual Object does not exist in the DIVA Core database.

## DIVA\_ERR\_SEVERAL\_OBJECTS

More than one Virtual Object with the specified name exists in the DIVA Core database.

## DIVA\_ERR\_OBJECT\_IN\_USE

The Virtual Object is currently in use (being archived, restored, deleted, and so on).

## DIVA\_ERR\_OBJECT\_BEING\_ARCHIVED

The specified Virtual Object does not exist in the DIVA Core database, but it is currently being archived.

See also [DIVA\\_getRequestInfo](#) and [DIVA\\_deleteInstance](#).

# DIVA\_ejectTape

Submits an Eject Request to DIVA Core. The request completes when the specified tapes are outside of the Managed Storage.

If at least one of the tapes does not exist, is already ejected, or currently in use by another request, the DIVA\_ERR\_INVALID\_PARAMETER status code is returned and no tapes are ejected.

## Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_ejectTape (
IN  vector<DIVA_STRING>    *vsnList,
IN  bool                   release
IN  DIVA_STRING            comment,
IN  int                   priorityLevel,
OUT int                   *requestNumber
);
```

## vsnList

List of VSNs for identifying the tapes to be ejected.

## release

When true, perform a DIVA\_release() on every instance located on the successfully ejected tapes.

## comment

Externalization comment.

## priorityLevel

The level of priority for this request. The priorityLevel can be in the range zero to one hundred or the value DIVA\_DEFAULT\_REQUEST\_PRIORITY. The value zero is the lowest priority and one hundred is the highest priority.

There are six predefined values as follows:

- DIVA\_REQUEST\_PRIORITY\_MIN
- DIVA\_REQUEST\_PRIORITY\_LOW
- DIVA\_REQUEST\_PRIORITY\_NORMAL
- DIVA\_REQUEST\_PRIORITY\_HIGH
- DIVA\_REQUEST\_PRIORITY\_MAX
- DIVA\_DEFAULT\_REQUEST\_PRIORITY

When the DIVA\_DEFAULT\_REQUEST\_PRIORITY value is used, the Core Manager uses the default priority defined in the Core Manager configuration for the request.

Using a value either outside of the range of zero to one hundred or predefined values yields a DIVA\_ERR\_INVALID\_PARAMETER error.

## requestNumber

The number identifying the request.

## Return Values

One of these DIVA\_STATUS constants defined in DIVAapi.h:

### DIVA\_OK

The request was correctly submitted and accepted by the Core Manager.

### DIVA\_ERR\_NOT\_CONNECTED

No connection is open.

### DIVA\_ERR\_SYSTEM\_IDLE

The DIVA Core system cannot accept connections and queries.

### DIVA\_ERR\_BROKEN\_CONNECTION

The connection with the Core Manager was broken.

### DIVA\_ERR\_TIMEOUT

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the DIVA\_API\_TIMEOUT variable and equals one hundred-eighty (180) seconds by default.

## DIVA\_ERR\_UNKNOWN

An unknown status was received from the Core Manager.

## DIVA\_ERR\_INTERNAL

An internal error was detected by the Core Manager or by the API.

## DIVA\_ERR\_INVALID\_PARAMETER

A parameter value was not understood by the Core Manager, or at least one of the barcodes refers to a bad tape (that is, an unknown tape, offline tape, or tape in use).

## DIVA\_ERR\_CANNOT\_ACCEPT\_MORE\_REQUESTS

The count of simultaneous requests has reached the maximum allowed value. This variable is set in the `manager.conf` configuration file. The default value is three hundred.

See also [DIVA\\_insertTape](#).

# DIVA\_enable\_Automatic\_Repack

Enable or disable the automatic repack scheduling in the Core Manager.

When the automatic repack scheduling is enabled, the schedule defined in the System Management App is applied and tapes belonging to Tape Groups for which repack is allowed can be repacked according to the other automatic repack settings.

When the automatic repack scheduling is disabled, all running automatic repack requests might be canceled (or not, according to other automatic repack settings), and no other automatic repack requests will be started until the automatic repack scheduling is turned on again (either from this API or from the System Management App).

## Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_enableAutomaticRepack (  
IN bool      enable  
);
```

## enable

Set true to enable automatic repack scheduling, false to disable.

## Return Values

One of these `DIVA_STATUS` constants defined in `DIVAapi.h`.

**DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

**DIVA\_ERR\_NOT\_CONNECTED**

No connection is open.

**DIVA\_ERR\_SYSTEM\_IDLE**

The DIVA Core system cannot accept connections and queries.

**DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

**DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the `DIVA_API_TIMEOUT` variable and equals one hundred-eighty (180) seconds by default.

**DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

**DIVA\_ERR\_INTERNAL**

An internal error was detected by the Core Manager or by the API.

## **DIVA\_getArchiveSystemInfo**

Retrieves general information about the DIVA Core system.

A DIVA Core system communicates with a Robotic System composed of one or more independent ACSs (Automated Cartridge Systems). An ACS is composed of one or more LSMs (Managed Storage Modules) that can exchange tapes through a PTP (Pass Through Port). Each tape drive is located in a LSM.

### **Synopsis**

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_getArchiveSystemInfo (  
IN string          options;  
OUT DIVA_GENERAL_INFO *info  
);
```

## info

Pointer to a `DIVA_GENERAL_INFO` structure that will be modified to include information about the DIVA Core system.

```

typedef enum {
    DIVA_IS_ON = 0,
    DIVA_IS_OFF,
    DIVA_GLOBAL_STATE_IS_UNKNOWN
} DIVA_GLOBAL_STATE;

typedef enum {
    DIVA_LIBRARY_OK = 0,
    DIVA_LIBRARY_OUT_OF_ORDER,
    DIVA_LIBRARY_STATE_UNKNOWN
} DIVA_LIBRARY_STATE;

class DIVA_ACTOR_AND_DRIVES_DESC {
public:
    string          actorName;
    string          actorAddress;
    bool            actorIsAvailable;
    vector<string>  *connectedDrives;
    bool            repackEnabled;
    bool            classicEnabled;
    bool            cacheArchiveEnabled;
    bool            directArchiveEnabled;
    bool            cacheRestoreEnabled;
    bool            directRestoreEnabled;
    bool            deleteEnabled;
    bool            copyToGroupEnabled;
    bool            associativeCopyEnabled;
    int             cacheForRepack;
};

class DIVA_LSM_DESC {
public:
    string          lsmName;
    int             lsmID;
    bool            lsmIsAvailable;
};

class DIVA_DRIVE_DESC {
public:
    string          driveName;
    int             driveTypeID;
    string          driveType;
    int             lsmID;
    bool            driveIsAvailable;
    bool            repackEnabled;
    bool            classicEnabled;
};

class DIVA_GENERAL_INFO {
public:
    DIVA_GLOBAL_STATE status;
    DIVA_LIBRARY_STATE lib_status;

```

```

int
vector<DIVA_ACTOR_AND_DRIVES_DESC>
vector<DIVA_LSM_DESC>
vector<DIVA_DRIVE_DESC>
int
long
long
int
vector<int>
vector<int>
int
int
int
string
string
int
int
int
totalNumberOfObjects;
*actorsDrivesList;
*lsmList;
*drivesList;
numberOfBlankTapes;
remainSizeOnTapes;
totalSizeOnTapes;
capSize;
*pendingRequests;
*currentRequests;
numOfAvailableActors
numOfAvailableDrives
numOfAvailableDisks
siteName
siteIpAddress
sitePort
firstUsedRequestId
lastUsedRequestId
};

```

The following parameters are listed in the order they appear in the preceding code example. Therefore there may be duplicates because the same parameter is used in different places in the code to represent different items.

### **actorName**

The name of the Actor.

### **actorAddress**

The Actor IP address.

### **actorIsAvailable**

Determines if the Actor is available.

### **connectedDrives**

Identifies the connected drives.

### **repackEnabled**

This is true if Repack is enabled.

### **classicEnabled**

This parameter is maintained for compatibility purposes only. This is only true if all seven standard operations are enabled.

**cacheArchiveEnabled**

This is true if Cached Archive is enabled.

**directArchiveEnabled**

This is true if Direct Archive is enabled.

**cacheRestoreEnabled**

This is true if Cached Restore is enabled.

**directRestoreEnabled**

This is true if Direct Restore is enabled.

**deleteEnabled**

This is true if Delete is enabled.

**copyToGroupEnabled**

This is true if Copy To Group is enabled.

**associativeCopyEnabled**

This is true if Associative Copy is enabled.

**cacheForRepack**

This is true if Cached Repack is enabled.

**lsmName**

User-friendly Managed Storage Module name.

**lsmID**

This is the unique LSM ID.

**lsmsAvailable**

This is true if the LSM identified by the preceding [lsmID](#) parameter is available for DIVA Core.

**driveName**

This is the Drive Name.

**driveTypeID**

This is the Drive Type ID.

**driveType**

This is the Drive Type Name.

**lsmID**

This is the ID of the LSM containing the drive. See [lsmList](#).

**drivesAvailable**

This is true if the identified drive is available for DIVA Core.

**status**

The status of DIVA Core.

**lib\_status**

This is ok if at least one ACS is online. See [lsmList](#).

**totalNumberOfObjects**

The number of Virtual Objects managed by this DIVA Core system.

**actorsDrivesList**

<DIVA\_ACTOR\_AND\_DRIVES\_DESC>

**lsmList**

<DIVA\_LSM\_DESC>

**drivesList**

<DIVA\_DRIVE\_DESC>

**numberOfBlankTapes**

The number of blank tapes in a Set associated with at least one Tape Group. Tape(s) may be externalized or write disabled.

**remainSizeOnTapes**

The sum of the remaining size of tapes (in gigabytes) that are online, in a Set associated with at least one Tape Group in an ACS where DIVA Core has a drive that is writable, and

the remaining size on disks accepting permanent storage. Only disks that are currently visible are used in the calculation.

Remaining\_Size\_of\_Online\_Tapes +  
Remaining\_Size\_of\_Disks\_Accepting\_Permanent\_Storage

### **totalSizeOnTapes**

The sum of the total size of all tapes (in gigabytes) in a Set associated with at least one Tape Group available for DIVA Core, and of the total size of all disks accepting storage. Only disks that are currently visible are used in the calculation.

Total\_Size\_of\_all\_Available\_Tapes + Total\_Size\_of\_all\_Disks\_Accepting\_Storage

### **capSize**

The number of slots in the default CAP.

### **pendingRequests**

The number of pending requests.

### **currentRequests**

The number of current requests.

### **numOfAvailableActors**

The number of currently running Actors.

### **numOfAvailableDrives**

The number of drives currently in online status.

### **numOfAvailableDisks**

The number of disks currently in online status.

### **siteName**

The name of the main site as entered in the System Management App.

### **siteIpAddress**

The Core Manager IP Address.

### **sitePort**

The port number where the Core Manager is listening.

## **firstUsedRequestId**

The first request ID used by the current Core Manager session. This value is -1 if no requests were processed.

## **lastUsedRequestId**

The last request ID used by the current Core Manager session. This value is -1 if no requests were processed.

## **Return Values**

One of the following DIVA\_STATUS constants defined in DIVAapi.h:

### **DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

### **DIVA\_ERR\_NOT\_CONNECTED**

No connection is open.

### **DIVA\_ERR\_SYSTEM\_IDLE**

The DIVA Core system cannot accept connections and queries.

### **DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

### **DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the DIVA\_API\_TIMEOUT variable and equals one hundred-eighty (180) seconds by default.

### **DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

### **DIVA\_ERR\_INTERNAL**

An internal error was detected by the Core Manager or by the API.

## DIVA\_getArrayList

The purpose of this function is to provide a list of arrays and disks associated with the arrays in the DIVA Core system. It also returns arrays without any disks associated with them. In DIVA Core 8.2 and later the Source Media Priority and storage options are reported in the returned data from this call.

### Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_getArrayList (
    IN          string options;
    OUT vector<DIVA_ARRAY_DESC> *arraysInfo
);
```

### arraysInfo

A pointer to a list of DIVA\_ARRAY\_DESC structures.

```
#ifndef WIN32
typedef long long __int64;
#endif

typedef enum {
    DIVA_CLOUD_STORAGECLASS_NONE=0,
        DIVA_CLOUD_STORAGECLASS_ARCHIVE,
        DIVA_CLOUD_STORAGECLASS_STANDARD
    } DIVA_CLOUD_STORAGECLASS;

class DIVA_ARRAY_DESC {
public:
    DIVA_STRING          arrayDesc;
    DIVA_STRING          arrayName;
    int                  number_Of_Disk;
    int                  mediaFormatId;
    DIVA_CLOUD_STORAGECLASS cloudStorageClass; (deprecated)
    vector<DIVA_DISK_ARRAY> *arrayDiskList;
    DIVA_STRING          storageOptions
};

typedef enum {
    DIVA_DISK_STATUS_UNKNOWN = 0,
    DIVA_DISK_STATUS_ONLINE,
    DIVA_DISK_STATUS_OFFLINE,
    DIVA_DISK_STATUS_NOT_VISIBLE
    } DIVA_DISK_STATUS;

class DIVA_DISK_ARRAY {
public:
    __int64              disk_CurrentRemainingSize;
    bool                 disk_isWritable;
    __int64              disk_maxThroughput;
    __int64              disk_minFreeSpace;
    DIVA_STRING          disk_name;
    DIVA_STRING          disk_site;
```

```
DIVA_DISK_STATUS      disk_status;  
__int64               disk_total_size;  
__int64               consumedSize;  
DIVA_STRING           disk_array_name;  
};
```

## arrayDesc

The description of the array.

## arrayName

The name of the array.

## numberOfDisk

The number of disks in the array.

## mediaFormatId

The format of the data on disks in this array. The value can be DIVA\_MEDIA\_FORMAT\_LEGACY, DIVA\_MEDIA\_FORMAT\_AXF, or DIVA\_MEDIA\_FORMAT\_AXF\_10. See information on media formats in the [Glossary](#).

## storageOptions

The Storage Class and Storage Location. Formatted as follows:

- oracle\_storage\_class=[NONE|ARCHIVE|STANDARD]
- storage\_location=[LOCAL|OPC|OCI]

## arrayDiskList

A list of the disks in an array.

## DIVA\_DISK\_STATUS\_UNKNOWN = 0

The disk status is unknown.

## DIVA\_DISK\_STATUS\_ONLINE

The disk status is online.

## DIVA\_DISK\_STATUS\_OFFLINE

The disk status is offline.

## DIVA\_DISK\_STATUS\_NOT\_VISIBLE

The disk status is not visible.

**disk\_CurrentRemainingSize**

The current remaining disk size.

**disk\_consumedSize**

The current consumed size on disk in kilobytes. Useful for unlimited cloud disks to determine the space consumed on the disk.

**disk\_isWritable**

This flag checks to see whether the disk is writable.

**disk\_maxThroughput**

The maximum throughput of a disk.

**disk\_minFreeSpace**

The minimum free space available on a disk.

**disk\_name**

The name of the disk.

**disk\_site**

The name of the site where the disk is located.

**disk\_status**

The current disk status.

**disk\_total\_size**

The total size of the disk.

**disk\_array\_name**

The name of the array containing the disk.

**Return Values**

One of the following DIVA\_STATUS constants defined in DIVAapi.h:

**DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

## DIVA\_ERR\_NOT\_CONNECTED

No connection is open.

## DIVA\_ERR\_BROKEN\_CONNECTION

The connection with the Core Manager was broken.

## DIVA\_ERR\_TIMEOUT

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the `DIVA_API_TIMEOUT` variable and equals one hundred-eighty (180) seconds by default.

## DIVA\_ERR\_UNKNOWN

An unknown status was received from the Core Manager.

## DIVA\_ERR\_INTERNAL

An internal error was detected by the Core Manager or by the API.

## DIVA\_getFinishedRequestList

Get all of the finished requests starting from the specified number of seconds before the present. Finished requests are requests that have completed normally or were terminated.

Use this function as follows:

If the list of requests to be processed is greater than the batch size, make successive calls to this function. The first time the function is called, set `initialTime` to the desired number of seconds earlier, where the list is to start. The maximum is three days. For successive calls set `initialTime` to zero and set the `uniqueId` to the value returned by the previous call. The returned list will be empty after all of the requests have been returned.

### Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_getFinishedRequestList (  
    IN int                batchSize,  
    IN int                initialTime,  
    IN DIVA_STRING        uniqueId,  
    OUT DIVA_FINISHED_REQUEST_INFO *pFinishedRequestInfo  
);
```

## batchSize

The maximum size of the returned list of Virtual Objects. This must be set to a value no greater than 1000; the recommended setting is 500. This is only a suggestion and may be overridden by the underlying functionality. This parameter should not be used to guarantee that the list will be a certain size.

## initialTime

The first time the function is called this value defines how far back in time to go to look for finished requests. Requests that have finished between this time and the present will be retrieved. The valid range for this parameter is 1 to 259200 (three days). If the number of requests to be returned is greater than the batch size, the call is repeated. For these calls this parameter should be set to zero (0).

## uniqueId

The first time the function is called this value must be set to an empty string (`_T("")`). Do not set this parameter to NULL. If the number of request to be returned is greater than the batch size, the call is repeated. For these calls this value should be set to the `uniqueId` as found in `DIVA_FINISHED_REQUEST_INFO` that was returned by the previous call.

## pFinishedRequestInfo

This is a pointer to the returned data. See the description of `DIVA_FINISHED_REQUEST_INFO` later in this section. It is the user's responsibility to allocate and delete instances of this class.

```
class DIVA_FINISHED_REQUEST_INFO {  
public:  
    DIVA_STRING                uniqueId;  
    vector<DIVA_REQUEST_INFO> *pRequestList;  
};
```

## uniqueId

After the first (and any subsequent) call, the API libraries update this variable with the current position in the search. Use this value as the input parameter to subsequent calls.

## pRequestList

This is a pointer to the returned data. See the description of `DIVA_REQUEST_INFO` under the description of [DIVA\\_getRequestInfo](#).

## Return Values

One of the following `DIVA_STATUS` constants defined in `DIVAapi.h`:

**DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

**DIVA\_ERR\_NOT\_CONNECTED**

No connection is open.

**DIVA\_ERR\_SYSTEM\_IDLE**

The DIVA Core system cannot accept connections and queries.

**DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

**DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the `DIVA_API_TIMEOUT` variable and equals one hundred-eighty (180) seconds by default.

**DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

**DIVA\_ERR\_INTERNAL**

An internal error was detected by the Core Manager or by the API.

## DIVA\_getFilesAndFolders

Retrieves the names of the files and folders for the specified Virtual Object from DIVA Core. This function is included to support complex Virtual Objects, but is valid for any Virtual Object.

You set the `startIndex` to zero to get all of the file and folder names for a Virtual Object. A list of names of the specified size is returned. You then set `startIndex` to the value of `nextStartIndex` and again make the function call. Continue this process until the return value equals `DIVA_WARN_NO_MORE_OBJECTS`.

### Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_getFilesAndFolders (  
IN DIVA_STRING          objectName,  
IN DIVA_STRING          objectCollection,  
IN int                  listType,  
IN int                  startIndex,  
IN int                  batchSize,
```

```
IN DIVA String          options,  
OUT DIVA_FILES_AND_FOLDERS *pFilesAndFolders  
);
```

## objectName

The name of the Virtual Object to be queried.

## objectCollection

The Collection assigned to the Virtual Object when it was archived.

## listType

Specifies what the returned list will include. See the definition of `DIVA_FILE_FOLDER_LIST_TYPE` later in this section.

## startIndex

The position in the list to start this iteration. Set at one (1) to start at the beginning. Values less than one are not valid. Set `startIndex` equal to `nextStartIndex` as returned in `DIVA_FILES_AND_FOLDERS` for all subsequent calls.

## batchSize

The maximum size of the returned list of Virtual Objects. This must be set to a value no greater than 1000; the recommended setting is 500. This is only a suggestion and may be overridden by the underlying functionality. This parameter should not be used to guarantee that the list will be a certain size.

## options

Field for optional `getFilesAndFolders` parameters.

## pFilesAndFolders

This is a pointer to the returned data. See the description of `DIVA_FILES_AND_FOLDERS` later in this section. It is the responsibility of the user to allocate and delete instances of this class.

```
Typedef enum {  
    DIVA_LIST_TYPE_FILES_ONLY = 0,  
    DIVA_LIST_TYPE_FOLDERS_ONLY = 1,  
    DIVA_LIST_TYPE_FILES_AND_FOLDERS = 2  
} DIVA_FILE_FOLDER_LIST_TYPE;
```

## DIVA\_LIST\_TYPE\_FILES\_ONLY

This function will return files and symbolic links.

## DIVA\_LIST\_TYPE\_FOLDERS\_ONLY

This function will return folders only.

## DIVA\_LIST\_TYPE\_FILES\_AND\_FOLDERS

This function will return files and folders and symbolic links.

```
class DIVA_FILES_AND_FOLDERS {  
public:  
    DIVA_OBJECT_SUMMARY          objectSummary;  
    bool                         isComplex;  
    int                          nextStartIndex;  
    DIVA String                  siteName;  
    vector<DIVA_FILE_FOLDER_INFO> *pFileFolderList;  
};
```

### objectSummary

The ID of the Virtual Object. See the description later in this section.

### isComplex

This is true when the Virtual Object is a complex Virtual Object.

### nextStartIndex

After the first and any subsequent call, the API libraries update this variable with the current position in the search. Use this value as the input parameter for subsequent calls.

### siteName

This contains the site name of the Core Manager that satisfied the request.

### pFileFolderList

This is a pointer to the list of files and folders. See the description of DIVA\_FILE\_FOLDER\_INFO later in this section.

```
class DIVA_OBJECT_SUMMARY {  
public:  
    string      objectName;  
    string      objectCollection;  
};
```

### objectName

This is the name of the Virtual Object.

## objectCollection

This is the Collection of the Virtual Object.

```
class DIVA_FILE_FOLDER_INFO {  
public:  
    DIVA_STRING          fileOrFolderName;  
    bool                 isDirectory;  
    bool                 isSymbolicLink;  
    __int64              sizeBytes;  
    int                  fileId;  
    int                  totalNumFilesFolders;  
    __int64              totalSizeFilesFolders;  
    vector<DIVA_CHECKSUM_INFO> pChecksumInfoList;  
};
```

### fileOrFolderName

The name of the file or folder.

### isDirectory

This is true if the component is a directory.

### isSymbolicLink

This is true if the component is a symbolic link.

### sizeBytes

The size of the file in bytes. This is valid only for files.

### fileId

This is a unique ID for each file created by DIVA Core as part of the processing of this command.

### totalNumFilesFolders

The number of files and sub folders. This is valid only for folders in a complex Virtual Object.

### totalSizeFilesFolders

The total size of all files, including files in sub folders. This is valid only for folders in a complex Virtual Object.

### pChecksumInfoList

This is a pointer to a list of checksums for a file. Directories will not contain checksums. It is also possible that some files in the archive will not contain checksum information. See the description later in this section.

```
class DIVA_CHECKSUM_INFO {  
public:  
    DIVA_STRING      checksumType;  
    DIVA_STRING      checksumValue;  
    bool             isGenuine;  
};
```

## checksumType

The type of checksum (MD5, SHA1, and so on).

## checksumValue

The value of the checksum in hexadecimal string format.

## isGenuine

This is true if this checksum was provided at the time of archiving and verified as a Genuine Checksum.

## Return Values

The API includes the following return values for this call:

- The file list contains empty files for non-complex Virtual Objects.
- The folders list contains all folders in a non-complex Virtual Object.
- Both the Folders Only and Files and Folders options are available for use with non-complex Virtual Objects.

One of these DIVA\_STATUS constants defined in DIVAapi.h:

## DIVA\_OK

The request was correctly submitted and accepted by the Core Manager.

## DIVA\_ERR\_NOT\_CONNECTED

No connection is open.

## DIVA\_ERR\_SYSTEM\_IDLE

The DIVA Core system cannot accept connections and queries.

## DIVA\_ERR\_BROKEN\_CONNECTION

The connection with the Core Manager was broken.

## DIVA\_ERR\_TIMEOUT

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the `DIVA_API_TIMEOUT` variable and equals one hundred-eighty (180) seconds by default.

## DIVA\_ERR\_UNKNOWN

An unknown status was received from the Core Manager.

## DIVA\_ERR\_INTERNAL

An internal error was detected by the Core Manager or by the API.

## DIVA\_WARN\_NO\_MORE\_OBJECTS

The end of the list was reached during the call.

# DIVA\_getGroupsList

Returns the description of all Tape Groups. In DIVA Core 8.2 and later the Source Media Priority is reported in the returned data from this call.

## Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_getGroupsList (
OUT vector<DIVA_GROUP_DESC>      *&groups
);
```

## groups

This is a pointer to a list of `DIVA_GROUP_DESC` structures.

```
class DIVA_GROUP_DESC {
public:
    string      group_name;
    string      group_desc;
    int         mediaFormatId;
};
```

## group\_name

The configured name of the tape group.

## group\_desc

The description of the tape group.

## mediaFormatId

The format of the tapes added to this Tape Group. The value can be `DIVA_MEDIA_FORMAT_LEGACY`, `DIVA_MEDIA_FORMAT_AXF`, or `DIVA_MEDIA_FORMAT_AXF_10`. See information on media formats in the [Glossary](#).

## Return Values

One of these `DIVA_STATUS` constants defined in `DIVAapi.h`:

### DIVA\_OK

The request was correctly submitted and accepted by the Core Manager.

### DIVA\_ERR\_NOT\_CONNECTED

No connection is open.

### DIVA\_ERR\_SYSTEM\_IDLE

The DIVA Core system cannot accept connections and queries.

### DIVA\_ERR\_BROKEN\_CONNECTION

The connection with the Core Manager was broken.

### DIVA\_ERR\_TIMEOUT

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the `DIVA_API_TIMEOUT` variable and equals one hundred-eighty (180) seconds by default.

### DIVA\_ERR\_UNKNOWN

An unknown status was received from the Core Manager.

### DIVA\_ERR\_INTERNAL

An internal error was detected by the Core Manager or by the API.

See also [DIVA\\_getObjectInfo](#).

## DIVA\_getObjectDetailsList

The `DIVA_getObjectDetailsList` is an API call to retrieve Virtual Object information from the DIVA Core database. Only the latest state of the Virtual Object is returned. Virtual Objects may be repeated across batches if the Virtual Object is modified multiple times as the call advances (in time) from a user-specified time across Virtual Objects in the DIVA Core database.

- The created-since call retrieves all Virtual Objects created since a certain time.
- The deleted-since call retrieves all Virtual Objects deleted since a certain time.
- If starting from a user-specified time of zero, the modified-since call retrieves all Virtual Objects created since a certain time, and returns the state of the database from a time of zero.
- If starting from a user-specified time greater than zero, the call returns all Virtual Objects created and deleted since a certain time, and all Virtual Objects with newly created and (or) deleted instances.

In DIVA Core 8.2 and later storage options (at the instance level) are reported in the returned data from this call.

The listPosition vector returned by a GetObjectDetailsList call must be passed in to a subsequent call. Its content must not be altered by the user of the call.

Different detail levels can be specified (see the following Level of Detail Setting information). Level 0 will be the fastest, while Level 3 will return all possible details. Only the highest level of detail is supported. Using a lower level of detail will still return all information for Virtual Objects.

The output can be structured using the DIVA\_OBJECTS\_LIST option, or through the DIVA\_TAPE\_INFO\_LIST option. The output structure type is configured by setting the pListType parameter of the call.

The API client application should use the DIVA\_OBJECTS\_LIST setting in the following cases:

- To retrieve a list of Virtual Objects instances added to DIVA Core.
- To retrieve a list of Virtual Objects instances deleted from DIVA Core.
- To retrieve a combined list of all changes in the DIVA Core Virtual Object database (adding and deleting Virtual Objects, adding and deleting instances)
- To continuously monitor the DIVA Core system to retrieve events of adding and deleting Virtual Objects, and adding and deleting instances.

The API client application should use the DIVA\_TAPE\_INFO\_LIST setting to retrieve a list of tape instances for any instances added, deleted, repacked, ejected, or inserted.

---

**Note:** The DIVA\_TAPE\_INFO\_LIST will not return any results for deleted instances if all Virtual Objects are deleted.

---

## Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_getObjectDetailsList (
IN bool                fFirstTime,
IN time_t              *initialTime,
IN int                 pListType,
IN int                 pObjectsListType,
IN int                 pMaxListSize,
```

```

IN DIVA_STRING          pObjectName,
IN DIVA_STRING          pObjectCollection,
IN DIVA_STRING          pMediaName,
DIVA_LEVEL_OF_DETAIL    pLevelOfDetail,
IN vector<DIVA_STRING>   listPosition,
OUT vector<DIVA_OBJECT_DETAILS_LIST> *pObjectDetailsList
);

```

## fFirstTime

The first time this function is called this parameter must be set to true. Every subsequent call should be set to false and listPosition must be copied from the listPosition value returned by the previous call to DIVA\_GetObjectDetailsList.

## intialTime

The start time of the list. Data is collected and returned corresponding to this time and later. To retrieve all items in the database, use zero as the start time value.

## pListType

One of the codes defined by the enumeration DIVA\_LIST\_TYPE.

## pObjectsListType

One of the codes defined by the enumeration DIVA\_OBJECTS\_LIST\_TYPE.

To retrieve all Virtual Objects created, deleted, or modified since a certain time, set this to DIVA\_OBJECTS\_CREATED\_SINCE, DIVA\_OBJECTS\_DELETED\_SINCE, or DIVA\_OBJECTS\_MODIFIED\_SINCE, respectively.

To retrieve tape related information for all Virtual Objects that have been created, deleted, repacked, ejected, and (or) inserted since a certain time, set this parameter to DIVA\_INSTANCE\_CREATED, DIVA\_INSTANCE\_DELETED, DIVA\_INSTANCE\_REPACKED, DIVA\_INSTANCE\_EJECTED, DIVA\_INSTANCE\_INSERTED, respectively.

To retrieve any combination of the above, use the pipe operator. For example, to retrieve tape information for Virtual Objects with tape instances that have been created and repacked since a certain time, use DIVA\_INSTANCE\_CREATED | DIVA\_INSTANCE\_REPACKED.

## pMaxListSize

The maximum size of the returned list of Virtual Objects. This must be set to a value no greater than 1000; the recommended setting is 500. This is only a suggestion and may be overridden by the underlying functionality. This parameter should not be used to guarantee that the list will be a certain size.

## pObjectCollection

Filter the returned list of Virtual Objects based on the provided Virtual Object Collection. The asterisk wildcard can be used (for example, \*video).

## pMediaName

Filter the returned list of Virtual Objects based on the provided media name. The asterisk wildcard can be used (for example, soap\*).

## pLevelOfDetail

One of the codes defined by the enumeration `DIVA_LEVEL_OF_DETAIL`. Filtering by Virtual Object name, Collection, and Tape Group (media name) is performed at all levels of detail.

The `DIVA_OBJECTS_CREATED_SINCE` and `DIVA_OBJECTS_MODIFIED_SINCE` options work with all levels of detail.

The `DIVA_OBJECTS_DELETED_SINCE` option only works with the `DIVA_OBJECTNAME_AND_COLLECTION` level of detail.

The `DIVA_TAPE_INFO_LIST` only works with the `DIVA_OBJECTNAME_AND_COLLECTION` and `DIVA_INSTANCE` level of detail.

## listPosition

A vector of `DIVA_STRING` type. The elements of this list are for internal use only and do not need to be extracted by the user.

When `pFirstTime` is true, a new empty list must be constructed and included.

When `pFirstTime` is false, `listPosition` must be updated with the `listPosition` attribute of `pObjectDetailsList` since this attribute points to the last Virtual Object retrieved by the last call of `DIVA_getObjectDetailsList`.

## pObjectDetailsList

This is a pointer to the `DIVA_OBJECT_DETAILS_LIST` class. This is the output parameter that will contain the response to the call.

Use the `listPosition` parameter from this response as the `listPosition` argument in subsequent calls to `GetObjectDetailsList`.

For `pListType = DIVA_OBJECTS_LIST`, all of the Virtual Object and (or) instance information is stored in the `objectInfo` attribute.

For `pListType = DIVA_TAPE_INFO_LIST`, all Virtual Object and tape information is stored in the `objectTapeInfo` attribute.

```
typedef enum {  
  
    DIVA_OBJECTNAME_AND_Collection = 0,  
    DIVA_MISC = 1,
```

```
DIVA_COMPONENT = 2,
DIVA_INSTANCE = 3
} DIVA_LEVEL_OF_DETAIL;
```

## DIVA\_OBJECTNAME\_AND\_COLLECTION (0)

The getObjectDetailsList function will only return the Virtual Object name and Collection.

## DIVA\_MISC (1)

The getObjectDetailsList function will return the comments, archive date, name and path on the source, and all data returned with the DIVA\_OBJECTNAME\_AND\_COLLECTION level of detail.

## DIVA\_COMPONENT (2)

The getObjectDetailsList function will return the size of the Virtual Object, list of components value, and all data returned with the DIVA\_MISC level of details.

## DIVA\_INSTANCE (3)

The getObjectDetailsList function will return all instance information, repack state, related active request information data, and all data returned with the DIVA\_COMPONENT level of detail.

```
typedef enum {
```

```
DIVA_OBJECTS_LIST = 1,
DIVA_TAPE_INFO_LIST = 2
} DIVA_LIST_TYPE;
```

DIVA\_OBJECTS\_LIST\_TYPE is defined as follows:

```
typedef enum {
```

```
DIVA_OBJECTS_CREATED_SINCE = 0x0001,
DIVA_OBJECTS_DELETED_SINCE = 0x0002,
DIVA_OBJECTS_MODIFIED_SINCE = 0x0003,
DIVA_INSTANCE_NONE = 0x0000,
DIVA_INSTANCE_DELETED = 0x0020,
DIVA_INSTANCE_REPACKED = 0x0040,
DIVA_INSTANCE_EJECTED = 0x0080,
DIVA_INSTANCE_INSERTED = 0x0100
} DIVA_OBJECTS_LIST_TYPE;
```

```
class DIVA_OBJECT_DETAILS_LIST {
public:
    int listType;
    DIVA_STRING siteID;
    vector<DIVA_STRING> *listPosition;
    vector<DIVA_OBJECT_INFO> *objectInfo;
    vector<DIVA_OBJECT_TAPE_INFO> *objectTapeInfo;
};
```

## listType

One of the codes defined by the enumeration DIVA\_LIST\_TYPE.

## siteId

The DIVA Core system name as configured in manager.conf.

## listPosition

After the first and any subsequent call, the API libraries update this variable with the current position in the search. This Virtual Object must be provided as the input parameter to any subsequent calls.

## objectInfo

This is a pointer to a DIVA\_OBJECT\_INFO structure. The structure should be allocated and deleted by the caller. The structure contains information about the Virtual Object details, such as the list of components, tape instances, and other properties described in API call getObjectInfo.

## objectTapeInfo

This is a pointer to a list of DIVA\_OBJECT\_TAPE\_INFO structures. The structure should be allocated and deleted by the caller. The structure contains information about the tapes containing instances of the Virtual Object and other properties described in API call getObjectTapeInfo.

```

class DIVA_OBJECT_INFO {
public:
    DIVA_OBJECT_SUMMARY objectSummary;
    DIVA_STRING          uuid;
    int                  lockStatus;
    __int64              objectSize;
    __int64              objectSizeBytes;
    vector<string>        *filesList;
    string               objectComments;
    time_t               archivingDate;
    bool                 isInserted;
    vector<DIVA_TAPE_INSTANCE_DESC> *tapeInstances;
    vector<DIVA_ACTOR_INSTANCE_DESC> *actorInstances;
    string               objectSource;
    string               rootDirectory;
    vector<int>          *relatedRequests;
    bool                 toBeRepacked;
    int                  modifiedOrDeleted;
    bool                 isComplex;
    int                  nbFilesInComplexComponent;
    int                  nbFoldersInComplexComponent;
};

```

## **objectSummary**

The Virtual Object name and Collection.

## **UUID**

Universally Unique Identifier to uniquely identify each Virtual Object created in DIVA Core across all Telestream customer sites. This does not include Virtual Objects created using Copy As requests. A Virtual Object created through a Copy As request will contain the same UUID as that of the Source Server Virtual Object.

## **lockStatus**

This is the locking status of the Virtual Object. Virtual Objects in the archive can be locked. When a Virtual Object is locked it cannot be restored or copied to a new name. This feature prevents the use of a Virtual Object that has an expired copyright, and so on. The Virtual Object is unlocked when this value is zero.

## **objectSize**

This is the Virtual Object size in kilobytes.

## **objectSizeBytes**

This is the Virtual Object size in bytes.

## **filesList**

This is a list of the files in the Virtual Object. A single wrapper file name is returned for complex Virtual Objects.

## **objectComments**

This is the comments saved when the Virtual Object was archived.

## **archivingDate**

Then number of seconds since January 1, 1970.

## **isInserted**

This is true if at least one instance of this Virtual Object is either on a tape that is currently inserted in the Managed Storage, or a disk that is online.

## **tapeInstances**

This is a list of Virtual Object instances saved to tape.

**actorInstances**

This is a list of Virtual Object instances saved to disk.

**objectSource**

The Source Server system used to archive the Virtual Object.

**rootDirectory**

The root directory containing the Virtual Object files on the objectsource.

**relatedRequests**

This is non-terminated requests.

**toBeRepacked**

This is false unless all instances are going to be repacked.

**modifiedOrDeleted**

One of DIVA\_MODIFIED\_OR\_DELETED as follows:

UNDEFINED - The levelOfDetail does not equal DIVA\_INSTANCE.

DIVA\_CREATED\_OR\_MODIFIED - The Virtual Object was created, or an instance was either added or removed.

DIVA\_DELETED - The Virtual Object was removed.

**isComplex**

This is true if this is a complex Virtual Object.

**nbFilesInComplexComponent**

This is the number of files in the Virtual Object. This is used only for complex Virtual Objects. The value is zero for non-complex Virtual Objects.

**nbFoldersInComplexComponent**

This is the number of folders in the Virtual Object. This is used only for complex Virtual Objects. The value is zero for non-complex Virtual Objects.

```
class DIVA_OBJECT_SUMMARY {  
public:  
    string      objectName;  
    string      objectCollection;  
};
```

## objectName

This is the Virtual Object name.

## objectCollection

This is the Virtual Object Collection.

```
class DIVA_TAPE_INSTANCE_DESC {
public:
    int                instanceID;
    string             groupName;
    vector<DIVA_TAPE_DESC> *tapeDesc;
    bool              isInserted;
    DIVA_REQUIRE_STATUS reqStatus;
};
```

## instanceId

The numeric instance identifier.

## groupName

The name of the Tape Group this tape is assigned to.

## tapeDesc

Additional information about this tape.

## isInserted

This is true if at least one instance of this Virtual Object is either on a tape that is currently inserted in the Managed Storage, or a disk that is online.

## reqStatus

Determines if the instance is Required or Released.

DIVA\_REQUIRED - The instance is requested to be inserted into the Managed Storage.

DIVA\_RELEASED - There is no need to have this instance present in the Managed Storage.

```
class DIVA_TAPE_DESC {
public:
    string            vsn;
    bool              isInserted;
    string            externalizationComment;
    bool              isGoingToBeRepacked;
    int               mediaFormatId;
};
```

**vsn**

The volume serial number (barcode).

**isInserted**

This is true if at least one instance of this Virtual Object is either on a tape that is currently inserted in the Managed Storage or a disk that is online.

**externalizedComment**

Comment saved when the tape was exported.

**isGoingToBeRepacked**

This is false unless all instances are going to be repacked.

**mediaFormatId**

The format of the data on to be used. The value can be DIVA\_MEDIA\_FORMAT\_DEFAULT, DIVA\_MEDIA\_FORMAT\_LEGACY, DIVA\_MEDIA\_FORMAT\_AXF, or DIVA\_MEDIA\_FORMAT\_AXF\_10. This is only used when the listType is Tape.

```
typedef enum {
    DIVA_CLOUD_STORAGECLASS_NONE=0,
        DIVA_CLOUD_STORAGECLASS_ARCHIVE,
        DIVA_CLOUD_STORAGECLASS_STANDARD
    } DIVA_CLOUD_STORAGECLASS;

class DIVA_ACTOR_INSTANCE_DESC {
public:
    int                instanceID;
    string             actor;
    DIVA_CLOUD_STORAGECLASS cloudStorageClass; (deprecated)
    DIVA_STRING        storageOptions;
};
```

**instanceID**

The numeric ID of the instance.

**actor**

This field reports the name of the disk array where the instance is stored instead of the Actor name.

```
typedef enum {
    DIVA_REQUIRED = 0,
    DIVA_RELEASED
    } DIVA_REQUIRE_STATUS;

typedef enum {
```

```

DIVA_UNDEFINED = 0,
DIVA_CREATED_OR_MODIFIED,
DIVA_DELETED
} DIVA_MODIFIED_OR_DELETED;

```

## Return Values

The file list of each Virtual Object in the Virtual Objects list now contains empty files (that is, files of size 0 bytes). Client applications developed against API releases before release 7.5 will receive empty files in the file list that accompanies a Details List message. Depending on the input parameters, the `DIVA_getObjectDetailsList` function will return values as described in the following table.

List Type	Virtual Object List Type	Supported Detail Level	Return Value
DIVA_OBJECTS_LIST	DIVA_OBJECTS_CREATED_SINCE	All	List Virtual Objects that have been created since a specified time.
DIVA_OBJECTS_LIST	DIVA_OBJECTS_DELETED_SINCE	Only DIVA_OBJECTNAME_AND_COLLECTION	List Virtual Objects that have been deleted since a specified time.
DIVA_OBJECTS_LIST	DIVA_OBJECTS_MODIFIED_SINCE	Only DIVA_INSTANCE	List Virtual Objects that have been created/ deleted since a certain time, plus Virtual Objects with new or deleted instances.  If the list of instances is empty, Virtual Objects were deleted.  If the list of instances is not empty, Virtual Objects were created or updated.
DIVA_TAPE_INFO_LIST	DIVA_INSTANCE_NONE (0x0000)	Only DIVA_OBJECTNAME_AND_COLLECTION and DIVA_INSTANCE level.	List Virtual Objects and tape information for all tape instances (no filter).
DIVA_TAPE_INFO_LIST	DIVA_INSTANCE_CREATED (0x0010)	Only DIVA_OBJECTNAME_AND_COLLECTION and DIVA_INSTANCE level.	List Virtual Objects and tape information for all tape instances created since a specified time.
DIVA_TAPE_INFO_LIST	DIVA_INSTANCE_DELETED (0x0020)	Only DIVA_OBJECTNAME_AND_COLLECTION and DIVA_INSTANCE level.	List Virtual Objects and tape information for all tape instances deleted since a specified time.

List Type	Virtual Object List Type	Supported Detail Level	Return Value
DIVA_TAPE_INFO_LIST	DIVA_INSTANCE_REPACKED (0x0040)	Only DIVA_OBJECTNAME_AND_COLLECTION and DIVA_INSTANCE level.	List Virtual Objects and tape information for all tape instances repacked since a specified time.
DIVA_TAPE_INFO_LIST	DIVA_INSTANCE_EJECTED (0x0080)	Only DIVA_OBJECTNAME_AND_COLLECTION and DIVA_INSTANCE level.	List Virtual Objects and tape information for all tape instances ejected since a specified time.
DIVA_TAPE_INFO_LIST	DIVA_INSTANCE_INSERTED (0x0100)	Only DIVA_OBJECTNAME_AND_COLLECTION and DIVA_INSTANCE level.	List Virtual Objects and tape information for all tape instances inserted since a specified time.

## Use with DIVA Connect

All filters are applied at an Virtual Object level as follows:

- If you request Virtual Objects satisfying certain filter constraints, those constraints are applied to the Virtual Object and not to individual instances of a Virtual Object.
- If you specify a Virtual Object name and Collection filter, the list will be filtered to contain only Virtual Objects satisfying the specified Virtual Object name and Collection.

Media name is defined at an instance level, not at a Virtual Object level. A media name filter will only allow Virtual Objects with at least one instance satisfying the requested media name filter.

---

**Note:** If an instance of a Virtual Object is created or deleted, and you request all modified Virtual Objects with a particular media name, the Virtual Object will be returned if and only if any instance of the Virtual Object satisfies the media name filter.

---

### Example:

A new instance Virtual Object-A was added at time 101 with the media name CAR. Virtual Object-A has a total of two instances. One instance has the media name TRUCK and the other has the media name CAR.

An instance of Virtual Object-B was removed at time 101 with the media name CAR. Virtual Object-B has only one instance.

A new instance of Virtual Object-C was added at time 99 with the media name TRAIN. Virtual Object-C has a total of two instances. One instance has the media name TRAIN and the other has the media name HANG GLIDE.

A user executes a `getObjectDetailsList` call with `MODIFIED SINCE TIME 100` and `MEDIA NAME FILTER = T*`.

The only Virtual Object that was modified since time 100, and has at least one instance with a media name of T is Virtual Object-A. Therefore, the result is that the list returned by the `getObjectDetailsList` call contains only Virtual Object-A.

## Use and Recommended Practices

Telestream recommends that the API client application adhere to the following sequence of actions:

1. Create a variable of `DIVA_OBJECT_DETAILS_LIST` type to store the Virtual Object information returned by the call.
2. Create a variable of vector `<DIVA_STRING>` type to serve as the `listPosition` Virtual Object. This will be used as the `listPosition` argument to `DIVA_GetObjectDetailsList`.
3. Create a variable of `time_t` type and set to the time at which the list is to start. Set this to zero to include all Virtual Objects in the database.
4. Create a variable of Boolean type and set it to true to indicate that this is the first call in a sequence of calls.
5. Create a variables of Integer type to hold the `listType` and `objectsListType` to specify the type of call.  
Example: Use `DIVA_OBJECTS_LIST` and `DIVA_OBJECTS_MODIFIED SINCE` to indicate that you want Virtual Object information for modified Virtual Objects.
6. Create a variable of Integer type to hold the suggested number of Virtual Objects you want returned by the call.
7. Create list filtering variables of `DIVA_CHAR[ ]` type to hold the Virtual Object name, Collection and media filters.
8. Create a variable of Integer type to hold the level of detail you want returned.
9. Execute `DIVA_GetObjectDetailsList` with the variables previously mentioned.
10. Use the data stored in the variable from Step 1 as needed by your application.
11. Copy the `listPosition` attribute of the call's output created in Step 1 into the `listPosition` variable created in Step 2.
12. Repeat steps 8, 9, and 10 for until you no longer need to monitor DIVA Core.
13. All variables must be deallocated after exiting the loop.

Multiple simultaneous calls to `DIVA_getObjectDetailsList` are supported. However, this call places a heavy demand on the database. Therefore simultaneous and (or) frequent calls to this function should be avoided.

Continuous monitoring of DIVA Core requires a procedure similar to the one defined in the section [Recommended Practices for Continuous Updates Notification Design Pattern \(No Media Filter\)](#).

Duplication of Virtual Objects can occur across different return portions. It is important to handle these cases by examining the data returned by the call. For a `MODIFIED SINCE` call, you must compare the instances of the duplicate Virtual Object

returned by successive calls to identify whether new information about the Virtual Object is available and update your local repository accordingly.

An empty list may be returned as a valid result. This indicates that there were no changes to the system after the time specified in the last call. It is important to continue querying DIVA Core with the `DIVA_getObjectDetailsList` call using the ID from the previous call. However, the call frequency must be reduced after you receive an empty list. This reduces the load on the DIVA Core database.

The same application can use the `DIVA_getObjectDetailsList` function effectively for both the initial database synchronization (if the client application maintains a database) and later use it for continuous monitoring after the database is updated.

During the initial database synchronization phase, it is necessary for the application to make frequent sequential calls to synchronize the local database with the DIVA Core database. The application must call `DIVA_getObjectDetailsList`, wait for a response, and then repeat the process.

After the synchronization phase, it is necessary for the application to go into the continuous monitoring phase, where it must make periodic calls to update the system with the latest Virtual Object information. Telestream recommends a call interval of once every several minutes. Continuous, frequent execution of this call can heavily impact the database and degrade system performance.

The amount of data retrieved by the `CREATED_SINCE` and `MODIFIED_SINCE` call is substantial (Virtual Object, instance, and component data for each Virtual Object). Therefore, Telestream recommends that most applications use 500 as the maximum list size setting.

## Recommended Practices for Continuous Updates Notification Design Pattern (No Media Filter)

The continuous updates notification design pattern is used in multiple applications, and is important when using the API. The client application can use the internal database to continuously update the local database information with changes in the DIVA Core database. Following the design pattern helps develop the performance-optimized updates notification workflow.

The application must submit the call with the `objectListType` set to `MODIFIED_SINCE` with the level of detail required to collect instance-level information. Additionally, the `First Time` flag must be set true, and all necessary filter parameters must be set (Virtual Object name and Collection).

This is the process the application will follow:

1. The application receives a list of Virtual Objects and a new `listPosition`.
2. On the next cycle, the application will execute the call using the `listPosition` obtained in Step 1 and the `First Time` flag set to false. It is acceptable to submit another call immediately after receiving the list if the system is being used solely for synchronization purposes. Otherwise, it is recommended to wait for a period between calls to allow other requests to process.

3. Repeat Steps 1 and 2 for the course of execution to keep the internal database synchronized with DIVA Core database.
4. If none of the Virtual Objects in DIVA Core have been modified, the list will be EMPTY, which indicates there were no updates since the last call. The application should wait for a specific amount of time, and then retry.

The application must check the list of instances to see if the following occurred:

- The value of modifiedOrDeleted in the DIVA\_OBJECT\_INFO equals DELETED, Virtual Objects were deleted and the database must be updated.
- The value of modifiedOrDeleted in the DIVA\_OBJECT\_INFO equals CREATED\_OR\_MODIFIED, the Virtual Object was either created or updated.
  - If the Virtual Object previously existed in the database, the database list of instances must be updated.
  - If the Virtual Object does not exist in the database, it must be added to the database.

---

**Note:** To ensure continuous updates, the listPosition Virtual Object should be preserved throughout the course of operations.

---

### Example:

MAIN:

```

CREATE LIST_POSITION VARIABLE
CREATE DETAILS_LIST VARIABLE
SET FIRST_TIME = TRUE
SET INITIAL_TIME = 0
SET LIST_TYPE = DIVA_OBJECTS_LIST
SET OBJECTS_LIST_TYPE = DIVA_OBJECTS_MODIFIED_SINCE
SET LEVEL_OF_DETAIL = DIVA_OBJECTS_MODIFIED_SINCE
SET SIZE = 500
SET OBJECT_NAME = "*"
SET COLLECTION = "*"
SET MEDIA_NAME = "*"
CALL GetObjectDetailsList(FIRST_TIME, LIST_TYPE,
OBJECTS_LIST_TYPE, LIST_POSITION, SIZE, INITIAL_TIME, OBJECT_NAME,
COLLECTION, MEDIA_NAME, LEVEL_OF_DETAIL, DETAILS_LIST)
// 1

UNIQUE_ID AND DETAILS_LIST VARIABLES WERE UPDATED BY CALL // 2

CALL SYNC_OBJECTS // 6

START LOOP
  SET FIRST_TIME = FALSE
  CALL GetObjectDetailsList(...) // 3
  LIST_POSITION AND DETAILS_LIST VARIABLES WERE UPDATED BY CALL
  CALL SYNC_OBJECTS // 6
END LOOP (TERMINATE AT END OF APPLICATION LIFE) // 4

SYNC_OBJECTS:
  IF (DETAILS_LIST IS NOT EMPTY) // 5

```

```

FOR(OBJECT IN DETAILS_LIST)
  IF (OBJECT.modifiedOrDeleted EQUALS DELETED)
    DELETE OBJECT FROM DATABASE // 6a
  ELSE
    IF (OBJECT.modifiedOrDeleted EQUALS CREATED_OR_MODIFIED)
      ADD OR UPDATE OBJECT TO DATABASE // 6b
    END IF
  END IF
END FOR
END IF

```

### Return Values

One of these DIVA\_STATUS constants defined in DIVAapi.h:

#### **DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

#### **DIVA\_ERR\_NOT\_CONNECTED**

No connection is open.

#### **DIVA\_ERR\_SYSTEM\_IDLE**

The DIVA Core system cannot accept connections and queries.

#### **DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

#### **DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the DIVA\_API\_TIMEOUT variable and equals one hundred-eighty (180) seconds by default.

#### **DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

#### **DIVA\_ERR\_INTERNAL**

An internal error was detected by the Core Manager or by the API.

#### **DIVA\_WARN\_NO\_MORE\_OBJECTS**

The end of the list was reached during the call.

## DIVA\_getObjectInfo

Returns information about a particular Virtual Object in the DIVA Core system.

The vector<DIVA\_ACTOR\_INSTANCE\_DESC> \*actorInstances parameter is kept unchanged for compatibility, although it is formally a vector of diskInstance and not actorInstance.

The file list can contain empty files (that is, files of size 0 bytes). Client applications developed against API releases before release 7.5 will also receive empty files in the file list that accompanies an objectInfo message.

For compatibility reasons, the class DIVA\_ACTOR\_INSTANCE\_DESC designates a disk instance (not a Actor instance) and its string actor field now contains the array name instead of a Actor name.

In DIVA Core 8.2 and later storage options (at the instance level) are reported in the returned data from this call.

### Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_getObjectInfo (
IN DIVA_STRING          objectName,
IN DIVA_STRING          objectCollection,
IN DIVA_STRING          options,
OUT DIVA_OBJECT_INFO    *objectInfo
);
```

### objectName

The name of the queried Virtual Object.

### objectCollection

The Collection assigned to the Virtual Object when it was archived. This parameter can be a null string, however this may result in an error if several Virtual Objects have the same name.

### options

Optional string attribute for specifying additional parameters to the request.

### objectInfo

Pointer to a DIVA\_OBJECT\_INFO structure allocated and deleted by the caller. See [DIVA\\_getObjectDetailsList](#) for a description of DIVA\_OBJECT\_INFO.

### Return Values

One of these DIVA\_STATUS constants defined in DIVAapi.h:

**DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

**DIVA\_ERR\_NOT\_CONNECTED**

No connection is open.

**DIVA\_ERR\_SYSTEM\_IDLE**

The DIVA Core system cannot accept connections and queries.

**DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

**DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the `DIVA_API_TIMEOUT` variable and equals one hundred-eighty (180) seconds by default.

**DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

**DIVA\_ERR\_INTERNAL**

An internal error was detected by the Core Manager or by the API.

**DIVA\_ERR\_OBJECT\_DOESNT\_EXIST**

The specified Virtual Object does not exist in the DIVA Core Database.

**DIVA\_ERR\_SEVERAL\_OBJECTS**

More than one Virtual Object with the specified name exists in the DIVA Core Database.

See also [DIVA\\_archiveObject](#), [DIVA\\_restoreObject](#), and [DIVA\\_deleteObject](#).

## **DIVA\_getPartialRestoreRequestInfo**

When processing the request `DIVA_PartialRestoreObject()`, and the format for the offsets was specified as timecodes, the offsets that are actually used may differ (somewhat) from what was specified in the request. Once the Partial File Restore request is complete, you can use this command to obtain the actual offsets of the restored files.

This is a special purpose command that is valid only as follows:

- The request number to be queried must be a partial file restore request that has been successfully completed.
- The format specified in the partial file restore request must be a timecode type. This command is therefore not valid when the format of the request was folder-based or DPX.

## Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_getPartialRestoreRequestInfo (  
IN int requestNumber,  
OUT vector <DIVA_OFFSET_SOURCE_DEST> *fileList  
);
```

## requestNumber

Identifies the completed Partial File Restore request to be queried.

## fileList

List of the files of an Virtual Object that have been partially restored. Each structure contains the Source Server file name, a vector of the offsets used for the transfer, and a Destination Server file name. This vector must be similar to the vector provided to the `DIVA_partialRestoreObject()` function in terms of files and offset pairs. This function is provided to eventually detect that the actual offsets used for the transfer to the Destination Server have been adapted based on the format of the data to transfer.

## Return Values

One of these `DIVA_STATUS` constants defined in `DIVAapi.h`:

### DIVA\_OK

The request was correctly submitted and accepted by the Core Manager.

### DIVA\_ERR\_NOT\_CONNECTED

No connection is open.

### DIVA\_ERR\_SYSTEM\_IDLE

The DIVA Core system cannot accept connections and queries.

### DIVA\_ERR\_BROKEN\_CONNECTION

The connection with the Core Manager was broken.

## DIVA\_ERR\_TIMEOUT

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the `DIVA_API_TIMEOUT` variable and equals one hundred-eighty (180) seconds by default.

## DIVA\_ERR\_UNKNOWN

An unknown status was received from the Core Manager.

## DIVA\_ERR\_INTERNAL

An internal error was detected by the Core Manager or by the API.

## DIVA\_ERR\_NO\_SUCH\_REQUEST

The `requestNumber` identifies no request.

## DIVA\_ERR\_INVALID\_PARAMETER

The `requestNumber` identifies no completed partial file restore request.

See also [DIVA\\_partialRestoreObject](#) and [DIVA\\_getRequestInfo](#).

# DIVA\_getRequestInfo

Obtains information about an archive, restore, delete, or repack request.

## Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_getRequestInfo (
    IN int requestNumber,
    OUT DIVA_REQUEST_INFO *requestInfo
);
```

## requestNumber

Identifies the queried request.

## requestInfo

Pointer to a `DIVA_REQUEST_INFO` structure. This is allocated and deleted by the caller.

```
class DIVA_REQUEST_INFO {
public:
    int requestNumber;
    DIVA_REQUEST_TYPE requestType;
    DIVA_REQUEST_TYPE requestState;
    DIVA_REQUEST_STATE requestState;
    int progress;
```

```
DIVA_ABORTION_REASON      abortionReason;  
DIVA_OBJECT_SUMMARY      objectSummary;  
DIVA_REPACK_TAPES_INFO    repackTapes;  
int                       currentPriority;  
DIVA_STRING               additionalInfo;  
time_t                   submissiondate  
time_t                   completiondate  
};
```

## **requestNumber**

The DIVA Core request number.

## **requestType**

See the definition of DIVA\_REQUEST\_TYPE later in this section.

## **requestState**

See the definition of DIVA\_REQUEST\_STATE later in this section.

## **progress**

The progress of the request from zero to one hundred percent if the requestState is DIVA\_TRANSFERRING or DIVA\_MIGRATING.

## **abortionReason**

The reason the request was terminated if the requestState is DIVA\_ABORTED, otherwise this is zero.

## **objectSummary**

See the definition of DIVA\_OBJECT\_SUMMARY later in this section.

## **repackTapes**

Used if the requestType is REPACK.

## **additionalInfo**

See [Additional\\_Info](#) later in this section for use of this field.

## **submissionDate**

The date and time the request was submitted. This is UTC time in seconds (that is, seconds since January 1, 1970).

## completionDate

The date and time the request completed. This is UTC time in seconds and will be -1 if the request is still processing.

```
typedef enum {  
    DIVA_ARCHIVE_REQUEST = 0,  
    DIVA_RESTORE_REQUEST,  
    DIVA_DELETE_REQUEST,  
    DIVA_EJECT_REQUEST,  
    DIVA_INSERT_REQUEST,  
    DIVA_COPY_REQUEST,  
    DIVA_COPY_TO_NEW_REQUEST,  
    DIVA_RESTORE_INSTANCE_REQUEST,  
    DIVA_DELETE_INSTANCE_REQUEST,  
    DIVA_UNKNOW_REQUEST_TYPE,  
    DIVA_AUTOMATIC_REPACK_REQUEST,  
    DIVA_ONDEMAND_RAPACK_REQUEST,  
    DIVA_ASSOC_COPY_REQUEST,  
    DIVA_PARTIAL_RESTORE_REQUEST,  
    DIVA_MULTIPLE_RESTORE_REQUEST,  
    DIVA_TRANSCODE_ARCHIVED_REQUEST,  
    DIVA_EXPORT_REQUEST,  
    DIVA_TRANSFER_REQUEST,  
    DIVA_AUTOMATIC_VERIFY_TAPES_REQUEST,  
    DIVA_MANUAL_VERIFY_TAPES_REQUEST,  
} DIVA_REQUEST_TYPE ;
```

```
typedef enum {  
    DIVA_PENDING = 0,  
    DIVA_TRANSFERRING,  
    DIVA_MIGRATING,  
    DIVA_COMPLETED,  
    DIVA_ABORTED,  
    DIVA_CANCELLED,  
    DIVA_UNKNOWN_STATE,  
    DIVA_DELETING,  
    DIVA_WAITING_FOR_RESOURCES,  
    DIVA_WAITING_FOR_OPERATOR,  
    DIVA_ASSIGNING_POOL,  
    DIVA_PARTIALLY_ABORTED,  
    DIVA_RUNNING  
} DIVA_REQUEST_STATE;
```

```
typedef enum {  
    DIVA_AR_NONE = 0,  
    DIVA_AR_DRIVE,  
    DIVA_AR_TAPE,  
    DIVA_AR_ACTOR,  
    DIVA_AR_DISK,  
    DIVA_AR_DISK_FULL,  
    DIVA_AR_SOURCE_DEST,  
    DIVA_AR_RESOURCES,  
    DIVA_AR_LIBRARY,  
    DIVA_AR_PARAMETERS,  
    DIVA_AR_UNKNOWN,  
    DIVA_AR_INTERNAL,
```

```
DIVA_AR_SOURCE_DEST2  
} DIVA_ABORTION_CODE;
```

## **DIVA\_AR\_NONE = 0**

Request not terminated.

## **DIVA\_AR\_DRIVE**

Drive trouble

## **DIVA\_AR\_TAPE**

Tape trouble

## **DIVA\_AR\_ACTOR**

Actor trouble

## **DIVA\_AR\_DISK**

Disk trouble

## **DIVA\_AR\_DISK\_FULL**

The disk is full.

## **DIVA\_AR\_SOURCE\_DEST**

Server trouble

## **DIVA\_AR\_RESOURCES**

Resource attribution trouble

## **DIVA\_AR\_LIBRARY**

Managed Storage trouble

## **DIVA\_AR\_PARAMETERS**

Incorrect request parameters

## **DIVA\_AR\_UNKNOWN**

Unknown code

## **DIVA\_AR\_INTERNAL**

Internal Core Manager error

## DIVA\_AR\_SOURCE\_DEST2

This parameter has been deprecated but left intact for software compatibility.

```
class DIVA_ABORTION_REASON {
public:
    DIVA_ABORTION_CODE code;
    string description;
};

class DIVA_OBJECT_SUMMARY {
public:
    string      objectName;
    string      objectCollection ;
};
```

### objectName

The name of the Virtual Object.

### objectCollection

The Collection of the Virtual Object.

## Return Values

One of these DIVA\_STATUS constants defined in DIVAapi.h:

### DIVA\_OK

The request was correctly submitted and accepted by the Core Manager.

### DIVA\_ERR\_NOT\_CONNECTED

No connection is open.

### DIVA\_ERR\_SYSTEM\_IDLE

The DIVA Core system cannot accept connections and queries.

### DIVA\_ERR\_BROKEN\_CONNECTION

The connection with the Core Manager was broken.

### DIVA\_ERR\_TIMEOUT

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the DIVA\_API\_TIMEOUT variable and equals one hundred-eighty (180) seconds by default.

## DIVA\_ERR\_UNKNOWN

An unknown status was received from the Core Manager.

## DIVA\_ERR\_INTERNAL

An internal error was detected by the Core Manager or by the API.

## DIVA\_ERR\_NO\_SUCH\_REQUEST

The requestNumber identifies no request.

## Additional\_Info

The Additional\_Info field of the DIVA\_REQUEST\_INFO structure can contain one or more of the following depending on the request type:

### MOB ID

MOB ID is a unique Virtual Object identifier generated and used by AVID software. The API provides the interface to retrieve the MOB ID for third party vendors after restoring archived Virtual Objects to Unity. The MOB ID is available in the additionalInfo field of the DIVA\_REQUEST\_INFO structure. The MOB ID can be retrieved only when the Virtual Object is restored to the AVID Unity system.

#### Example MOB ID:

060c2b34020511010104100013-000000-002e0815d552002b-060e2b347f7f-2a80

## XML Document

Depending on the type of request the XML document may be empty, or it may contain any combination of the following elements. See the schema additionalInfoRequestInfo.xsd found in the program\Common\schemas folder of the DIVA Core installation.

When the request was a Restore, N-Restore, Partial File Restore, Copy, or Copy To New the list of media that contains the requested Virtual Object is provided as follows:

```
<ADDITIONAL_INFO xmlns="http://www.telestream.net/divacore/
additionalInfoRequestInfo/v1.0"> <Object>
  <Name>Object Name</Name>
  <Collection>Collection</Collection>
  <Instances>
    <DiskInstance>
      <Id>0</Id>
      <Disk>
        <MediaName>disk name</MediaName>
      </Disk>
    </DiskInstance>
    <TapeInstance>
      <Id>1</Id>
      <Tape>
```

```

        <MediaName>barcode</MediaName>
    </Tape>
</TapeInstance>
</Instances>
</Object>
</ADDITIONAL_INFO>

```

The following is included when the request was a Multiple Restore. If the restore is OK for one of the Destination Servers, but NOT OK for another, the Request State Parameter is DIVA\_PARTIALLY\_ABORTED and the Request Abortion Code is DIVA\_AR\_SOURCE\_DEST. The status of each Destination Server is as follows:

```

<ADDITIONAL_INFO xmlns="http://www.telestream.net/divacore/
additionalInfoRequestInfo/v1.0">
    <request id="12345" type="Restore">
        <destination name="destination name one" success="true"/>
        <destination name="destination name two" success="false"/>
    </request>
</ADDITIONAL_INFO>

```

The ClipID is included when the request was for a restore to a Quantel device. An ISA gateway never overwrites clips. A new ClipID is created for every imported clip. The ClipID of the created clip will be supplied after the Transfer Complete message as follows:

226 Transfer Complete. [new ClipID]

The Actor captures this new ClipID after the transfer and forwards it to the Core Manager. To use the API, DIVA\_GetRequestInfo must be called. If the request is completed, the new ClipID will be in the Additional Request Information field as follows:

```

<ADDITIONAL_INFO xmlns="http://www.telestream.net/divacore/
additionalInfoRequestInfo/v1.0">
    <ClipID>98765</ClipID>
</ADDITIONAL_INFO>

```

## DIVA\_getSourceDestinationList

This function returns a list of Source Servers present in a particular DIVA Core System.

### Synopsis

```
#include "DIVAapi.h"
```

```

DIVA_STATUS
DIVA_getSourceDestinationList (
    IN string                                options;
    OUT vector<DIVA_ACTOR_INSTANCE_DESC>    *arraysInfo
)

```

### arraysInfo

Pointer to a list of DIVA\_SOURCE\_DESTINATION\_LIST structures.

```

#ifndef WIN32
typedef long long __int64;
#endif

typedef enum {
    DIVA_SOURCE_TYPE_UNKNOWN = 0,
    DIVA_SOURCE_TYPE_MSS,
    DIVA_SOURCE_TYPE_PDR,
    DIVA_SOURCE_TYPE_SEACHANGE_BMC,
    DIVA_SOURCE_TYPE_SEACHANGE_BML,
    DIVA_SOURCE_TYPE_SEACHANGE_FTP,
    DIVA_SOURCE_TYPE_LEITCH,
    DIVA_SOURCE_TYPE_FTP_STANDARD,
    DIVA_SOURCE_TYPE_SFTP,
    DIVA_SOURCE_TYPE_DISK,
    DIVA_SOURCE_TYPE_LOCAL,
    DIVA_SOURCE_TYPE_CIFS,
    DIVA_SOURCE_TYPE_SIMULATION,
    DIVA_SOURCE_TYPE_OMNEON,
    DIVA_SOURCE_TYPE_MEDIAGRID,
    DIVA_SOURCE_TYPE_AVID_DHM,
    DIVA_SOURCE_TYPE_AVID_DET,
    DIVA_SOURCE_TYPE_AVID_AMC,
    DIVA_SOURCE_TYPE_QUANTEL_ISA,
    DIVA_SOURCE_TYPE_QUANTEL_QCP,
    DIVA_SOURCE_TYPE_SONY_HYPER_AGENT,
    DIVA_SOURCE_TYPE_METASOURCE,
    DATA_SOURCE_TYPE_MOVIETOME,
    DATA_SOURCE_TYPE_EXPEDAT,
    DATA_SOURCE_TYPE_AVID_DIRECT
} DIVA_SOURCE_TYPE;

class DIVA_SOURCE_DESTINATION_LIST{
public:
    DIVA_STRING server_Address;
    DIVA_STRING server_ConnectOption;
    int server_MaxAccess;
    int server_MaxReadAccess;
    __int64 server_MaxThroughput;
    int server_MaxWriteAccess;
    DIVA_STRING server_Name;
    DIVA_STRING server_ProductionSystem;
    DIVA_STRING server_RootPath;
    DIVA_SOURCE_TYPE server_SourceType;
};

```

**server\_Address**

The server IP address.

**server\_ConnectOption**

The server connection options.

**server\_MaxAccess**

The server maximum number of accesses.

**server\_MaxReadAccess**

The server maximum number of read accesses.

**server\_MaxThroughput**

The server maximum throughput.

**server\_MaxWriteAccess**

The server maximum write access.

**server\_Name**

The server name.

**Server\_ProductionSystem**

The server Network name.

**server\_RootPath**

The server root path.

**server\_SourceType**

The Source Server type.

**Return Values**

One of these DIVA\_STATUS constants defined in DIVAapi.h:

**DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

**DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

**DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the DIVA\_API\_TIMEOUT variable and equals one hundred-eighty (180) seconds by default.

## DIVA\_ERR\_UNKNOWN

An unknown status was received from the Core Manager.

## DIVA\_ERR\_INTERNAL

An internal error was detected by the Core Manager or by the API.

# DIVA\_getStoragePlanList

This function returns the list of Storage Plan Names that are defined in the DIVA Core system.

## Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS          DIVA_getStoragePlanList (
IN string            options;
OUT vector<DIVA_STRING> *&spList
);
```

## spList

A pointer to a list of Storage Plan Names.

## Return Values

One of these DIVA\_STATUS constants defined in DIVAapi.h:

## DIVA\_OK

The request was correctly submitted and accepted by the Core Manager.

## DIVA\_ERR\_NOT\_CONNECTED

No connection is open.

## DIVA\_ERR\_SYSTEM\_IDLE

The DIVA Core system cannot accept connections and queries.

## DIVA\_ERR\_BROKEN\_CONNECTION

The connection with the Core Manager was broken.

## DIVA\_ERR\_TIMEOUT

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the DIVA\_API\_TIMEOUT variable and equals one hundred-eighty (180) seconds by default.

## DIVA\_ERR\_UNKNOWN

An unknown status was received from the Core Manager.

## DIVA\_ERR\_INTERNAL

An internal error was detected by the Core Manager or by the API.

# DIVA\_getTapeInfo

Returns detailed information about a given tape identified by its barcode.

## Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_getTapeInfo (
    IN DIVA_STRING barcode,
    OUT DIVA_DETAILED_TAPE_DESC *tapeInfo
);
```

## barcode

The barcode of the tape for which information is to be returned.

## tapeInfo

The returned information.

```
class DIVA_DETAILED_TAPE_DESC {
public:
    string      vsn;
    int         setID;
    string      group;
    int         typeID;
    string      type;
    int         fillingRatio;
    int         fragmentationRatio;
    __int64     remainingSize;
    __int64     totalSize;
    bool        isInserted;
    string      externalizationComment;
    bool        isGoingToBeRepacked;
    int         mediaFormatId;
};
```

## setID

Tape Set ID

## typeID

Tape Type ID

**type**

Tape Type Name

**fillingRatio**

The tape filling ratio using the equation:

$\text{last\_written\_block} / \text{total\_block\_count}$ .

**fragmentationRatio**

The tape fragmentation ration using the equation:

$1 - (\text{valid\_blocks\_count}) / (\text{last\_written\_block})$

Valid blocks are blocks used for archived Virtual Objects not currently deleted.

**mediaFormatId**

The format of the data on to be used. The value can be DIVA\_MEDIA\_FORMAT\_DEFAULT, DIVA\_MEDIA\_FORMAT\_LEGACY, DIVA\_MEDIA\_FORMAT\_AXF, or DIVA\_MEDIA\_FORMAT\_AXF\_10.

**Return Values**

One of these DIVA\_STATUS constants defined in DIVAapi.h:

**DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

**DIVA\_ERR\_NOT\_CONNECTED**

No connection is open.

**DIVA\_ERR\_SYSTEM\_IDLE**

The DIVA Core system cannot accept connections and queries.

**DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

**DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the DIVA\_API\_TIMEOUT variable and equals one hundred-eighty (180) seconds by default.

## DIVA\_ERR\_UNKNOWN

An unknown status was received from the Core Manager.

## DIVA\_ERR\_INTERNAL

An internal error was detected by the Core Manager or by the API.

## DIVA\_ERR\_TAPE\_DOESNT\_EXIST

There is no tape associated with the given barcode.

# DIVA\_insertTape

Submits an Insert request to DIVA Core. This request completes when the operator has entered the requested tapes into the Managed Storage. The application is responsible for managing which tapes must be entered.

## Synopsis

```
#include "DIVAapi.h"

DIVA_STATUS DIVA_insertTape (
IN bool      require,
IN int       priorityLevel,
OUT int      *requestNumber
)

DIVA_STATUS DIVA_insertTape (
IN bool      require,
IN int       priorityLevel,
IN int       acsId,
IN int       capId,
OUT int      *requestNumber
);
```

## require

When true, perform a DIVA\_require() on every instance located on the successfully inserted tapes.

## priorityLevel

The priority level for this request. The priorityLevel can be in the range zero to one hundred, or the value DIVA\_DEFAULT\_REQUEST\_PRIORITY. The value zero is the lowest priority and one hundred the highest priority.

There are six predefined values as follows:

- DIVA\_REQUEST\_PRIORITY\_MIN
- DIVA\_REQUEST\_PRIORITY\_LOW
- DIVA\_REQUEST\_PRIORITY\_NORMAL

- `DIVA_REQUEST_PRIORITY_HIGH`
- `DIVA_REQUEST_PRIORITY_MAX`
- `DIVA_DEFAULT_REQUEST_PRIORITY`

When the `DIVA_DEFAULT_REQUEST_PRIORITY` value is used, the Core Manager uses the default priority defined in the Core Manager configuration for the request.

Using a value either outside of the range of zero to one hundred, or predefined values yields a `DIVA_ERR_INVALID_PARAMETER` error.

### **acslId (second form only)**

The numeric ID of the ACS where the Insert operation must be executed.

When `acslId = -1` (default used for the first form), the Insert attempt will be performed in all known ACSs.

### **capId (second form only)**

The numeric ID of the CAP from where tapes will be inserted.

When `capId = -1` (default used for the first form), the Insert attempt will be performed in the first available CAP in the specified ACS.

### **requestNumber**

The number identifying the request.

### **Return Values**

One of these `DIVA_STATUS` constants defined in `DIVAapi.h`:

#### **DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

#### **DIVA\_ERR\_NOT\_CONNECTED**

No connection is open.

#### **DIVA\_ERR\_SYSTEM\_IDLE**

The DIVA Core system cannot accept connections and queries.

#### **DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

## DIVA\_ERR\_TIMEOUT

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the `DIVA_API_TIMEOUT` variable and equals one hundred-eighty (180) seconds by default.

## DIVA\_ERR\_UNKNOWN

An unknown status was received from the Core Manager.

## DIVA\_ERR\_INTERNAL

An internal error was detected by the Core Manager or by the API.

## DIVA\_ERR\_INVALID\_PARAMETER

A parameter value was not understood by the Core Manager.

## DIVA\_ERR\_CANNOT\_ACCEPT\_MORE\_REQUESTS

The count of simultaneous requests reached the maximum allowed value. This variable is set in the `manager.conf` configuration file. The default value is 300.

See also [DIVA\\_ejectTape](#).

## DIVA\_linkObjects

This function provides the opportunity to link together two existing Virtual Objects; parent and child. If the Virtual Objects are linked for Delete, anytime the parent Virtual Object is deleted, the child will also be deleted. If Virtual Objects are linked for Restore, anytime the parent Virtual Object is restored, the child will be restored to the original location from where the child Virtual Object was archived.

### Synopsis

```
#include "DIVAapi.h"

DIVA_STATUS DIVA_linkObjects (
    IN DIVA_STRING      parentName,
    IN DIVA_STRING      parentCollection,
    IN DIVA_STRING      childName,
    IN DIVA_STRING      childCollection,
    IN bool              cascadeDelete,
    IN bool              cascadeRestore
);
```

### parentName

The parent Virtual Object name.

**parentCollection**

The parent Virtual Object Collection.

**childName**

The child Virtual Object name.

**childCollection**

The child Virtual Object Collection.

**cascadeDelete**

Indicates if the child Virtual Object should be deleted along with parent.

**cascadeRestore**

Indicates if the child Virtual Object should be restored along with parent.

**Return Values**

One of these DIVA\_STATUS constants defined in DIVAapi.h:

**DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

**DIVA\_ERR\_OBJECT\_ALREADY\_EXISTS**

An Virtual Object with this name and Collection already exists in the DIVA Core system.

**DIVA\_ERR\_INVALID\_PARAMETER**

A parameter value was not understood by the Core Manager.

**DIVA\_ERR\_INTERNAL**

An internal error was detected by the Core Manager or by the API.

## DIVA\_lockObject

A call to this function will lock an Virtual Object. Locked Virtual Objects cannot be restored.

### Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_lockObject (  
IN DIVA_STRING      objectName,  
IN DIVA_STRING      Collection,  
IN string            options  
);
```

### objectName

The name of the Virtual Object.

### Collection

The Collection assigned to the Virtual Object when it was archived.

### options

Not currently in use.

### Return Values

One of these DIVA\_STATUS constants defined in DIVAapi.h:

#### DIVA\_OK

The request was correctly submitted and accepted by the Core Manager.

#### DIVA\_ERR\_NOT\_CONNECTED

No connection is open.

#### DIVA\_ERR\_SYSTEM\_IDLE

The DIVA Core system cannot accept connections and queries.

#### DIVA\_ERR\_BROKEN\_CONNECTION

The connection with the Core Manager was broken.

## DIVA\_ERR\_TIMEOUT

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the `DIVA_API_TIMEOUT` variable and equals one hundred-eighty (180) seconds by default.

## DIVA\_ERR\_UNKNOWN

An unknown status was received from the Core Manager.

## DIVA\_ERR\_INTERNAL

An internal error was detected by the Core Manager or by the API.

# DIVA\_multipleRestoreObject

Submits an Virtual Object Restore request to the Core Manager using several Destination Servers. The Core Manager chooses the appropriate instance to be restored. This function returns as soon as the Core Manager accepts the request.

The request will continue even if an error occurs with one of the Destination Servers. To check that the operation was successful the application must call the function `DIVA_getRequestInfo()`.

If `DIVA_MultipleRestoreObject()` is launched with a single Destination Server, the restore automatically converts to a `DIVA_RestoreObject()`.

## Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_MultipleRestoreObject (
    IN DIVA_STRING          objectName,
    IN DIVA_STRING          objectCollection,
    IN vector <DIVA_DESTINATION_INFO> destinations,
    IN DIVA_RESTORE_QOS     qualityOfService,
    IN int                  priorityLevel,
    IN DIVA_STRING          restoreOptions,
    OUT int                 *requestNumber
)
public typedef struct _DIVA_DESTINATION_INFO {
    DIVA_STRING destination;
    DIVA_STRING filePathRoot;
} DIVA_DESTINATION_INFO, *PDIVA_DESTINATION_INFO;
```

## objectName

The name of the Virtual Object to be restored.

## objectCollection

The Collection assigned to the Virtual Object when it was archived. This parameter can be a null string, however this may result in an error if several Virtual Objects have the same name.

## destinations

A list of available Destination Servers (for example, a video server or browsing server) where Virtual Object files can be restored. The names must be known by the DIVA Core configuration description.

A root folder where the Virtual Object files will be placed is associated with each Destination Server. If null (string("")), the files will be placed in the FILES\_PATH\_ROOT folder specified when archiving the Virtual Object using the DIVA\_archiveObject() function.

## qualityOfService

One of the following codes:

### DIVA\_QOS\_DEFAULT

Restoring is performed according to the default Quality Of Service (currently direct and cache for restore operations).

### DIVA\_QOS\_CACHE\_ONLY

Use cache restore only.

### DIVA\_QOS\_DIRECT\_ONLY

Use direct restore only - no disk instance is created.

### DIVA\_QOS\_CACHE\_AND\_DIRECT

Use cache restore if available, or direct restore if cache restore is not available.

### DIVA\_QOS\_DIRECT\_AND\_CACHE

Use direct restore if available, or cache restore if direct restore is not available.

### DIVA\_QOS\_NEARLINE\_ONLY

Use nearline restore only. Nearline restore will restore from a disk instance if a disk instance exists, otherwise, it will create a disk instance and restore from the newly created disk instance.

### DIVA\_QOS\_NEARLINE\_AND\_DIRECT

Use nearline restore if available, or direct restore if nearline restore is not available.

Additional and optional services are available. To request those services, use a logical OR between the previously documented Quality Of Service parameter and the following constant:

**DIVA\_RESTORE\_SERVICE\_DO\_NOT\_OVERWRITE**

Do not overwrite existing files on the Destination Server.

**priorityLevel**

The priority level for this request. The priorityLevel can be in the range zero to one hundred, or the value `DIVA_DEFAULT_REQUEST_PRIORITY`. The value zero is the lowest priority and one hundred the highest priority.

There are six predefined values as follows:

- `DIVA_REQUEST_PRIORITY_MIN`
- `DIVA_REQUEST_PRIORITY_LOW`
- `DIVA_REQUEST_PRIORITY_NORMAL`
- `DIVA_REQUEST_PRIORITY_HIGH`
- `DIVA_REQUEST_PRIORITY_MAX`
- `DIVA_DEFAULT_REQUEST_PRIORITY`

When the `DIVA_DEFAULT_REQUEST_PRIORITY` value is used, the Core Manager uses the default priority defined in the Core Manager configuration for the request.

Using a value either outside of the range of zero to one hundred, or predefined values yields a `DIVA_ERR_INVALID_PARAMETER` error.

**restoreOptions**

Additional options that must be used for performing the transfer of data from DIVA Core to the Destination Server. These options supersede any options specified in the DIVA Core configuration database. Currently the possible values for restoreOptions are:

- A null string to specify no Virtual Objects
- `-login` represents the log in required for some Source Servers. This option obsoletes the `-gateway` option in earlier releases.
- `-pass` represents the password used with the `-login` option for some Source Servers.

**requestNumber**

The request number assigned to this request. This number is used for querying the status or canceling the request.

**Return Values**

One of these `DIVA_STATUS` constants defined in `DIVAapi.h`:

**DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

**DIVA\_ERR\_NOT\_CONNECTED**

No connection is open.

**DIVA\_ERR\_SYSTEM\_IDLE**

The DIVA Core system cannot accept connections and queries.

**DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

**DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the `DIVA_API_TIMEOUT` variable and equals one hundred-eighty (180) seconds by default.

**DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

**DIVA\_ERR\_INTERNAL**

An internal error was detected by the Core Manager or by the API.

**DIVA\_ERR\_INVALID\_PARAMETER**

A parameter value was not understood by the Core Manager.

**DIVA\_ERR\_CANNOT\_ACCEPT\_MORE\_REQUESTS**

The count of simultaneous requests reached the maximum allowed value. This variable is set in the `manager.conf` configuration file. The default is 300.

**DIVA\_ERR\_OBJECT\_DOESNT\_EXIST**

The specified Virtual Object does not exist in the DIVA Core database.

**DIVA\_ERR\_OBJECT\_OFFLINE**

There is no inserted instance in the Managed Storage and no Actor could provide a disk instance.

**DIVA\_ERR\_SEVERAL\_OBJECTS**

More than one Virtual Object with the specified name exists in the DIVA Core database.

## DIVA\_ERR\_OBJECT\_IN\_USE

The Virtual Object is currently in use (*for example, Archived, Restored, Deleted, and so on*).

## DIVA\_ERR\_SOURCE\_OR\_DESTINATION\_DOESNT\_EXIST

The specified Server is unknown by the DIVA Core system.

## DIVA\_ERR\_OBJECT\_PARTIALLY\_DELETED

The specified Virtual Object has instances that are partially deleted.

See also [DIVA\\_restoreObject](#), [DIVA\\_getRequestInfo](#), and [DIVA\\_copyToGroup and DIVA\\_copy](#).

## DIVA\_partialRestoreObject

Submits a Partial Virtual Object Restore request to the Core Manager and the Core Manager chooses the appropriate instance to be restored. This function returns as soon as the Core Manager accepts or rejects the request. To check that the operation was successful the application must call the `DIVA_getRequestInfo()` function.

If the request was not accepted (for example, if the requested Virtual Object is on media not currently available) the request will generate an error. The media names (tape barcodes and disk names) that contain instances of the Virtual Object are included in the `additionalInfo` field of the `DIVA_getRequestInfo()` response.

The Core Manager will use the `instanceID` field to select the instance of the Virtual Object to use for the Partial Restore operation. The Core Manager will choose an appropriate instance to restore if `DIVA_ANY_INSTANCE` is used.

DIVA Core supports four types of Partial Restore. The type implemented is determined by the `format` parameter in the request.

The following describes each type of Partial Virtual Object Restore:

### Byte Offset

The `format` equals `DIVA_FORMAT_BYTES` and provides for a range of bytes to be extracted from a particular file in the archive. For example, you can extract bytes 1 to 2000 (the first 2000 bytes of the file), or byte 5000 to the end of the file (or both) and store them to an output file such as `movie.avi`.

The result of the Byte Offset Partial Restore is usually not playable when applied to video files. Actor will not apply the header, footer, and so on, according to the video format.

To issue a Byte Offset Partial Restore, pass `DIVA_FORMAT_BYTES` in the `format` field of the request. Create a `DIVA_OFFSET_SOURCE_DEST` Virtual Object (in the `fileList` parameter of the request). In the Virtual Object you must specify the `sourceFile` in the archive and name the output file (`destFile`). One or more `DIVA_OFFSET_PAIR` Virtual

Objects must be inserted within the DIVA\_OFFSET\_SOURCE\_DEST Virtual Object. These offset Virtual Objects contain the ranges of bytes to be restored to the output file. The fileFolder and range fields within the DIVA\_OFFSET\_SOURCE\_DEST Virtual Object do not need to be populated.

**Example:**

start=10000 end=50000

## Timecode

The format equals DIVA\_FORMAT\_VIDEO\_\* and provides for a selected portion of a particular media file based on timecode. For example, you could extract from 00:00:04:00 to 00:10:04:00 (a 10 minute segment starting 4 seconds in and ending at 10 minutes and 4 seconds) and place that segment into an output file such as movie.avi. The file is a smaller version of the original movie file.

The result of the Timecode Partial Restore is a valid clip when applied to video files. Actor will apply the header, footer, and so on, according to the video format. The request will be terminated if the Actor cannot parse the format. This type of Partial Restore can only be applied to a valid video clip.

To issue a Timecode Partial Restore populate the format field in the request with the format of the file being partially restored. For example, if the file being restored is a GXF file, specify a value of DIVA\_FORMAT\_VIDEO\_GXF in the format field of the request. DIVA Core provides an auto-detect feature that works for many types of media. Specify DIVA\_FORMAT\_AUTODETECT in the format field to use auto-detect.

Create a DIVA\_OFFSET\_SOURCE\_DEST Virtual Object in the fileList parameter of the request. In this Virtual Object, add a DIVA\_OFFSET\_PAIR Virtual Object using the offsetVector parameter that contains the start and end time. Use DIVA\_OFFSET\_TC\_END to indicate the final timecode in the media file. The fileFolder and range fields within the DIVA\_OFFSET\_SOURCE\_DEST Virtual Object do not need to be populated.

**Example:**

start=01:01:01:00 end=02:02:02:00

## Files and Folders

---

**Caution:** In the following process The offsetVector, sourceFile, destFile, and range parameters should not be specified for the Files and Folders Partial Virtual Object restore type.

---

The format equals DIVA\_FORMAT\_FOLDER\_BASED and provides for extracting entire files from the archive, or extracting entire directories and their contents. In DIVA Core you can extract multiple files and directories in the same request. The files are restored with the file names and path names that were specified in the archive. No renaming option is valid in Files and Folders Partial Restore. For example, a file archived as misc/

12-2012/movie.avi would be partially restored to a misc/12-2012 subdirectory with the name movie.avi.

When a folder is specified in a Files and Folders Partial Restore, the folder and all files within that folder are restored. Each directory to be restored can have the -r option to recursively restore all folders nested within the target folder.

To issue a Files and Folders Partial Restore, the format field in the request must be populated with the `DIVA_FORMAT_FOLDER_BASED` value. Create a `DIVA_OFFSET_SOURCE_DEST` Virtual Object in the `fileList` parameter of the request. In the Virtual Object add a `DIVA_FILE_FOLDER` Virtual Object in the `fileFolder` parameter containing the name of the file or folder to be restored, and any options (such as the recursive option) for that directory.

## DPX

The format equals `DIVA_FORMAT_DPX` and provides for extracting a range of DPX files from the archive. In this type of restore, the entire Virtual Object is viewed as a single media item. One DPX file represents one frame of media. Only .dpx, .tif, and .tiff files in the archive are considered frames for the purposes of this command.

The first .dpx, .tif, or .tiff file in the archived Virtual Object is considered Frame 1, the second .dpx in the archive is Frame 2, and so on.

For example, if you extract frame 10 through frame 15 using DPX Partial Restore, it would restore the 10th .dpx file that appears in the archive, through (and including) the 15th .dpx file, resulting in six total files. Any other files (such as .wav files) are skipped by DPX Partial Restore.

Special frame numbers 0 and -1 may be used to refer to the first and last frame respectively. Frame 0 is valid as the start of a frame range and Frame -1 is valid as the end of a range.

Valid frames and ranges are as follows:

- Frame 0 = first frame
- Frame 1 = the first frame in the sequence.
- Frame n = the n<sup>th</sup> frame in the sequence.
- Frame -1 = last frame

Specifying frame 0 as the last frame is invalid.

Specifying Frame 0 to 0 is invalid and will not return the first frame as you have intended.

Specifying Frame 0 to 1 or Frame 1 to 1 will return the first frame.

Specifying the Frame -1 in the first frame produces an error. If the frame number of the last frame is unknown, you cannot specify Frame -1 to -1 to return the exact last frame.

**Examples:****start=0 - end=1**

This will restore only the first frame.

**start=600 - end=635, start=679 - end=779**

This will restore frames 600 through 635, and frames 679 through 779.

**start=810 - end=-1**

This will restore all frames from frame 810 to the end of the archive.

---

**Caution: In the following process the `offsetVector`, `sourceFile`, `destFile`, and `fileFolder` parameters should not be specified for the DPX Partial Virtual Object restore type.**

---

To issue a DPX Partial Restore you populate the format field in the request with the value `DIVA_FORMAT_DPX`. Create a `DIVA_OFFSET_SOURCE_DEST` Virtual Object in the `fileList` parameter of the request. In this Virtual Object, you add a `DIVA_RANGE` Virtual Object in the range parameter that contains the start and end frames of the range to be restored.

To specify another range of frames within the same request, another `DIVA_OFFSET_SOURCE_DEST` Virtual Object should be added to the request in the same manner.

The actual file name may, or may not, match the frame number in DIVA Core. During the restore process DIVA Core interrogates the archive, finds the file order, and determines the frame number from the resulting file order. It does not consider the file name. The first `.dpx`, `.tif`, or `.tiff` file found is considered frame 1.

You must be careful when archiving DPX files to ensure they can be partially restored properly, in part because DPX Partial Restore does not examine the file name or the DPX header information to determine which file is assigned to which frame. The assignment is based purely on the order in which the `.dpx` files appear in the archive. By default, the ordering is established by the Source Server and is typically alphanumeric. For example, NTFS DISK Servers order files and folders case insensitively as a general rule except where diacritical marks such as `'`, ```, `^`, and so on are applied.

By default, when DIVA Core encounters a subfolder it recursively processes all of the children of that folder before continuing with other files. If a folder appears in the alphanumeric folder listing it is archived recursively in the order that it appears.

However, this can create some issues. For example, if you want all of the subdirectories of a given directory processed first, followed by the files in the directory, or you might want all files processed first and then subdirectories. The Actor allows the archive options `-file_order DIRS_FIRST` or `-file_order FILES_FIRST` to address these issues.

DPX Partial Restore looks at the entire Virtual Object as a single piece of media. If multiple reels or clips appear in an archive they can be stored in folders and partially restored through a Files and Folders Partial Restore. However, they will be viewed as

one long movie clip to DPX Partial Restore. If this is desired, ensure that the directories are sorted alphanumerically in the order the frames should be arranged.

DIVA Core does not perform any special audio handling for DPX media other than what might be embedded in DPX files themselves. DIVA Core supports transcoding of DPX media; however a transcoder may change the file names and (or) file order of the DPX archive.

## Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_SPEC DIVA_partialRestoreObject (
IN string                                     objectName,
IN string                                     objectCollection,
IN int                                         instanceID,
IN vector <DIVA_OFFSET_SOURCE_DEST>          fileList,
IN string                                     destination,
IN string                                     filesPathRoot,
IN DIVA_RESTORE_QOS                          qualityOfService,
IN int                                         priorityLevel,
IN string                                     restoreOptions,
IN DIVA_FORMAT                               format,
OUT int                                       *requestNumber
);
```

## objectName

The name of the Virtual Object to be partially restored.

## objectCollection

Collection assigned to the Virtual Object when it was archived. This parameter can be a null string, which can result in an error if several Virtual Objects have the same name.

## instanceID

The ID of a non-spanned tape instance or DIVA\_ANY\_INSTANCE.

## filelist

List of the files of the Virtual Object to be partially restored. Each structure contains the Source Server file name, a vector of offset pairs, and a Destination Server file name. The same source file can be used in several structures, but Destination Server files must be unique. A file present in the Virtual Object cannot be in any structure or it won't be restored.

## destination

Destination Server (for example, a video server or browsing server) to put the Virtual Object files. This name must be known by the DIVA Core configuration description.

## filePathRoot

The root folder on the Destination Server where the Virtual Object files will be placed. If this is null (string("")), the files will be placed in the FILES\_PATH\_ROOT folder specified when archiving the Virtual Object using the DIVA\_archiveObject() function.

## qualityOfService

One of the following codes:

### DIVA\_QOS\_DEFAULT

Restoring is performed according to the default Quality Of Service (currently direct restore).

### DIVA\_QOS\_CACHE\_ONLY (-qos\_cache\_only)

Use cache restore only.

### DIVA\_QOS\_DIRECT\_ONLY (-qos\_direct\_only)

Use direct restore only.

### DIVA\_QOS\_CACHE\_AND\_DIRECT (-qos\_cache\_and\_direct)

Use cache restore if available, or direct restore if cache restore is not available.

### DIVA\_QOS\_DIRECT\_AND\_CACHE (-qos\_direct\_and\_cache)

Use direct restore if available, or cache restore if direct restore is not available.

Additional and optional services are available. To request those services, use a logical OR between the previously documented Quality Of Service parameter and the following constant:

### DIVA\_RESTORE\_SERVICE\_DO\_NOT\_OVERWRITE

Do not overwrite existing files on the Destination Server.

## priorityLevel

The priority level for this request. The priorityLevel can be in the range zero to one hundred, or the value DIVA\_DEFAULT\_REQUEST\_PRIORITY. The value zero is the lowest priority and one hundred the highest priority.

There are six predefined values as follows:

- DIVA\_REQUEST\_PRIORITY\_MIN
- DIVA\_REQUEST\_PRIORITY\_LOW
- DIVA\_REQUEST\_PRIORITY\_NORMAL
- DIVA\_REQUEST\_PRIORITY\_HIGH
- DIVA\_REQUEST\_PRIORITY\_MAX

- **DIVA\_DEFAULT\_REQUEST\_PRIORITY**

When the **DIVA\_DEFAULT\_REQUEST\_PRIORITY** value is used, the Core Manager uses the default priority defined in the Core Manager configuration for the request.

Using a value either outside of the range of zero to one hundred, or predefined values yields a **DIVA\_ERR\_INVALID\_PARAMETER** error.

## restoreOptions

Additional options that must be used for performing the transfer of data from DIVA Core to the Destination Server. These options supersede any options specified in the DIVA Core configuration database. Currently the possible values for **restoreOptions** are:

- A null string to specify no Virtual Objects
- **-do\_not\_overwrite** executes this additional service
- **-do\_not\_check\_existence** executes this additional service
- **-delete\_and\_write** executes this additional service
- **-login** represents the log in required for some Source Servers. This option obsoletes the **-gateway** option in earlier releases.
- **-pass** represents the password used with the **-login** option for some Source Servers.

## format

### DIVA\_FORMAT\_BYTES

Offsets must be given as byte offsets. When the **offsetVector** field of a **DIVA\_OFFSET\_SOURCE\_DEST** structure contains more than one **DIVA\_OFFSET\_PAIR** element, every corresponding extract is concatenated to create the Destination Server file.

### DIVA\_FORMAT\_BYTES\_HEADER

This has been deprecated but left for compatibility purposes only.

### DIVA\_FORMAT\_VIDEO\_GXF

Offsets must be given as timecodes, and the file to be partially restored must be in GXF format.

The **fileList** vector parameter must contain only one **DIVA\_OFFSET\_SOURCE\_DEST** element.

The **offsetVector** vector parameter must contain only one **DIVA\_OFFSET\_PAIR** element.

Only the **DIVA\_QOS\_DIRECT\_ONLY** Quality Of Service is supported for this format.

**DIVA\_FORMAT\_VIDEO\_SEA**

Offsets must be given as timecodes. The file to be partially restored must be in SAF format and provide an index file.

A part description then contains one `DIVA_OFFSET_SOURCE_DEST` structure for each WAV file of the clip. There must be at least one WAV file per clip part.

- The Source Server file name in each structure must have the .wav or the .WAV extension.
- Each structure must contain exactly one `DIVA_OFFSET_PAIR` structure with a timecode pair equal to the timecode pair associated with the AVI file.
- The next part is delimited by the first `DIVA_OFFSET_SOURCE_DEST` structure associated with an AVI file.
- The Destination Server must support the successive restore of each part, with the AVI file (without WAV file) and then of the WAV files all at once in the same connection session.

**DIVA\_FORMAT\_VIDEO\_MPEG2\_TS**

Offsets must be given as timecodes. The video file must be encoded using the MPEG2 Transport Stream format. Use this for VELA encoders.

**DIVA\_FORMAT\_VIDEO\_MXF**

Offsets must be given as timecodes. The file format expected by this type of Partial File Restore is a single MXF file. A detailed matrix of supported MXF files is given in the product description.

**DIVA\_FORMAT\_VIDEO\_PINNACLE**

Offsets must be given as timecodes. This Partial File Restore format expects a specific Virtual Object structure. This is applicable to Pinnacle clips composed of three files (header, ft, and std). DIVA Core prefers the MSS Server type for creating this clip.

The `fileList` vector parameter must contain only one `DIVA_OFFSET_SOURCE_DEST` element. The `offsetVector` vector must contain only one `DIVA_OFFSET_PAIR` element. The `DIVA_OFFSET_SOURCE_DEST` element must be associated with the header file only. The Destination Server name is also the header.

**DIVA\_FORMAT\_VIDEO\_OMNEON**

Offsets must be given as timecodes. You can use this type of Partial File Restore to partially restore QuickTime files (referenced and self-contained clips are supported). A detailed matrix of supported QuickTime clips is given in the product description.

The `fileList` vector parameter must contain only one `DIVA_OFFSET_SOURCE_DEST` element. The `offsetVector` vector must contain only one `DIVA_OFFSET_PAIR` element. The `DIVA_OFFSET_SOURCE_DEST` element must be associated with the .mov file only if it's not a self-contained clip.

**DIVA\_FORMAT\_VIDEO\_LEITCH**

Offsets must be given as timecodes. The video file must be encoded using the LEITCH Video Server and the format is LXF.

**DIVA\_FORMAT\_VIDEO\_QUANTEL**

Offsets must be given as timecodes. You can use this type of Partial File Restore to partially restore Quantel clips that have been archived with a QUANTEL\_QCP Server type.

**DIVA\_FORMAT\_AUTODETECT**

Offsets must be given as timecodes. This type of Partial File Restore can detect video clips with the following archive formats:

- QuickTime self-contained
- QuickTime with referenced media files (the .mov file must be in the first position)
- DIF + WAV files
- AVI with audio interleaved (separated WAV is not currently supported)
- MXF (self-contained)
- MPEG PS
- LXF
- Seachange (the .pd file must be in the first position)

The fileList vector parameter must contain only one DIVA\_OFFSET\_SOURCE\_DEST element. The offsetVector vector must contain only one DIVA\_OFFSET\_PAIR element. The DIVA\_OFFSET\_SOURCE\_DEST element must be associated with the following:

- The .mov file if it is a QuickTime clip.
- The .dif file if it is a DV file.
- The .avi file if it is an AVI clip.

**DIVA\_FORMAT\_FOLDER\_BASED**

Specifies a set of files and folders to be restored. You can set a recursive flag to restore subfolders. All specified files and folders are restored.

**DIVA\_FORMAT\_DPX**

Specifies a set of intervals, frame X through frame Y, where frames are sorted and traversed alphanumerically.

Only files with .tif or .tiff data formats are supported. All files must have a .dpx extension. The first frame of a DPX Virtual Object is Frame 1. You can use frame numbers 0 and -1 to refer to the first and last frame respectively.

## requestNumber

The request number assigned to this request. This number is used for querying the status or canceling this request.

```
class DIVA_OFFSET_SOURCE_DEST {
public:
    DIVA_STRING          sourceFile;
    vector<DIVA_OFFSET_PAIR> offsetVector;
    DIVA_STRING          destFile;
    DIVA_FILE_FOLDER     fileFolder;
    DIVA_RANGE           range;
};
```

## sourceFile

The Source Server file name when the format is other than DIVA\_FORMAT\_FOLDER\_BASED or DIVA\_FORMAT\_DPX.

## offsetVector

The vector of intervals to restore. The type of all offsets in all DIVA\_OFFSET\_SOURCE\_DEST structures must be compliant with the format parameter of the Partial File Restore request. Valid only when the format is other than DIVA\_FORMAT\_FOLDER\_BASED or DIVA\_FORMAT\_DPX.

## destFile

The file name to be used at the Destination Server. Valid only when format is other than DIVA\_FORMAT\_FOLDER\_BASED or DIVA\_FORMAT\_DPX.

## fileFolder

The file or folder name. Used only when the format is DIVA\_FORMAT\_FOLDER\_BASED.

## range

The range of frames to be restored. Used only when the format is DIVA\_FORMAT\_DPX.

## DIVA\_OFFSET\_PAIR (This class only has public functions.)

The following are the constructors:

### DIVA\_SPEC DIVA\_OFFSET\_PAIR ( \_\_int64 pBegin, \_\_int64 pEnd, bool \_isTimeCode)

Constructor for use with byte offsets. DIVA\_OFFSET\_BYTE\_BEGIN and DIVA\_OFFSET\_BYTE\_END are valid.

**DIVA\_SPEC DIVA\_OFFSET\_PAIR (const DIVA\_STRING &pBegin, const DIVA\_STRING &pEnd)**

Constructor for use with timecode offsets. Timecodes are formatted as HH:MM:SS:FF.

The following are the attribute accessors:

**DIVA\_SPEC bool isTimeCode();**

This is true if the offset pair was constructed with timecode offsets.

**DIVA\_SPEC DIVA\_STRING getTimeCodeBegin();**

Return the beginning offset as a timecode.

**DIVA\_SPEC DIVA\_STRING getTimeCodeEnd();**

Return the ending offset as a timecode.

**DIVA\_SPEC \_\_int64 getByteBegin();**

Return the beginning offset as bytes.

**DIVA\_SPEC \_\_int64 getByteEnd();**

Return the ending offset as bytes.

```
class DIVA_FILE_FOLDER {
public:
    DIVA_STRING    fileFolder;
    DIVA_STRING    option
};
```

**fileFolder**

The file or folder name.

**option**

Options (for example, -r to recurse folders).

```
class DIVA_RANGE {
public:
    int    startRange;
    int    endRange;
};
```

**startRange**

The first frame number to be restored.

## endRange

The last frame number to be restored.

The format gives information about how to interpret the interval and about which specific operation should eventually be performed.

```
typedef enum {  
    DIVA_FORMAT_BYTES = 0,  
    DIVA_FORMAT_BYTES_HEADER,  
    DIVA_FORMAT_VIDEO_GXF,  
    DIVA_FORMAT_VIDEO_SEA,  
    DIVA_FORMAT_VIDEO_AVI_MATROX,  
    DIVA_FORMAT_VIDEO_MPEG2_TS,  
    DIVA_FORMAT_VIDEO_MXF,  
    DIVA_FORMAT_VIDEO_PINNACLE,  
    DIVA_FORMAT_VIDEO_OMNEON,  
    DIVA_FORMAT_VIDEO_LEITCH,  
    DIVA_FORMAT_VIDEO_QUANTEL,  
    DIVA_FORMAT_AUTODETECT,  
    DIVA_FORMAT_FOLDER_BASED,  
    DIVA_FORMAT_DPX  
} DIVA_FORMAT;
```

## DIVA\_FORMAT\_BYTES

Raw bytes

## DIVA\_FORMAT\_VIDEO\_GXF

GXF video format

## DIVA\_FORMAT\_VIDEO\_SEA

Seachange video format

## DIVA\_FORMAT\_VIDEO\_AVI\_MATROX

Matrox-specific AVI format (+ WAV files)

## DIVA\_FORMAT\_VIDEO\_MPEG\_TS

MPEG Transport Stream

## DIVA\_FORMAT\_VIDEO\_MXF

MXF video format

## DIVA\_FORMAT\_VIDEO\_PINNACLE

Pinnacle video format

**DIVA\_FORMAT\_VIDEO\_OMNEON**

Omneon video format

**DIVA\_FORMAT\_VIDEO\_LEITCH**

Leitch video format

**DIVA\_FORMAT\_VIDEO\_QUANTEL**

Quantel QCP video format

**DIVA\_FORMAT\_VIDEO\_AUTODETECT**

Automatic format detection

**DIVA\_FORMAT\_FOLDER\_BASED**

Fully restore the specified files and (or) folders

**DIVA\_FORMAT\_DPX**

DPX video format

**Return Values**

One of the following DIVA\_STATUS constants defined in DIVAapi.h:

**DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

**DIVA\_ERR\_NOT\_CONNECTED**

No connection is open.

**DIVA\_ERR\_SYSTEM\_IDLE**

DIVA Core can no longer accept connections and queries.

**DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

**DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. You set the timeout duration using the DIVA\_API\_TIMEOUT variable. The default value is one hundred-eighty (180) seconds.

**DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

**DIVA\_ERR\_INTERNAL**

The Core Manager or API detected an internal error.

**DIVA\_ERR\_INVALID\_PARAMETER**

The Core Manager did not understand a parameter value.

**DIVA\_ERR\_CANNOT\_ACCEPT\_MORE\_REQUESTS**

The count of simultaneous requests reached the maximum allowed value. You set this variable in the manager.conf configuration file. The default value is three hundred.

**DIVA\_ERR\_OBJECT\_DOESNT\_EXIST**

The specified Virtual Object does not exist in the DIVA Core database.

**DIVA\_ERR\_OBJECT\_OFFLINE**

There is no inserted instance in the Managed Storage and no Actor could provide a disk instance.

**DIVA\_ERR\_SEVERAL\_OBJECTS**

More than one Virtual Object with the specified name exists in the DIVA Core database.

**DIVA\_ERR\_INSTANCE\_OFFLINE**

The instance specified for restoring this Virtual Object is ejected, or the Actor owning the specified disk instance is not available.

**DIVA\_ERR\_INSTANCE\_DOESNT\_EXIST**

The instance specified for restoring this Virtual Object does not exist.

**DIVA\_ERR\_OBJECT\_IN\_USE**

The Virtual Object is currently in use (being Archived, Restored, Deleted, and so on).

**DIVA\_ERR\_SOURCE\_OR\_DESTINATION\_DOESNT\_EXIST**

The specified Server is unknown by the DIVA Core system.

**DIVA\_ERR\_OBJECT\_PARTIALLY\_DELETED**

The specified Virtual Object has instances that are partially deleted.

See also [DIVA\\_restoreObject](#), [DIVA\\_getRequestInfo](#), and [DIVA\\_getPartialRestoreRequestInfo](#).

## DIVA\_release

Indicates to the Core Manager that this instance can be externalized. This function has no effect if the instance has already been released. The list of instances that are RELEASED and INSERTED may be retrieved and shown at the System Management App.

### Synopsis

```
#include "DIVAapi.h"

DIVA_STATUS DIVA_release (
    IN DIVA_STRING      objectName,
    IN DIVA_STRING      CollectionName,
    IN int               instanceID
);
```

### objectName

The name of the Virtual Object to be copied.

### objectCollection

The Collection assigned to the Virtual Object when it was archived. This parameter can be a null string; however this may result in an error if several Virtual Objects have the same name.

### instanceID

A value of DIVA\_EVERY\_INSTANCE forces this function to apply to every instance of the given Virtual Object.

### Return Values

One of the following DIVA\_STATUS constants defined in DIVAapi.h:

#### DIVA\_OK

The request was correctly submitted and accepted by the Core Manager.

#### DIVA\_ERR\_NOT\_CONNECTED

No connection is open.

#### DIVA\_ERR\_SYSTEM\_IDLE

DIVA Core can no longer accept connections and queries.

**DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

**DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. You set the timeout duration using the `DIVA_API_TIMEOUT` variable. The default value is one hundred-eighty (180) seconds.

**DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

**DIVA\_ERR\_INTERNAL**

The Core Manager or API detected an internal error.

**DIVA\_ERR\_INVALID\_PARAMETER**

The Core Manager did not understand a parameter value.

**DIVA\_ERR\_OBJECT\_DOESNT\_EXIST**

The specified Virtual Object does not exist in the DIVA Core database.

**DIVA\_ERR\_INSTANCE\_DOESNT\_EXIST**

The instance specified for restoring this Virtual Object does not exist.

**DIVA\_ERR\_INSTANCE\_MUST\_BE\_ON\_TAPE**

No tape instance exists for this Virtual Object.

**DIVA\_ERR\_NO\_INSTANCE\_TAPE\_EXIST**

The specified Virtual Object has instances that are partially deleted.

**DIVA\_ERR\_SEVERAL\_OBJECTS**

More than one Virtual Object with the specified name exists in the DIVA Core database.

See also [DIVA\\_require](#).

**DIVA\_require**

Indicates to the Core Manager that this instance must be inserted. If the instance is already inserted, this function has no effect. The list of instances that are REQUIRED and EJECTED can be retrieved and shown at the System Management App.

## Synopsis

```
#include "DIVAapi.h"

DIVA_STATUS DIVA_require(
    IN DIVA_STRING      objectName,
    IN DIVA_STRING      CollectionName,
    IN int               instanceID
);
```

### objectName

Name of the Virtual Object to be copied.

### objectCollection

Collection assigned to the Virtual Object when it was archived. This parameter can be a null string, however this may result in an error if several Virtual Objects have the same name.

### instanceID

A value of DIVA\_EVERY\_INSTANCE forces the function to apply to every instance of the given Virtual Object.

## Return Values

One of the following DIVA\_STATUS constants defined in DIVAapi.h:

### DIVA\_OK

The request was correctly submitted and accepted by the Core Manager.

### DIVA\_ERR\_NOT\_CONNECTED

No connection is open.

### DIVA\_ERR\_SYSTEM\_IDLE

DIVA Core can no longer accept connections and queries.

### DIVA\_ERR\_BROKEN\_CONNECTION

The connection with the Core Manager was broken.

### DIVA\_ERR\_TIMEOUT

The timeout limit was reached before communication with the Core Manager could be performed. You set the timeout duration using the DIVA\_API\_TIMEOUT variable. The default value is one hundred-eighty (180) seconds.

**DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

**DIVA\_ERR\_INTERNAL**

The Core Manager or API detected an internal error.

**DIVA\_ERR\_INVALID\_PARAMETER**

The Core Manager did not understand a parameter value.

**DIVA\_ERR\_OBJECT\_DOESNT\_EXIST**

The specified Virtual Object does not exist in the DIVA Core database.

**DIVA\_ERR\_INSTANCE\_DOESNT\_EXIST**

The instance specified for restoring this Virtual Object does not exist.

**DIVA\_ERR\_INSTANCE\_MUST\_BE\_ON\_TAPE**

No tape instance exists for this Virtual Object.

**DIVA\_ERR\_NO\_INSTANCE\_TAPE\_EXIST**

The specified Virtual Object has instances that are partially deleted.

**DIVA\_ERR\_SEVERAL\_OBJECTS**

More than one Virtual Object with the specified name exists in the DIVA Core database.

See also [DIVA\\_release](#).

**DIVA\_restoreInstance**

Restores an Virtual Object from a specific instance. If the instance is externalized the operation fails even if there are other instances available for the Virtual Object.

**Synopsis**

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_restoreInstance (
IN DIVA_STRING          objectName,
IN DIVA_STRING          CollectionName,
IN int                  instanceID,
IN DIVA_STRING          destination,
IN DIVA_STRING          filePathRoot,
IN DIVA_RESTORE_QOS     qualityOfService,
IN int                  priorityLevel,
IN DIVA_STRING          restoreOptions,
```

```
OUT int *requestNumber  
);
```

## **objectName**

Name of the Virtual Object to be restored.

## **objectCollection**

Collection assigned to the Virtual Object when it was archived. This parameter can be a null string, however this may result in an error if several Virtual Objects have the same name.

## **instanceID**

The instance identifier.

## **destination**

The Destination Server (for example, a video server or browsing server) where the Virtual Object files will be restored. This name must be known by the DIVA Core configuration description.

## **filePathRoot**

Root folder on the Destination Server where the Virtual Object files will be placed. If this is null (string("")), the files will be placed in the FILES\_PATH\_ROOT folder specified when archiving the Virtual Object using the DIVA\_archiveObject() function.

## **qualityOfService**

One of the following codes:

### **DIVA\_QOS\_DEFAULT**

Restoring is performed according to the default Quality Of Service (currently direct and cache for restore operations).

### **DIVA\_QOS\_CACHE\_ONLY**

Use cache archive only.

### **DIVA\_QOS\_DIRECT\_ONLY**

Use direct restore only - no disk instance is created.

### **DIVA\_QOS\_CACHE\_AND\_DIRECT**

Use cache restore if available, or direct restore if cache restore is not available.

**DIVA\_QOS\_DIRECT\_AND\_CACHE**

Use direct restore if available, or cache restore if direct restore is not available.

Additional and optional services are available. To request those services, use a logical OR between the previously documented Quality Of Service parameter and the following constant:

**DIVA\_RESTORE\_SERVICE\_DO\_NOT\_OVERWRITE**

Do not overwrite existing files on the Destination Server.

**priorityLevel**

The priority level for this request. The priorityLevel can be in the range zero to one hundred, or the value DIVA\_DEFAULT\_REQUEST\_PRIORITY. The value zero is the lowest priority and one hundred the highest priority.

There are six predefined values as follows:

- DIVA\_REQUEST\_PRIORITY\_MIN
- DIVA\_REQUEST\_PRIORITY\_LOW
- DIVA\_REQUEST\_PRIORITY\_NORMAL
- DIVA\_REQUEST\_PRIORITY\_HIGH
- DIVA\_REQUEST\_PRIORITY\_MAX
- DIVA\_DEFAULT\_REQUEST\_PRIORITY

When the DIVA\_DEFAULT\_REQUEST\_PRIORITY value is used, the Core Manager uses the default priority defined in the Core Manager configuration for the request.

Using a value either outside of the range of zero to one hundred, or predefined values yields a DIVA\_ERR\_INVALID\_PARAMETER error.

**restoreOptions**

Additional options that must be used for performing the transfer of data from DIVA Core to the Destination Server. These options supersede any options specified in the DIVA Core configuration database. Currently the possible values for restoreOptions are as follows:

**Null String**

A null string specifies no options.

**-login**

A user name and password is required to log in to some Source Servers. This option obsoletes the -gateway option from earlier releases.

**-pass**

The password used with -login.

**requestNumber**

A number identifying this request.

**Return Values**

One of the following DIVA\_STATUS constants defined in DIVAapi.h:

**DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

**DIVA\_ERR\_NOT\_CONNECTED**

No connection is open.

**DIVA\_ERR\_SYSTEM\_IDLE**

DIVA Core can no longer accept connections and queries.

**DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

**DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. You set the timeout duration using the DIVA\_API\_TIMEOUT variable. The default value is one hundred-eighty (180) seconds.

**DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

**DIVA\_ERR\_INTERNAL**

The Core Manager or API detected an internal error.

**DIVA\_ERR\_INVALID\_PARAMETER**

The Core Manager did not understand a parameter value.

**DIVA\_ERR\_CANNOT\_ACCEPT\_MORE\_REQUESTS**

Count of simultaneous requests has reached the maximum allowed value. This variable is set in the manager.conf configuration file. The default is 300.

**DIVA\_ERR\_OBJECT\_DOESNT\_EXIST**

The specified Virtual Object does not exist in the DIVA Core database.

## DIVA\_ERR\_SEVERAL\_OBJECTS

More than one Virtual Object with the specified name exists in the DIVA Core database.

## DIVA\_ERR\_INSTANCE\_OFFLINE

The specified instance for restoring this Virtual Object is ejected, or the Actor owning the specified disk instance is not available.

## DIVA\_ERR\_INSTANCE\_DOESNT\_EXIST

The instance specified for restoring this Virtual Object does not exist.

## DIVA\_ERR\_OBJECT\_IN\_USE

The Virtual Object is currently in use (being Archived, Restored, Deleted, and so on).

## DIVA\_ERR\_SOURCE\_OR\_DESTINATION\_DOESNT\_EXIST

The specified Server is not known by the DIVA Core system.

## DIVA\_ERR\_OBJECT\_PARTIALLY\_DELETED

The specified Virtual Object has instances that are partially deleted.

See also [DIVA\\_archiveObject](#) and [DIVA\\_getObjectInfo](#).

# DIVA\_restoreObject

Submits an Virtual Object Restore request to the Core Manager and the Core Manager chooses the appropriate instance to be restored. This function returns as soon as the Core Manager accepts the request. To check that the operation was successful, the application must call the function `DIVA_getRequestInfo()`.

If the requested Virtual Object is on media that is not available, the request will fail. The media names (tape barcodes and disk names) that contain instances of the Virtual Object will be included in the `additionalInfo` field of the `DIVA_getRequestInfo()` response.

## Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_restoreObject (
IN DIVA_STRING          objectName,
IN DIVA_STRING          objectCollection,
IN DIVA_STRING          destination,
IN DIVA_STRING          filePathRoot,
IN DIVA_RESTORE_QOS     qualityOfService,
IN int                  priorityLevel,
IN DIVA_STRING          restoreOptions,
OUT int                 *requestNumber
);
```

## **objectName**

Name of the Virtual Object to be restored.

## **objectCollection**

Collection assigned to the Virtual Object when it was archived. This parameter can be a null string, but this may result in an error if several Virtual Objects have the same name.

## **destination**

The Destination Server (for example, a video server or browsing server) where the Virtual Object files will be restored. This name must be known by the DIVA Core configuration description.

## **filePathRoot**

Root folder on the Destination Server where the Virtual Object files will be placed. If this is null (string("")), the files will be placed in the FILES\_PATH\_ROOT folder specified when archiving the Virtual Object using the DIVA\_archiveObject() function.

## **qualityOfService**

One of the following codes:

### **DIVA\_QOS\_DEFAULT**

Restoring is performed according to the default Quality Of Service (currently direct and cache for restore operations).

### **DIVA\_QOS\_CACHE\_ONLY (-qos\_cache\_only)**

Use cache restore only.

### **DIVA\_QOS\_DIRECT\_ONLY (-qos\_direct\_only)**

Use direct restore only.

### **DIVA\_QOS\_CACHE\_AND\_DIRECT (-qos\_cache\_and\_direct)**

Use cache restore if available, or direct restore if cache restore is not available.

### **DIVA\_QOS\_DIRECT\_AND\_CACHE (-qos\_direct\_and\_cache)**

Use direct restore if available, or cache restore if direct restore is not available.

Additional and optional services are available. To request those services, use a logical OR between the previously documented Quality Of Service parameter and the following constant:

### **DIVA\_QOS\_NEARLINE\_ONLY (-qos\_nearline\_only)**

Use nearline restore only. Nearline restore will restore from a disk instance if it exists, otherwise, it will create a disk instance and restore from the newly created disk instance.

**DIVA\_QOS\_NEARLINE\_AND\_DIRECT (-qos\_nearline\_and\_direct)**

Use Nearline restore if available, or direct restore if Nearline restore is not available. Additional and optional services are available. To request those services use a logical OR between the previously documented Quality Of Service parameter and the following constants:

**DIVA\_RESTORE\_SERVICE\_DO\_NOT\_OVERWRITE**

Do not overwrite existing files on the Destination Server.

**DIVA\_RESTORE\_SERVICE\_DO\_NOT\_CHECK\_EXISTENCE**

Do not check existence of the clip on the server.

**DIVA\_RESTORE\_SERVICE\_DELETE\_AND\_WRITE**

Force delete and rewrite if Virtual Object exists on the server.

**DIVA\_RESTORE\_SERVICE\_DEFAULT**

Operate using the default setting in the Core Manager configuration.

**priorityLevel**

The priority level for this request. The priorityLevel can be in the range zero to one hundred, or the value DIVA\_DEFAULT\_REQUEST\_PRIORITY. The value zero is the lowest priority and one hundred the highest priority.

There are six predefined values as follows:

- DIVA\_REQUEST\_PRIORITY\_MIN
- DIVA\_REQUEST\_PRIORITY\_LOW
- DIVA\_REQUEST\_PRIORITY\_NORMAL
- DIVA\_REQUEST\_PRIORITY\_HIGH
- DIVA\_REQUEST\_PRIORITY\_MAX
- DIVA\_DEFAULT\_REQUEST\_PRIORITY

When the DIVA\_DEFAULT\_REQUEST\_PRIORITY value is used, the Core Manager uses the default priority defined in the Core Manager configuration for the request.

Using a value either outside of the range of zero to one hundred, or predefined values yields a DIVA\_ERR\_INVALID\_PARAMETER error.

**restoreOptions**

Additional options that must be used for performing the transfer of data from DIVA Core to the Destination Server. These options supersede any options specified in the DIVA Core configuration database. Currently the possible values for restoreOptions are as follows:

**Null String**

A null string specifies no options.

**-login**

A user name and password is required to log in to some Source Servers. This option obsoletes the -gateway option from earlier releases.

**-pass**

The password used with -login.

**requestNumber**

Request number assigned to this request. This number is used for querying the status or canceling this request.

**Return Values**

One of the following DIVA\_STATUS constants defined in DIVAapi.h:

**DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

**DIVA\_ERR\_NOT\_CONNECTED**

No connection is open.

**DIVA\_ERR\_SYSTEM\_IDLE**

DIVA Core can no longer accept connections and queries.

**DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

**DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. You set the timeout duration using the DIVA\_API\_TIMEOUT variable. The default value is one hundred-eighty (180) seconds.

**DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

**DIVA\_ERR\_INTERNAL**

The Core Manager or API detected an internal error.

**DIVA\_ERR\_INVALID\_PARAMETER**

The Core Manager did not understand a parameter value.

## DIVA\_ERR\_CANNOT\_ACCEPT\_MORE\_REQUESTS

The count of simultaneous requests reached the maximum allowed value. You set this variable in the `manager.conf` configuration file. The default value is three hundred.

## DIVA\_ERR\_OBJECT\_DOESNT\_EXIST

The specified Virtual Object does not exist in the DIVA Core database.

## DIVA\_ERR\_OBJECT\_OFFLINE

There is no inserted instance in the Managed Storage and no Actor could provide a Disk Instance.

## DIVA\_ERR\_SEVERAL\_OBJECTS

More than one Virtual Object with the specified name exists in the DIVA Core database.

## DIVA\_ERR\_OBJECT\_IN\_USE

The Virtual Object is currently in use (being Archived, Restored, Deleted, and so on).

## DIVA\_ERR\_SOURCE\_OR\_DESTINATION\_DOESNT\_EXIST

The specified Server is not known by the DIVA Core system.

## DIVA\_ERR\_OBJECT\_PARTIALLY\_DELETED

The specified Virtual Object has instances that are partially deleted.

See also [DIVA\\_getRequestInfo](#) and [DIVA\\_copyToGroup](#) and [DIVA\\_copy](#).

# DIVA\_transcodeArchive

Submits a Transcode Archive request to the Core Manager. The original Virtual Object will be restored to the local Actor cache then transcoded to the format defined in the option field. A new Virtual Object containing the transcoded clip will then be archived back to DIVA Core.

## Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_transcodeArchive (  
    IN DIVA_STRING          parentObjectName,  
    IN DIVA_STRING          parentObjectCollection,  
    IN int                  instance,  
    IN DIVA_STRING          objectName,  
    IN DIVA_STRING          objectCollection,  
    IN DIVA_STRING          mediaName,  
    IN DIVA_STRING          comments,
```

```

IN DIVA_STRING          archiveOptions,
IN DIVA_ARCHIVE_QOS     qualityOfService,
IN bool                 bCascadeDelete,
IN int                  priorityLevel,
OUT int                 *requestNumber
);

```

## parentObjectName

Name of the original Virtual Object to be transcoded.

## parentObjectCollection

Collection assigned to the original Virtual Object.

## instance

Instance of the parent Virtual Object. The default is -1.

## objectName

Name of the resulting transcoded Virtual Object from the transcoding operation.

## objectCollection

Collection of the transcoded Virtual Object.

## mediaName

The tape group or disk array where the Virtual Object is to be saved. The media may be defined as follows:

### Name (of the Tape Group or Array)

Provide the tape group or disk array name as defined in the configuration. The Virtual Object is saved to the specified media and assigned to the default SP (Storage Plan).

### SP Name

Provide a SP Name (Storage Plan Name) as defined in the configuration. The Virtual Object will be assigned to the specified Storage Plan and saved to the default media specified.

### Both of the above (Name and SP Name)

The Virtual Object is saved to the specified media as in Name, and assigned to the specified Storage Plan as in SP Name. The Name and the SP Name must be separated by the & delimiter (this is configurable).

When this parameter is a null string, the default group of tapes called DEFAULT is used. Complex Virtual Objects can only be saved to AXF media types.

## comments

Optional information describing the Virtual Object. This can be a null string.

## archiveOptions

Additional options that must be used for performing the transfer of data from the Source Server to DIVA Core. These options supersede any options specified in the DIVA Core configuration database. Currently the possible values for archiveOptions are:

### **-tr\_archive\_format FORMAT**

Destination Server format of the retrieved Virtual Object. This is required.

### **-tr\_names trans1**

Names of the transcoders that have to perform this operation. If more than one transcoder is selected, the performing transcoder will be chosen based on the current loading. If this option is not specified, the performing transcoder will be chosen from all DIVA Core transcoders based on the current loading. This is optional.

### **-tr\_names trans1,trans2**

Names of the transcoders that have to perform this operation. Multiple transcoders are identified in a comma separated list (trans1, trans2, and so on). If more than one transcoder is selected, the performing transcoder will be chosen based on the current loading. If this option is not specified, the performing transcoder will be chosen from all DIVA Core transcoders based on the current loading. This is optional.

## qualityOfService

One of the following codes:

### **DIVA\_QOS\_DEFAULT**

Restoring is performed according to the default Quality Of Service (currently cache for archive operations).

### **DIVA\_QOS\_CACHE\_ONLY**

Use cache archive only.

### **DIVA\_QOS\_DIRECT\_ONLY**

Use direct archive only - no disk instance is created.

### **DIVA\_QOS\_CACHE\_AND\_DIRECT**

Use cache archive if available, or direct archive if cache archive is not available.

### **DIVA\_QOS\_DIRECT\_AND\_CACHE**

Use direct archive if available, or cache archive if direct archive is not available.

## **bCascadeDelete**

Shows if transcoded Virtual Object is linked to the original Virtual Object. If true both the original Virtual Object and the transcoded Virtual Object will be deleted.

## **priorityLevel**

The priority level for this request. The priorityLevel can be in the range zero to one hundred, or the value `DIVA_DEFAULT_REQUEST_PRIORITY`. The value zero is the lowest priority and one hundred the highest priority.

There are six predefined values as follows:

- `DIVA_REQUEST_PRIORITY_MIN`
- `DIVA_REQUEST_PRIORITY_LOW`
- `DIVA_REQUEST_PRIORITY_NORMAL`
- `DIVA_REQUEST_PRIORITY_HIGH`
- `DIVA_REQUEST_PRIORITY_MAX`
- `DIVA_DEFAULT_REQUEST_PRIORITY`

When the `DIVA_DEFAULT_REQUEST_PRIORITY` value is used, the Core Manager uses the default priority defined in the Core Manager configuration for the request.

Using a value either outside of the range of zero to one hundred, or predefined values yields a `DIVA_ERR_INVALID_PARAMETER` error.

## **requestNumber**

Request number assigned to this request. This number is used for querying the status or canceling this request.

## **Return Values**

One of the following `DIVA_STATUS` constants defined in `DIVAapi.h`:

### **DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

### **DIVA\_ERR\_NOT\_CONNECTED**

No connection is open.

### **DIVA\_ERR\_SYSTEM\_IDLE**

The DIVA Core system can no longer accept connections and queries.

### **DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

## DIVA\_ERR\_TIMEOUT

The timeout limit was reached before communication with the Core Manager could be performed. You set the timeout duration using the `DIVA_API_TIMEOUT` variable. The default value is one hundred-eighty (180) seconds.

## DIVA\_ERR\_UNKNOWN

An unknown status was received from the Core Manager.

## DIVA\_ERR\_INTERNAL

An internal error was detected by the Core Manager or by the API.

## DIVA\_ERR\_INVALID\_PARAMETER

A parameter value was not understood by the Core Manager.

## DIVA\_ERR\_CANNOT\_ACCEPT\_MORE\_REQUESTS

The count of simultaneous requests reached the maximum allowed value. You set this variable in the `manager.conf` configuration file. The default value is three hundred.

## DIVA\_ERR\_OBJECT\_ALREADY\_EXISTS

The specified Virtual Object already exists in the DIVA Core database.

## DIVA\_ERR\_OBJECT\_PARTIALLY\_DELETED

The specified Virtual Object has instances that are partially deleted.

See also [DIVA\\_linkObjects](#).

## DIVA\_transferFiles

Submits a Transfer Files request to the Core Manager. The request will transfer files from a remote server (the Source Server) to another remote server (the Destination Server). This function returns as soon as the Core Manager accepts the request. The application must call the function `DIVA_getRequestInfo()` to confirm that the operation completed successfully.

### Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_transferFiles (
    IN DIVA_STRING          source,
    IN DIVA_STRING          sourcePathRoot,
    IN vector<DIVA_STRING>  filenamesList,
    IN DIVA_STRING          destination,
    IN DIVA_STRING          destinationPathRoot,
```

```
IN int                                priorityLevel,  
OUT int                              *requestNumber  
);
```

### source

Name of the Source Server (for example, a video server or browsing server). This name must be known by the DIVA Core configuration description.

### sourcePathRoot

Root folder for the files specified by the filenamesList parameter.

### filenamesList

List of file path names relative to the folder specified by the sourcePathRoot parameter. When the sourcePathRoot is null, path names must be absolute names.

### destination

Name of the Destination Server (for example a video server or browsing server). This name must be known by the DIVA Core configuration description.

### destinationPathRoot

Root folder where the files will be placed at the Destination Server.

### priorityLevel

The priority level for this request. The priorityLevel can be in the range zero to one hundred, or the value DIVA\_DEFAULT\_REQUEST\_PRIORITY. The value zero is the lowest priority and one hundred the highest priority.

There are six predefined values as follows:

- DIVA\_REQUEST\_PRIORITY\_MIN
- DIVA\_REQUEST\_PRIORITY\_LOW
- DIVA\_REQUEST\_PRIORITY\_NORMAL
- DIVA\_REQUEST\_PRIORITY\_HIGH
- DIVA\_REQUEST\_PRIORITY\_MAX
- DIVA\_DEFAULT\_REQUEST\_PRIORITY

When the DIVA\_DEFAULT\_REQUEST\_PRIORITY value is used, the Core Manager uses the default priority defined in the Core Manager configuration for the request.

Using a value either outside of the range of zero to one hundred, or predefined values yields a DIVA\_ERR\_INVALID\_PARAMETER error.

## **requestNumber**

Request number assigned to this request. This number is used for querying the status or canceling this request.

## **Return Values**

One of the following DIVA\_STATUS constants defined in DIVAapi.h:

### **DIVA\_OK**

The request was correctly submitted and accepted by the Core Manager.

### **DIVA\_ERR\_NOT\_CONNECTED**

No connection is open.

### **DIVA\_ERR\_SYSTEM\_IDLE**

The DIVA Core system is no longer able to accept connections and queries.

### **DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

### **DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the DIVA\_API\_TIMEOUT variable and equals one hundred-eighty (180) seconds by default.

### **DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

### **DIVA\_ERR\_INTERNAL**

An internal error was detected by the Core Manager or by the API.

### **DIVA\_ERR\_INVALID\_PARAMETER**

A parameter value was not understood by the Core Manager.

### **DIVA\_ERR\_CANNOT\_ACCEPT\_MORE\_REQUESTS**

The count of simultaneous requests reached the maximum allowed value. This variable is set in the manager.conf configuration file and the default value is three hundred.

## DIVA\_ERR\_SOURCE\_OR\_DESTINATION\_DOESNT\_EXIST

The specified Server is not known by the DIVA Core system.

Also see [DIVA\\_getRequestInfo](#).

## DIVA\_unlockObject

A call to this function will unlock an Virtual Object. Locked Virtual Objects cannot be restored.

### Synopsis

```
#include "DIVAapi.h"
```

```
DIVA_STATUS DIVA_unlockObject (  
IN DIVA_STRING      objectName,  
IN DIVA_STRING      Collection,  
IN string            options  
);
```

### objectName

Name of the Virtual Object.

### Collection

The Collection assigned to the Virtual Object when it was archived.

### options

TBD

### Return Values

One of the following DIVA\_STATUS constants defined in DIVAapi.h:

#### DIVA\_OK

The request was correctly submitted and accepted by the Core Manager.

#### DIVA\_ERR\_NOT\_CONNECTED

No connection is open.

#### DIVA\_ERR\_SYSTEM\_IDLE

The DIVA Core system is no longer able to accept connections and queries.

## **DIVA\_ERR\_BROKEN\_CONNECTION**

The connection with the Core Manager was broken.

## **DIVA\_ERR\_TIMEOUT**

The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the `DIVA_API_TIMEOUT` variable and equals one hundred-eighty (180) seconds by default.

## **DIVA\_ERR\_UNKNOWN**

An unknown status was received from the Core Manager.

## **DIVA\_ERR\_INTERNAL**

An internal error was detected by the Core Manager or by the API.

# Using the API with DIVA Connect

In addition to being able to connect to a DIVA Core system, you can use the API to connect to an DIVA Connect system. This functionality enables applications to access content across multiple DIVA Core systems, possibly in different geographical locations. DIVA Connect enables the content in each system to be retrieved and stored as if the sites together were one large archival system.

## Topics:

- [What is DIVA Connect?](#)
- [DIVA Core API Support](#)
- [Input Parameters](#)
- [Return Parameters](#)
- [Return Codes](#)
- [getObjectDetailsList Call](#)

## What is DIVA Connect?

DIVA Connect provides a unified view of archived content across multiple, distributed DIVA Core systems. It facilitates the moving of content among DIVA Core sites, and from customer Source and Destination Servers and disk. The purpose is for disaster recovery, content distribution, access control, performance, and content availability.

DIVA Connect synchronizes asset information from each DIVA Core site, so that users always have an up-to-date inventory of where content is. DIVA Connect uses this information to choose the best site for various requests, for example restores and copies. DIVA Connect also provides access rules to limit the operations that users are permitted to perform.

DIVA Connect 2.2 is compatible with DIVA Core 8.2 Linux-based installations. DIVA Connect 2.2 also runs on Windows-based systems. However, it is not backward compatible to releases before DIVA Core 7.3.1. You must use either DIVA Connect 2.0 or Legacy DIVA Connect when running DIVA Core releases earlier than DIVA Core 7.3.1.

The Legacy DIVA Connect is still available for connecting DIVA Core systems with different software release levels, and releases before DIVA Core 7.3.1.

If you are operating a DIVA Core release earlier than 7.3.1, refer to the DIVA Connect Installation, Configuration, and Operations Guide (named DIVA Connect for DIVA Core releases 6.5 and 7.2).

## DIVA Core API Support

DIVA Connect has partial support for the full API command set. Refer to the appropriate DIVA Connect documentation for a complete list of supported API commands. DIVA Connect will support client connections from API clients release 8.2 and earlier. New parameters or features added to the API after release 7.5 are not supported by Legacy DIVA Connect. In general, a released DIVA Connect can connect to newer releases of DIVA Core, and sometimes also can connect to older releases. This ability varies based on the specific release of DIVA Connect.

## Input Parameters

Invoking API calls to a DIVA Connect server is largely the same as invoking calls to DIVA Core. However, there are some differences. DIVA Connect sometimes accepts additional information by using common DIVA Core API parameters in a slightly a different way.

For example, you can use the DIVA Connect Copy command (CopyToGroup) to copy content from one DIVA Core system to another. DIVA Connect needs to know, at a minimum, what the target DIVA Core site is. This information can be provided in multiple ways, for example you can prefix the target\_sitename to the media provided in the call (for example, sitename2\_mytapeigroup). Refer to the appropriate DIVA Connect documentation for more information on specifying DIVA Connect-specific information in API calls.

## Return Parameters

A DIVA Connect system sometimes returns API information that is slightly different than you would typically see in a DIVA Core system. For example, the DIVA Connect `getObjectInfo()` call returns information about an archived Virtual Object across all DIVA Core sites. To distinguish which site is which, the Source Server site name is prefixed to the media of each archived Virtual Object instance returned in the call. For example, a Virtual Object on `sitename2` that is stored on `mytapegroup` would have a media value of `sitename2_mytapegroup`.

Another example of a slight difference is the Virtual Object instance ID. DIVA Core has a unique instance ID for each instance of an archived Virtual Object (starting at zero and incrementing by one for each new instance). However, this value is not unique across DIVA Core sites. DIVA Connect applies a simple algorithm to the instance ID to make it unique across sites (but not across Virtual Objects). The unique DIVA Connect instance IDs for a Virtual Object can be queried by making a DIVA Connect `getObjectInfo()` call.

The Request ID returned by each DIVA Connect request does not necessarily correspond to a DIVA Core Request ID. Refer to the appropriate DIVA Connect documentation for more information.

## Return Codes

DIVA Connect will return `DIVA_ERR_ACCESS_DENIED` if a user or connection does not have permission to perform a particular action. DIVA Core does not return this code. DIVA Connect can possibly refuse an API connection altogether because of configured permissions. DIVA Core will accept the connection if it hasn't run out of available connections. There are cases where DIVA Connect will choose to acknowledge a request with `DIVA_OK` and then subsequently return an error (for example, an Invalid Media error). DIVA Core will simply reject the request with the `DIVA_ERR_INVALID_PARAMETER` error.

## getObjectDetailsList Call

The `GetObjectDetailsList()` command retrieves a list of Virtual Objects from each site. DIVA Connect retrieves the Virtual Object information directly from each DIVA Core system, one site at a time, in a round-robin fashion. It returns one batch per site to the initiator. The initiator must keep calling `GetObjectDetailsList()` with the same query parameters - passing all received list position data as input to the next call.

If a Virtual Object is returned in one batch, the initiator can possibly receive the same Virtual Object again in the next batch (for the second site). This makes `GetObjectDetailsList()` different from `GetObjectInfo()`, which returns information from all sites in one call.

The query parameters and time ranges queried in each batch are specific to each site. It is possible that if Site1 contains many Virtual Objects in a given query (and Site2 does not). The batches from Site2 that are near the end of the calling sequence might be completely empty.

Keep calling `GetObjectDetailsList()`, ignoring empty batches until the call returns either a status of `DIVA_WARN_NO_MORE_OBJECTS` or an error. All DIVA Core sites in the DIVA Connect network must be online for `GetObjectDetailsList()` to succeed. If, for any reason, an error is returned before the list has been fully returned the entire calling sequence must be repeated.

Other details of the `GetObjectDetailsList()` call remain in effect for the DIVA Connect release. For example, while the batches returned are ordered by time, the order of entries within each batch is not guaranteed. Although duplicate Virtual Objects will not appear within a batch, the same Virtual Object may appear in the next batch - the likelihood of this occurrence increases when you use the `MODIFIED_SINCE` parameter.

If a Virtual Object has been deleted and subsequently re-added, `GetObjectDetailsList()` will return one record for every time this has occurred for the entire period that DIVA Core retains the records.

To continuously monitor DIVA Connect for new Virtual Objects and instances, you can continue to call `GetObjectDetailsList()` even after it has returned a status of `DIVA_WARN_NO_MORE_OBJECTS`. To do this you must provide the exact same query information (passing all received list position data into the next call) to get any new updates since you last called it. If an error occurs, you must use the exact same list position that was received on the last successful call.

Refer to the appropriate DIVA Connect documentation for more information on specific API calls.

# Appendix

The following sections include additional information not previously described in this book.

## Topics:

- [List of Authorized Special Characters in DIVA Core](#)
- [Maximum Allowed Number of Characters](#)
- [API Static Constant Values](#)

## List of Authorized Special Characters in DIVA Core

The following table lists the special characters that can be used in DIVA Core and in which fields they are valid.

Character	Name	Collection	Source	Media	Path	File	Comments	Options
~	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
'	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
!	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
@	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
#	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
%	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
^	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
&	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
*	No	No	Yes	Yes	No	Yes	Yes	Yes
(	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
_	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
-	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
+	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
=	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
\	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
{	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
[	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
}	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
:	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
;	Yes	Yes	Yes	Yes	Yes <sup>1</sup>	Yes	Yes	Yes
"	Yes	Yes	Yes	Yes	No	Yes	Yes	No
'	Yes	Yes	No	No	Yes <sup>1</sup>	Yes	Yes	Yes

Character	Name	Collection	Source	Media	Path	File	Comments	Options
<	Yes	Yes	Yes	Yes	No	Yes	Yes	No
,	Yes	Yes	Yes	Yes	Yes <sup>1</sup>	Yes	Yes	Yes
>	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
.	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
?	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
/	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
Space	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes

1. In a Windows environment, the file and folder name restrictions depend on the file system restrictions. File and folder names cannot solely consist of one or more spaces, and cannot contain a double-quote.

# Maximum Allowed Number of Characters

The maximum allowable number of characters are as follows:

## **Name**

192 maximum characters

## **Collection**

96 maximum characters

## **Source**

96 maximum characters

## **Media**

96 maximum characters

## **Path and File Name**

1536 maximum characters per folder or per file

## **Comments**

4000 maximum characters

## **Options**

768 maximum characters

## API Static Constant Values

The following table identifies the values for each of the API static constants.

Static Constant Name	Description	Value
DIVA_OK	The request was correctly submitted and accepted by the Core Manager.	1000
DIVA_ERR_UNKNOWN	An unknown status was received from the Core Manager.	1001
DIVA_ERR_INTERNAL	An internal error was detected by the Core Manager or the API.	1002
DIVA_ERR_NO_ARCHIVE_SYSTEM	Problem when establishing a connection with the specified DIVA Core system.	1003
DIVA_ERR_BROKEN_CONNECTION	The connection with the Core Manager was broken.	1004
DIVA_ERR_DISCONNECTING	Problem when disconnecting. The connection is still considered to be open.	1005
DIVA_ERR_ALREADY_CONNECTED	A connection is already open.	1006
DIVA_ERR_WRONG_VERSION	Release level of the API and the Core Manager are not compatible.	1007
DIVA_ERR_INVALID_PARAMETER	A parameter value was not understood by the Core Manager.	1008

Static Constant Name	Description	Value
DIVA_ERR_OBJECT_DOESNT_EXIST	The specified Virtual Object does not exist in the DIVA Core database.	1009
DIVA_ERR_SEVERAL_OBJECTS	More than one Virtual Object with the specified name exists in the DIVA Core database.	1010
DIVA_ERR_NO_SUCH_REQUEST	The requestNumber identifies no request.	1011
DIVA_ERR_NOT_CANCELABLE	The request is at the point where it cannot be canceled.	1012
DIVA_ERR_SYSTEM_IDLE	The DIVA Core System is no longer able to accept connections and queries.	1013
DIVA_ERR_WRONG_LIST_SIZE	The list size is zero or larger than the maximum allowable value.	1014
DIVA_ERR_LIST_NOT_INITIALIZED	The specified list was not properly initialized. Initialization call was not executed.	1015
DIVA_ERR_OBJECT_ALREADY_EXISTS	A Virtual Object with this name and Collection already exists in the DIVA Core system.	1016
DIVA_ERR_GROUP_DOESNT_EXIST	The specified Tape Group does not exist.	1017

Static Constant Name	Description	Value
DIVA_ERR_SOURCE_OR_DESTINATION_DOESNT_EXIST	The specified Source or Destination Server does not exist.	1018
DIVA_WARN_NO_MORE_OBJECTS	The end of the list was reached during the call.	1019
DIVA_ERR_NOT_CONNECTED	No open connection.	1020
DIVA_ERR_GROUP_ALREADY_EXISTS	The specified Tape Group already exists.	1021
DIVA_ERR_GROUP_IN_USE	The Tape Group contains at least one Virtual Object instance.	1022
DIVA_ERR_OBJECT_OFFLINE	There is no inserted instance in the Managed Storage and no Actor could provide a disk instance.	1023
DIVA_ERR_TIMEOUT	The timeout limit was reached before communication with the Core Manager could be performed. The timeout duration is set by the DIVA_API_TIMEOUT variable and equals one hundred-eighty (180) seconds by default.	1024

Static Constant Name	Description	Value
DIVA_ERR_LAST_INSTANCE	DIVA_deleteObject () must be used to delete the last instance of a Virtual Object.	1025
DIVA_ERR_PATH_DESTINATION	The specified Destination Server path is invalid.	1026
DIVA_ERR_INSTANCE_DOESNT_EXIST	Instance specified for restoring this Virtual Object does not exist.	1027
DIVA_ERR_INSTANCE_OFFLINE	Instance specified for restoring this Virtual Object is ejected, or the Actor owning the specified disk instance is unavailable.	1028
DIVA_ERR_INSTANCE_MUST_BE_ON_TAPE	The specified instance is not a tape instance.	1029
DIVA_ERR_NO_INSTANCE_TAPE_EXIST	No tape instance exists for this Virtual Object.	1030
DIVA_ERR_OBJECT_IN_USE	The Virtual Object is currently in use (being Archived, Restored, Deleted, and so on).	1031
DIVA_ERR_CANNOT_ACCEPT_MORE_REQUESTS	The count of simultaneous requests reached the maximum allowed value. This variable is set in the manager.conf configuration file. The default is 300.	1032

Static Constant Name	Description	Value
DIVA_ERR_TAPE_DOESNT_EXIST	There is no tape associated with the given barcode.	1033
DIVA_ERR_INVALID_INSTANCE_TYPE	Cannot partially restore this instance.	1034
DIVA_ERR_OBJECT_PARTIALLY_DELETED	The specified Virtual Object has instances that are partially deleted.	1036
DIVA_ERR_COMPONENT_NOT_FOUND	The specified component (file) is not found.	1038
DIVA_ERR_OBJECT_IS_LOCKED	Attempted to restore a Virtual Object that has been locked. A locked Virtual Object cannot be Restored or Copied to New.	1039
DIVA_ALL_REQUESTS	Specify all requests. Used by DIVA_cancelRequest.	-2
DIVA_ALL_INSTANCE	Specify all instances. Used by DIVA_release.	-1
DIVA_ANY_INSTANCE	Allow Core Manager to choose the instance.	-1
DIVA_DEFAULT_REQUEST_PRIORITY	The default request priority. This is used if no specific priority is selected when the request is configured.	-1
DIVA_REQUEST_PRIORITY_MIN	The default minimum request priority.	Default = 0

Static Constant Name	Description	Value
DIVA_REQUEST_PRIORITY_LOW	The default low request priority.	Default = 25
DIVA_REQUEST_PRIORITY_NORMAL	The default normal request priority.	Default = 50
DIVA_REQUEST_PRIORITY_HIGH	The default high request priority.	Default = 75
DIVA_REQUEST_PRIORITY_MAX	The default maximum request priority.	Default = 100
DIVA_MEDIA_FORMAT_UNKNOWN	The specified tape format is unknown.	-1
DIVA_MEDIA_FORMAT_LEGACY	The specified media format for the Tape Group or array is Legacy.	0
DIVA_MEDIA_FORMAT_AXF	The specified media format for the Tape Group or array is AXF 0.9.	1
DIVA_MEDIA_FORMAT_AXF_10	The specified media format for the Tape Group or array is AXF 1.0.	2
DIVA_OFFSET_BYTE_BEGIN	__int64 - The beginning byte of the file.	0
DIVA_OFFSET_BYTE_END	__int64 - The ending byte of the file.	-1
DIVA_OFFSET_INVALID	__int64 - The specified timecode offset is invalid.	-2
DIVA_OFFSET_TC_BEGIN	string - The file's beginning timecode.	00:00:00:00
DIVA_OFFSET_TC_END	string - The file's ending timecode.	99:99:99:99

# Glossary

## **Archive Related Operations Initiator**

An entity submitting requests to DIVA Core (typically, an automation process).

## **Array**

In DIVA Core, an array designates a collection of disks identified by their name as they are declared in the DIVA Core configuration. A disk name is associated with a mounting point. Archive requests can be submitted with an array as the Destination Server. DIVA Core is responsible for choosing the disk location to write the data to when several disks belong to the same array.

## **AXF (Archive Exchange Format)**

The AXF (Archive Exchange Format), or AXF Media Format, is based on a file and storage media agnostic encapsulation approach which abstracts the underlying file system, operating system, and storage technology making the format truly open and non-proprietary.

## **Collection**

Part of the access key to a Virtual Object. Categories are an approach to linking the Virtual Object with the user activity field. It must not be confused with a [Tape Group](#), which is a storage concept.

## **Complex Virtual Object**

A Virtual Object is defined as complex when it contains 1000 (this is the default, but the value is configurable) or more components. Complex Virtual Object handling may differ from non-complex Virtual Objects as noted throughout this document.

## **Critical Section**

A piece of code that accesses a shared resource (data structure or device) that must not be concurrently accessed by more than one execution thread.

## **Destination**

A system that receives restored data in the DIVA Core system (for example, video servers, remote computers, FTP servers, and so on). Destination Servers can also be used as a [Source](#) certain operations.

**DPX (Digital Moving-Picture Exchange)**

The DPX (Digital Moving-Picture Exchange) format is a high quality video format that consists of one or more files for each frame of video. This format is likely to be used with complex Virtual Objects.

**Externalization**

A Virtual Object instance is ejected (externalized) when one of the tapes containing the instance's elements is ejected. A Virtual Object is ejected when all of its instances are ejected. A Virtual Object is considered inserted when at least one instance of the Virtual Object is inserted.

**Initiator**

See [Archive Related Operations Initiator](#) previously described.

**Legacy Format**

DIVA Core proprietary storage format used in DIVA Core releases 1.0 through 6.5.

**Media Format**

Tapes and disks may be formatted as either [AXF \(Archive Exchange Format\)](#) or [Legacy Format](#). The format is set for tape groups and disk arrays during configuration.

**Medium (Media)**

A set of storage resources. Currently DIVA Core provides two types of media: Groups of Tapes and Arrays of Disks. The `DIVA_archiveObject()` and `DIVA_copyToGroup()` requests transfer Virtual Objects to a Medium.

**Migration**

Copying of data from a DIVA Core media to a tape (Archive operation) or from a tape to a DIVA Core media (Restore operation).

**Mutual Exclusion (Mutex)**

Mutual Exclusion (mutex) avoids the simultaneous use of a common resource (that is, mutual exclusion among threads).

**Name**

Part of the access key to an Virtual Object. Names (file names) typically identify the Virtual Object based on the content within the Virtual Object.

**Repack**

Elimination of blank blocks between two Virtual Objects on a tape (these blocks are caused by the deletion of Virtual Objects), by moving the Virtual Objects to a different, empty tape.

**Request**

A request is an operation running in DIVA Core which progresses through steps (migration, transfer, and so on) and ends as either Completed, Aborted, or Canceled.

**Resource**

Used to denote the necessary elements involved for processing requests (for example, Actors, Core Managers, Disks, Drives, and Tapes).

**Set (of Tapes)**

Every tape in a DIVA Core system belongs to one and only one Set. If the tape is not available to DIVA Core, it belongs to Set #0, otherwise it belongs to a set with a strictly positive ID (for example, Set #1). Each [Tape Group](#) is associated with a Set. When the Tape Group needs an additional tape, it takes it from its associated Set.

**Source**

A system that produces data to be archived in the DIVA Core system (for example, video servers, browsing servers, remote computers, and so on). Source Servers can also be used as a [Destination](#) for certain operations.

**Spanning**

Splitting an Virtual Object's components onto several tapes (typically two). This can occur when the component size is larger than the remaining size left on the initial tape.

**Tape Group**

A Tape Group is a logical notion for characterizing a set of Virtual Object instances. This concept has a direct influence on the instance's storage policy for tapes. Instances of the same Tape Group will be stored on the same tapes. However, Virtual Objects cannot have multiple instances stored on the same tape.

Tape Groups are based on the DIVA Core Tape Set. Each tape inserted in the system is assigned to a Set. Tape Groups are then associated with a single Set. Multiple Tape Groups may be associated with the same set. No Tape Group can use the set number 0.

Several kinds of tape can be used in a DIVA Core system. Tape Groups can be defined either by using a Set, in which you assign only tapes of the same type, or by defining the Set in which you can mix tape types. Therefore, the first case specifies the tape type that stores the Virtual Object instance. See [Set \(of Tapes\)](#) in this section for more information.

**Transfer**

Copying data from a [Source](#) to a DIVA Core media (Archive operation) or from a DIVA Core media to a [Destination](#) (Restore operation). See [Request](#) for more information.

**UUID (Universally Unique Identifier)**

A UUID (Universally Unique Identifier) uniquely identifies each Virtual Object created in DIVA Core across all Telestream customer sites. Virtual Objects created using the Copy As request are not assigned a UUID. A Virtual Object created by a Copy As request contains the same UUID as that of the Source Server Virtual Object.

**Virtual Object**

Virtual Objects are archive entries in DIVA Core. A Virtual Object is identified by a pair ([Name](#) and [Collection](#)) and contains one or more components. A component is the DIVA Core representation of a file. The components are stored in DIVA Core as an [Virtual Object Instance](#). Also see [Complex Virtual Object](#).

**Virtual Object Instance**

The mapping of an Virtual Object's components onto a set of storage resources belonging to the same storage space. Deleting instances cannot result in deleting the

related Virtual Object and therefore the deletion of an instance, when that instance is unique, is not permitted.