NavisXtend Provisioning Server Programmer's Reference

Ascend Communications, Inc.

Product Code: 80024
Revision 01
December 1997

Copyright © 1997 Ascend Communications, Inc. All Rights Reserved.

This document contains information that is the property of Ascend Communications, Inc. This document may not be copied, reproduced, reduced to any electronic medium or machine readable form, or otherwise duplicated, and the information herein may not be used, disseminated or otherwise disclosed, except with the prior written consent of Ascend Communications, Inc.

ASCEND COMMUNICATIONS, INC. END-USER LICENSE AGREEMENT

ASCEND COMMUNICATIONS, INC. IS WILLING TO LICENSE THE ENCLOSED SOFTWARE AND ACCOMPANYING USER DOCUMENTATION (COLLECTIVELY, THE "PROGRAM") TO YOU ONLY UPON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS AND CONDITIONS OF THIS LICENSE AGREEMENT. PLEASE READ THE TERMS AND CONDITIONS OF THIS LICENSE AGREEMENT CAREFULLY BEFORE OPENING THE PACKAGE(S) OR USING THE ASCEND SWITCH(ES) CONTAINING THE SOFTWARE, AND BEFORE USING THE ACCOMPANYING USER DOCUMENTATION. OPENING THE PACKAGE(S) OR USING THE ASCEND SWITCH(ES) CONTAINING THE PROGRAM WILL INDICATE YOUR ACCEPTANCE OF THE TERMS OF THIS LICENSE AGREEMENT. IF YOU ARE NOT WILLING TO BE BOUND BY THE TERMS OF THIS LICENSE AGREEMENT, ASCEND IS UNWILLING TO LICENSE THE PROGRAM TO YOU, IN WHICH EVENT YOU SHOULD RETURN THE PROGRAM WITHIN TEN (10) DAYS FROM SHIPMENT TO THE PLACE FROM WHICH IT WAS ACQUIRED, AND YOUR LICENSE FEE WILL BE REFUNDED. THIS LICENSE AGREEMENT REPRESENTS THE ENTIRE AGREEMENT CONCERNING THE PROGRAM BETWEEN YOU AND ASCEND, AND IT SUPERSEDES ANY PRIOR PROPOSAL, REPRESENTATION OR UNDERSTANDING BETWEEN THE PARTIES.

1. License Grant. Ascend hereby grants to you, and you accept, a non-exclusive, non-transferable license to use the computer software, including all patches, error corrections, updates and revisions thereto in machine-readable, object code form only (the "Software"), and the accompanying User Documentation, only as authorized in this License Agreement. The Software may be used only on a single computer owned, leased, or otherwise controlled by you; or in the event of inoperability of that computer, on a backup computer selected by you. You agree that you will not pledge, lease, rent, or share your rights under this License Agreement, and that you will not, without Ascend's prior written consent, assign or transfer your rights hereunder. You agree that you may not modify, reverse assemble, reverse compile, or otherwise translate the Software or permit a third party to do so. You may make one copy of the Software and User Documentation for backup purposes. Any such copies of the Software or the User Documentation shall include Ascend's copyright and other proprietary notices. Except as authorized under this paragraph, no copies of the Program or any portions thereof may be made by you or any person under your authority or control.

2. Ascend's Rights. You agree that the Software and the User Documentation are proprietary, confidential products of Ascend or Ascend's licensor protected under US copyright law and you will use your best efforts to maintain their confidentiality. You further acknowledge and agree that all right, title and interest in and to the Program, including associated intellectual property rights, are and shall remain with Ascend or Ascend's licensor. This License Agreement does not convey to you an interest in or to the Program, but only a limited right of use revocable in accordance with the terms of this License Agreement.

3. License Fees. The license fees paid by you are paid in consideration of the license granted under this License Agreement.

4. Term. This License Agreement is effective upon your opening of the package(s) or use of the switch(es) containing Software and shall continue until terminated. You may terminate this License Agreement at any time by returning the Program and all copies or portions thereof to Ascend. Ascend may terminate this License Agreement upon the breach by you of any term hereof. Upon such termination by Ascend, you agree to return to Ascend the Program and all copies or portions thereof. Termination of this License Agreement shall not prejudice Ascend's rights to damages or any other available remedy.

5. Limited Warranty. Ascend warrants, for your benefit alone, for a period of 90 days from the date of shipment of the Program by Ascend (the "Warranty Period") that the program diskettes in which the Software is contained are free from defects in material and workmanship. Ascend further warrants, for your benefit alone, that during the Warranty Period the Program shall operate substantially in accordance with the User Documentation. If during the Warranty Period, a defect in the Program appears, you may return the Program to the party from which the Program was acquired for either replacement or, if so elected by such party, refund of amounts paid by you under this License Agreement. You agree that the foregoing constitutes your sole and exclusive remedy for breach by Ascend of any warranties made under this Agreement. EXCEPT FOR THE WARRANTIES SET FORTH ABOVE, THE PROGRAM IS LICENSED "AS IS", AND ASCEND DISCLAIMS ANY AND ALL OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE AND ANY WARRANTIES OF NONINFRINGEMENT.

6. Limitation of Liability. Ascend's cumulative liability to you or any other party for any loss or damages resulting from any claims, demands, or actions arising out of or relating to this License Agreement shall not exceed the greater of: (i) ten thousand US dollars ($10,000) or (ii) the total license fee paid to Ascend for the use of the Program. In no event shall Ascend be liable for any indirect, incidental, consequential, special, punitive or exemplary damages or lost profits, even if Ascend has been advised of the possibility of such damages.

7. Proprietary Rights Indemnification. Ascend shall at its expense defend you against and, subject to the limitations set forth elsewhere herein, pay all costs and damages made in settlement or awarded against you resulting from a claim that the Program as supplied by Ascend infringes a United States copyright or a United States patent, or misappropriates a United States trade secret, provided that you: (a) provide prompt written notice of any such claim, (b) allow Ascend to direct the defense and settlement of the claim, and (c) provide Ascend with the authority, information, and assistance that Ascend deems reasonably necessary for the defense and settlement of the claim. You shall not consent to any judgment or decree or do any other act in compromise of any such claim without first obtaining Ascend's written consent. In any action based on such a claim, Ascend may, at its sole option, either: (1) obtain for you the right to continue using the Program, (2) replace or modify the Program to avoid the claim, or (3) if neither (1) nor (2) can reasonably be effected by Ascend, terminate the license granted hereunder and give you a prorata refund of the license fee paid for such Program, calculated on the basis of straight-line depreciation over a five-year useful life. Notwithstanding the preceding sentence, Ascend will have no liability for any infringement or misappropriation claim of any kind if such claim is

based on: (i) the use of other than the current unaltered release of the Program and Ascend has provided or offers to provide such release to you for its then current license fee, or (ii) use or combination of the Program with programs or data not supplied or approved by Ascend to the extent such use or combination caused the claim.

8. Export Control. You agree not to export or disclose to anyone except a United States national any portion of the Program supplied by Ascend without first obtaining the required permits or licenses to do so from the US Office of Export Administration, and any other appropriate government agency.

9. Governing Law. This License Agreement shall be construed and governed in accordance with the laws and under the jurisdiction of the Commonwealth of Massachusetts, USA. Any dispute arising out of this Agreement shall be referred to an arbitration proceeding in Boston, Massachusetts, USA by the American Arbitration Association.

10. Miscellaneous. If any action is brought by either party to this License Agreement against the other party regarding the subject matter hereof, the prevailing party shall be entitled to recover, in addition to any other relief granted, reasonable attorneys' fees and expenses of arbitration. Should any term of this License Agreement be declared void or unenforceable by any court of competent jurisdiction, such declaration shall have no effect on the remaining terms hereof. The failure of either party to enforce any rights granted hereunder or to take action against the other party in the event of any breach hereunder shall not be deemed a waiver by that party as to subsequent enforcement of rights or subsequent actions in the event of future breaches.

Contents

About This Guide

What You Need to Know xi
Documentation Reading Path xii
How to Use This Guide xiv
What's New in This Guide xv
Related Documents xix
Conventions xx
Terminology xxi

1 Overview

The NavisXtend Provisioning Server API 1-1
Servicing Asynchronous Functions 1-2
Function Names 1-4
Functions That Take an Argument List 1-4
API Functions by Group 1-4
Reusing an Existing Object ID 1-10

2 C Functions

Session Control Functions 2-4
Operational Functions 2-10
Select Loop Processing Functions 2-64
Utility Functions 2-72

3 C++ Functions

C++ Classes 3-1
Session Control 3-5
Operational Functions 3-10
Select Loop Processing 3-60
General Notes 3-60
CvClient::ObjectId 3-68
CvClient::Args 3-126
CvClient::IpAddr 3-139
CvSVCAddress 3-142

A Sample Code

C Sample Code A-1
C++ Sample Code A-20
Sample Code for SMDS Objects A-20
Sample Code for Other Objects A-38

B Error Codes

Success Condition B-2
Error Codes B-2
NavisXtend Provisioning Server Errors B-73
Overridable Error Messages B-80

Index

List of Figures

Figure 1-1. Flow of Control for an Asynchronous Function 1-3

List of Tables

Table 1-1. Session Control Functions 1-5
Table 1-2. Operational Functions 1-5
Table 1-3. Select Loop Processing Functions 1-6
Table 1-4. Utility Functions 1-7
Table 2-1. Valid Parent and Child Object Types 2-23
Table 2-2. Valid Parent and Child Object Types 2-28
Table 2-3. Valid Parent and Child Object Types 2-50
Table 2-4. Valid Parent and Child Object Types 2-55
Table 3-1. Valid Parent and Child Object Types 3-35
Table 3-2. Valid Parent and Child Object Types 3-39
Table 3-3. Valid Parent and Child Object Types 3-43
Table 3-4. Valid Parent and Child Object Types 3-49
Table 3-5. Calculated nBits Values 3-145

About This Guide

The *NavisXtend Provisioning Server Programmer's Reference* provides detailed information about the functions of the NavisXtend Provisioning Server API. The API is used to develop a *provisioning client* — an application that runs on a workstation in an Ascend network and interacts with a Provisioning Server. Much of the provisioning functionality of CascadeView is available through the API. You use the client to query and configure switch nodes, cards, physical ports, circuits, and other objects.

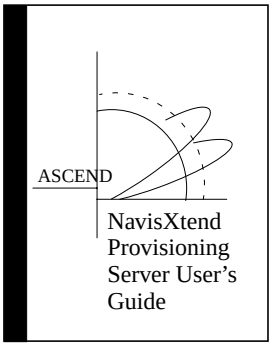
What You Need to Know

This guide assumes that you have a working knowledge of network management and provisioning operations. This guide assumes that you have installed the Ascend switch hardware. To develop a provisioning client, you need to be familiar with programming in C or C++ in a UNIX environment. Programming experience is *not* required if you plan to use the Command Line Interface only.

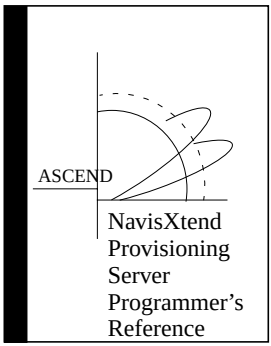
Documentation Reading Path

Before you read this guide, read the *Software Release Notice (SRN) for NavisXtend Provisioning Server* that accompanies the software. The SRN will alert you to any documentation updates or special conditions that you should be aware of.

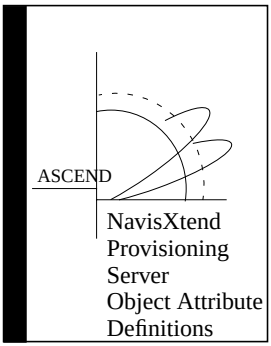
The complete document set for the NavisXtend Provisioning Server includes the following manuals:



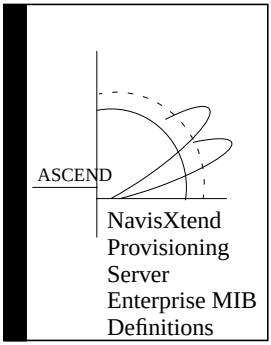
If you are using the NavisXtend Provisioning Server Application Toolkit for the first time, read the entire *NavisXtend Provisioning Server User's Guide*, which describes the interface, features, and typical applications for the NavisXtend Provisioning Server Application Toolkit. It explains, in step-by-step format, what is involved in developing a provisioning client and a provisioning script. It also describes how to use the SNMP MIB.



Once you are ready to begin developing a provisioning client, use this guide for detailed information on the NavisXtend Provisioning Server Application Programming Interface (API).



Use this guide for detailed information on the various object types supported by the Provisioning Server and their associated attributes.



If you are using the SNMP MIB to access the Provisioning Server, use this guide for detailed information on the MIB.

How to Use This Guide

The following table summarizes the information contained in this guide:

Read	To Learn About
Chapter 1	The features of the NavisXtend Provisioning Server API. The chapter lists the API functions by functional group and identifies what function to use for a specific operation.
Chapter 2	The C functions of the NavisXtend Provisioning Server API.
Chapter 3	The C++ classes and member functions of the NavisXtend Provisioning Server API.
Appendix A	Commented source listings of a sample NavisXtend client. Use this information as a guide when you write your program.
Appendix B	Error codes that can be returned by the functions of the NavisXtend Provisioning Server API.

What's New in This Guide

The following table lists the new product features in this release.

Provisioning Server 2.0 New Features	Enables you to
Supports: CBX 500 Switch	Configure elements that are part of CBX 500 switches.
Supports: 8-port ATM E3 Card 8-port ATM DS3 Card 8-port ATM E1 Card 8-port ATM T1 Card 4-port ATM OC3 Card 1-port OC-12c Card	Configure elements of these cards for CBX 500 switches.
Supports: ATM IWU Card ATM CS Card 1-port ATM CS/E3 Card 12-port E1 Card	Configure elements of these cards for B-STDX 9000 switches.
Supports these LPorts and their associated attributes: ATM Direct Cell Trunk LPort ATM OPTimum Frame Trunk LPort ATM OPTimum Cell Trunk LPort SMDS OPTimum Trunk LPort Frame OPTimum Trunk LPort Direct Line Trunk LPort ATM NNI LPort	Configure elements of these LPorts.

Provisioning Server 2.0 New Features	Enables you to
Supports these new objects and their associated attributes: **Network** Customer Network Connection Admission Control (Applies to B-STDX and CBX LPorts only) Service Name SMDS Country Code SVC CUG SVC CUG Member Rule SVC CUG Member SVC Security Screen Traffic Descriptor Virtual Private Network **Switch** SVC Node Prefix **PPort** Automatic Protection Switching Performance Monitor Extended Super Frame Data Link Traffic Shaper **LPort** Assigned SVC Security Screen PMP Circuit Leaf Endpoint PMP Circuit Root Endpoint PMP Spvc Leaf Endpoint PMP Spvc Root Endpoint Soft PVC Endpoint SVC Security Screen Action Param SVC User Part SVC Address SVC Prefix SVC Config	Configure elements of these objects.

Provisioning Server 2.0 New Features	Enables you to
Supports these CBX 500 features: **Switch** Switch Billing configuration **LPort** Direct Versus Virtual UNI CLLM **Circuit** Multihop PVC management **Miscellaneous** Flow Control Processors	Configure elements related to these features.
C Functions CvCreateApsId CvCreateAssignedSvcSecScnId CvCreateCustomerId CvCreateFdlId CvCreateNetCacId CvCreateNullId CvCreatePerformanceMonitorId CvCreatePMPCktLeafId CvCreatePMPCktRootId CvCreatePMPSpvcLeafId CvCreatePMPSpvcRootId CvCreateServiceNameId CvCreateSmdsCountryCodeId CvCreateSpvcId CvCreateSvcAddressId CvCreateSvcConfigId CvCreateSvcCUGId CvCreateSvcCUGMbrId	Create an object ID of a specified type and read a specified value out of the object ID (in support of new objects).

Provisioning Server 2.0 New Features	Enables you to
C Functions *(Continued)* CvCreateSvcCUGMbrRuleId CvCreateSvcNodePrefixId CvCreateSvcPrefixId CvCreateSvcSecScnId CvCreateSvcSecScnActParamId CvCreateSvcUserPartId CvCreateTrafficShaperId CvCreateVirtualUniLPortId CvCreateVPNId CvGetName CvGetName2 CvGetPMPSpvcLeaf CvGetStartVpi CvGetSVCAddress CvGetTrafficShaperId	Create an object ID of a specified type and read a specified value out of the object ID (in support of new objects).
C++ Functions getName getName2 getPMPSpvcLeaf getStartVpi getSVCAddress getTrafficShaperId setName setName2 setNameAndType setPMPSpvcLeafNo setSVCAddressAndType setTrafficShaper setVirtualUniLPort	Create an object ID of a specified type and read a specified value out of the object ID (in support of new objects).
Enterprise-specific MIB	Access the Provisioning Server using an SNMP MIB.

Related Documents

This section lists the related Ascend documentation that may be useful to reference.

- *Network Management Station Installation Guide* (Product code: #80014)
- *Network Configuration Guide for B-STDX/STDX* (Product code: #80017)
- *SYBASE 11 SQL Server Upgrade Guide (*Product code: #80040)
- *Upgrading to Solaris 2.5.1 and HP OpenView 4.11 (*Product code: #80045)
- *Network Configuration Guide for CBX* 500 (Product code: #80049)

Conventions

This guide uses the following conventions to emphasize certain information, such as user input, screen options and output, and menu selections. For example:

Convention	Indicates	Example
`Courier`	Program source code.	`unsigned long`
`Courier Bold`	User input on a separate line and screen or system output.	**`eject cdrom`** **`Please wait...`**
Helvetica	Structure names or other source code in body text.	CvObjectId structure
Boldface	Function name, CLI command, UNIX command, or user input in body text.	**CvCreateNetworkId** **cvaddmember** **select** Type **cd install** and ...
Italics	Variable used by a function or command.	*UserArg* argument
	Book titles, new terms, and emphasized text.	*NavisXtend Provisioning Server User's Guide*
`<key name>`	A keyboard entry.	`<Return>`
Border surrounding text	Notes and warnings.	See the following examples.

Provides helpful suggestions or reference to materials not contained in this manual.

Warns the reader to proceed carefully in order to avoid equipment damage or personal harm.

Terminology

The product name for the Provisioning Server product has changed from the NMS Provisioning Server to the *NavisXtend Provisioning Server*. The *NavisXtend Provisioning Server* is referred to in text using any of the following terms:

- NavisXtend Provisioning Server
- Provisioning Server
- server

The *NavisXtend Provisioning Server Application Toolkit* is referred to in text using any of the following terms:

- NavisXtend Provisioning Server Application Toolkit
- Application Toolkit
- toolkit

The *NavisXtend Provisioning client* is referred to in text using any of the following terms:

- NavisXtend Provisioning client
- Provisioning client
- client
- application

The product name for CascadeView has changed to *NavisCore*. The old and new product names are used interchangeably in the software and in the manuals.

1

Overview

This chapter provides an overview of the API functions.

The NavisXtend Provisioning Server API

The NavisXtend Provisioning Server Application Toolkit provides API functions that allow your application to provision network components and update the CascadeView database.

For the convenience of the programmer, the API functions are available in various versions. For example, the Application Toolkit provides both a C and C++ interface for each API function.

In addition, each operational function is available as an *asynchronous* and a *synchronous* function:

- With the asynchronous function, the application continues with other work while your request is being processed. The asynchronous function takes a callback function to deliver the response.

- With the synchronous function, the application waits for the request to complete before continuing with other processing. This is also known as a *blocking* request, because each function blocks to completion.

Synchronous functions involve less coding but may also be less efficient, because the application waits while they are processed.

Asynchronous functions allow your application to continue processing rather than wait for completion. However, asynchronous functions require additional coding. The application programmer must provide the callback handler and must ensure that the application invokes the API appropriately when the response returns from the server. Specifically, the application must be built around the UNIX **select** system call and must ensure that all processing is done in between calls to **select**.

Servicing Asynchronous Functions

To support asynchronous functions, the API must have access to the **select** loop *(refer* to Figure 1-1).

During an asynchronous function, you take the following actions:

After you issue an asynchronous function, issue **CvGetSelectInfo** (or **CvClient::getSelectInfo**) whenever you want to check if data is ready to return. This function obtains the information needed to pass to a **select** call (**Step 3**). The application needs to know what file descriptor is being used for the client/server communications messages.

When activity is found on a file descriptor, issue **CvProcessEvents** (or **CvClient::processEvents**). This function goes into the API library to receive and process the response (**Step 6**). In turn, the API passes the response to the client response handler (**Step 7**). **CvProcessEvents** can time out if there is no reply within a specified interval.

*If your application does not need access to the **select** loop, issue **CvProcessEvents(-1)** to get each response for an outstanding asynchronous request.*

For more information on these functions in C, refer to "Operational Functions" in Chapter 2. For more information on these functions in C++, refer to "CvClient::Callback" in Chapter 3.

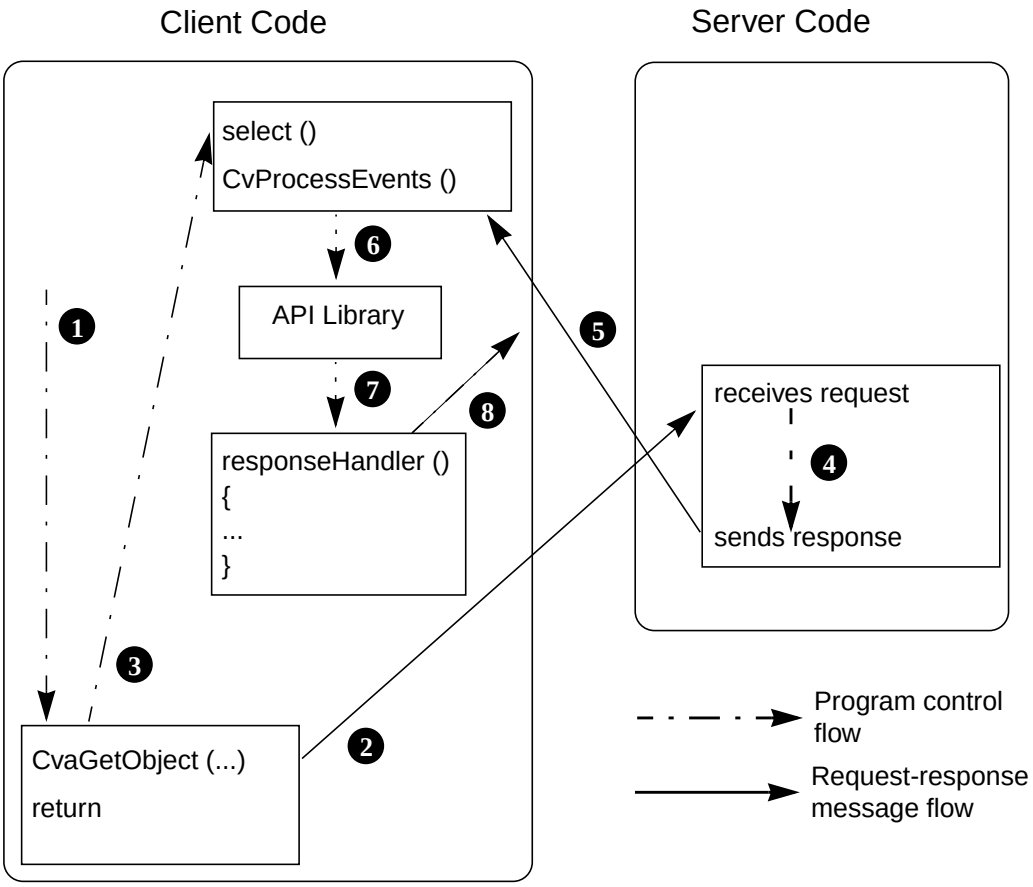


Figure 1-1. Flow of Control for an Asynchronous Function

Function Names

In C, the synchronous and asynchronous functions are differentiated by their function names. The name of each synchronous function contains the **s** character before the verb; the name of each asynchronous function contains the **a** character before the verb. For example, **CvsAddObject** is the synchronous version and **CvaAddObject** is the asynchronous version of the same function.

In C++, function names are overloaded to distinguish between the synchronous and asynchronous functions.

Functions That Take an Argument List

In C, the Application Toolkit provides two options for specifying an argument list. Before you issue a function that takes an argument list, you can either:

- Issue a single function (**CvArgsMakeVals** or **CvArgsMakeIds**) that takes a variable number of arguments and builds the required data structure.
- Issue a series of utility functions that create (**CvArgsMake**) and fill in (**CvArgsSet***AttrType*) the required data structure.

In C++, multiple constructors for the argument object (**CvClient::Args**) handle variable argument lists.

API Functions by Group

The API functions are divided into the following groups:

Session Control Functions — Open and close sockets and control session settings.

Operational Functions — Perform provisioning operations on the network components.

Select Loop Processing Functions — Support loop processing of the select system call.

Utility Functions — Build argument lists, handle initialization, and manage storage.

The following tables describe the API functions by group. For a complete description of each function, refer to the appropriate function section in Chapter 2 or Chapter 3.

Table 1-1. Session Control Functions

Operation	C Function	CvClient Member Function	Description
Connect	**CvConnect**	**open**	Establishes a session with the Provisioning Server.
Close	**CvClose**	**close**	Terminates a session with the Provisioning Server.
Set Update Type	**CvSetModifyType**	**setModifyType**	Specifies whether updates are made to the network component and database, or to the database only.

Table 1-2. Operational Functions

Operation	C Function	CvClient Member Function	Description
Add	**CvaAddObject** **CvsAddObject**	**addObject**	Creates an object in the database and (optionally) in the switch.
Add Member	**CvaAddMember** **CvsAddMember**	**addMember**	Adds an address to a screen and netwide group address.
Modify	**CvaModifyObject** **CvsModifyObject**	**modifyObject**	Modifies specific attributes of an object in the database and (optionally) in the switch.
Delete	**CvaDeleteObject** **CvsDeleteObject**	**deleteObject**	Deletes an object in the database and (optionally) in the switch.
Delete Member	**CvaDeleteMember** **CvsDeleteMember**	**deleteMember**	Deletes an address from a screen and netwide group address.
Get	**CvaGetObject** **CvsGetObject**	**getObject**	Retrieves the values of specific attributes.

Table 1-2. Operational Functions (Continued)

Operation	C Function	CvClient Member Function	Description
List Contained	**CvaListContained Objects** **CvsListContained Objects**	**listContainedObjects**	Queries the database for a list of objects of a specified object type that are contained by a specified parent object.
List All Contained	**CvaListAllContained Objects** **CvsListAllContained Objects**	**listContainedObjects**	Queries the database for a list of objects of any object type that are contained by a specified parent object.
List Next	**CvsNextObject**	**nextObject**	Retrieves the next object in a list of objects.
Get Error Msg	**CvGetErrorMsg**	**getErrorMsg**	Returns an error message.
Get	**CvGetResponseArgs**	**getResponseArgs**	Returns the argument list returned by a synchronous function.

Table 1-3. Select Loop Processing Functions

Operation	C Function	CvClient Member Function	Description
Client Callback Handler	**CVCBH**	**Callback** (virtual member function)	Is the prototype for a function supplied by the client. The API calls this callback handler to deliver a response to an asynchronous request.
Obtains information for select	**CvGetSelectInfo**	**getSelectInfo**	Obtains information needed to pass to a **select** call after the completion of an asynchronous function.
Processes activity	**CvProcessEvents**	**processEvents**	Processes activity on file descriptors in order to receive responses to an asynchronous function.
Determines timeout	**CvTimeout**	**timeout**	Determines whether an outstanding asynchronous request timed out.

Table 1-4. Utility Functions

Operation	C Function	CvClient Member Function	Description
Make	**CvArgsMake** **CvArgsMakeIds** **CvArgsMakeVals**	Construct an **Args** class.	Creates argument list.
Free	**CvArgsFree**	—	Deletes a pointer to an argument list.
Set	**CvArgsSetInt** **CvArgsSetStr** **CvArgsSetIpAddr** **CvArgsSetObjectId** **CvArgsSetNull**	operator **[ ]**	Sets a value for a given argument ID in the argument list. If a value already exists for that ID, the existing value is replaced with the given value.
Get	**CvArgsGetInt** **CvArgsGetStr** **CvArgsGetObjectId** **CvArgsGetIpAddr**	operator **[ ]**	Reads a value for a given argument ID out of the argument list.
Count	**CvArgsCount**	**Args::count**	Retrieves the number of arguments in the argument list.
Get ID At Index	**CvArgsIdAt**	**Args::idAt**	Retrieves the argument ID at a specified index location in the argument list.
Get Value At Index	—	**Args::valueAt**	Retrieves the argument value at a specified index location in the argument list.
Get Type At Index	**CvArgsTypeAt**	**Args::getType**	Retrieves a value's type at a specified index location in the argument list.
Get Error Index	**CvArgsErrorIndex**	**Args::errorIndex**	Determines if any argument in an argument list has an error status.
Test for Existence	—	**Args::exists**	Determines if a specified argument exists in an argument list.

Table 1-4. Utility Functions (Continued)

Operation	C Function	CvClient Member Function	Description
Get Status	**CvArgsGetStatus** **CvArgsStatusAt**	**Args::getStatus** **Args::statusAt**	Returns the error status code of a specified argument in an argument list.
Compare	**CvArgsEqual**	operator **=** operator **!=**	Compares two argument lists for equality.
Add	**CvArgsCombine**	operator +	Adds two argument lists together, resulting in a new argument list that combines both sets of arguments.
Subtract	**CvArgsRemove**	operator **—**	Subtracts two argument lists, resulting in a new argument list that contains the arguments from the first set that are not in the second set.
Intersect	**CvArgsSelect**	operator *****	Returns the intersection of two argument lists, resulting in a new argument list that contains the arguments that existed in both sets of arguments.
Convert	**CvArgsToString**	—	Converts an argument list to a printable string format.
Delete	**CvStringFree**	—	Deletes a pointer to a string.
Make	**CvAddArgumentBy NameValue**	—	Adds an argument to an argument list by specifying a textual argument name and value.
Get	**CvAddArgumentBy Name**	—	Adds an argument to an argument list by specifying a textual argument name.
Print	**CvArgsPrint**	operator <<	Prints the text description of an argument list to a file.
Make	**CvCreate***ObjectType***Id**	**set***ObjectType*	Creates an object ID of a specified type, based on the containment hierarchy.

Table 1-4. Utility Functions (Continued)

Operation	C Function	CvClient Member Function	Description
Make	**CvCreateNetworkIdFromString** **CvCreateSwitchIdFromString**	construct an **IpAddr** class.	Creates an object ID of a specified type, based on a text argument description.
Convert	**CvObjectIdToString**	operator <<	Converts an object ID to text argument descriptions.
Convert	**CvGetArgumentName**	**Args::idAt**	Converts an argument ID to a text description.
Convert	**CvGetEnumName**	**Args::valueAt**	Converts an enumerated value to a text description.
Convert	**CvGetObjectTypeName**	operator <<	Converts an object type to a text description.
Convert	**CvParseObjectId**	—	Converts text descriptions to an object ID.
Convert	**CvParseObjectType**	—	Converts a text description to an object type.
Get	**CvGet***ObjectType Value*	**get***ObjectType*	Reads a specified value out of the object ID.
Print	**CvObjectIdPrint**	operator <<	Prints the text description of an object ID to a file.

Reusing an Existing Object ID

The C++ API allows you to use the **set***ObjectType* family of functions to modify fields in an object ID. There are important steps that you must take if you want to make certain modifications to an existing object ID. Specifically, if you want to:

- Change any type of LPort to an atmXportForFrNni LPort, or vice versa
- Change a non-ATM circuit endpoint to an ATM circuit endpoint, or vice versa

You must:

1. Set to 0 any service-specific fields that do not apply to the new object.
2. Set appropriate values to the other service-specific fields.

Example 1

You have an object that is currently a non-ATM XportForFrNni LPort, and you want to modify the object ID to specify an atmXportForFrNni LPort. Use the **set***ObjectType* family of functions to set the LPort field in the object ID to 0 and then set the appropriate fields to the new values you want.

The current object is expressed as:

```
switch.1.1.1.1.card.5.pport.4.lport.1
```

To modify the object ID, perform the following steps:

1. Issue **setLPort** to set the LPort field of the object ID to 0. Specify 0 for the port number of the LPort:

   ```
   setLPort(0);
   ```

2. Issue **setAtmInfo** to set the new values for both the VPI and VCI numbers:

   ```
   setAtmInfo(1,32);
   ```

3. Issue **setType** to set the type of the object ID to the LPort type:

   ```
   setType(CVT_LPort);
   ```

The new object is expressed as:

```
switch.1.1.1.1.card.5.pport.4.vpi.1.vci.32
```

Example 2

You have an object that is currently an ATM endpoint of a circuit, and you want to modify the object ID to specify a Frame Relay endpoint of a circuit. Use the **set***ObjectType* family of functions to set the ATM fields in the object ID to 0 and then set the appropriate fields to the new values you want.

The current object is expressed as:

```
switch.1.1.1.1.card.5.pport.4.lport.1.vpi.2.vci.55
```

To modify the object ID, perform the following steps:

1. Issue **setAtmInfo** to set the circuit field of the object ID to 0. Specify 0 for both the VPI and VCI numbers:

   ```
   setAtmInfo(0,0);
   ```

2. Issue **setDlci** to set the DLCI number of the object ID:

   ```
   setDlci(100);
   ```

3. Issue **setType** to set the type of the object ID to the circuit type:

   ```
   setType(CVT_Circuit);
   ```

The new object is expressed as:

```
switch.1.1.1.1.card.5.pport.4.lport.1.dlci.100
```

2

C Functions

This chapter describes the C functions of the Provisioning Server API.

The functions are presented by functional group:

Session Control functions — Open and close sockets and control session settings.

Operational functions — Perform provisioning operations on the network components.

Select Loop Processing functions — Support loop processing of the **select** system call.

Utility functions — Build argument lists, handle initialization, and manage storage.

Each functional group starts with a contents listing of the functions, followed by the alphabetic presentation of the functions.

The master contents list of all the functions follows.

Master List of Functions

Session Control Functions

CvClose 2-5
CvConnect 2-6
CvSetModifyType 2-8

Operational Functions

CvaAddMember 2-11
CvaAddObject 2-13
CvaDeleteMember 2-16
CvaDeleteObject 2-18
CvaGetObject 2-20
CvaListAllContainedObjects 2-22
CvaListContainedObjects 2-26
CvaModifyObject 2-33
CvGetErrorMsg 2-36
CvGetResponseArgs 2-37
CvsAddMember 2-38
CvsAddObject 2-40
CvsDeleteMember 2-43
CvsDeleteObject 2-45
CvsGetObject 2-47
CvsListAllContainedObjects 2-49
CvsListContainedObjects 2-54
CvsModifyObject 2-60
CvsNextObject 2-62

Select Loop Processing Functions

CVCBH (Client Callback Handler) 2-65
CvGetSelectInfo 2-67
CvProcessEvents 2-69
CvTimeout 2-71

Utility Functions

CvAddArgumentByName 2-75
CvAddArgumentByNameValue 2-76
CvArgsCombine 2-77
CvArgsCount 2-78
CvArgsEqual 2-79
CvArgsErrorIndex 2-80
CvArgsFree 2-81
CvArgsGetInt 2-82
CvArgsGetIpAddr 2-83
CvArgsGetObjectId 2-84
CvArgsGetStatus 2-85
CvArgsGetStr 2-86
CvArgsIdAt 2-87
CvArgsMake 2-88
CvaArgsMakeIds 2-89
CvaArgsMakeVals 2-90
CvArgsPrint 2-92
CvArgsRemove 2-93
CvArgsSelect 2-94
CvArgsSetInt 2-95
CvArgsSetIpAddr 2-96
CvArgsSetNull 2-97
CvArgsSetObjectId 2-98
CvArgsSetStr 2-99
CvArgsStatusAt 2-100
CvArgsToString 2-101
CvArgsTypeAt 2-102
CvCreateApsId 2-103
CvCreateAssignedSvcSecScnId 2-104
CvCreateAtmEndpointId 2-106
CvCreateAtmLPortId 2-108
CvCreateCardId 2-110
CvCreateChannelId 2-111
CvCreateCustomerId 2-112
CvCreateDlciId 2-113
CvCreateFdlId 2-115
CvCreateLPortId 2-116
CvCreateNetCacId 2-118

CvCreateNetworkId 2-119
CvCreateNetworkIdFromString 2-120
CvCreateNullId 2-122
CvCreatePerformanceMonitorId 2-123
CvCreatePMPCktLeafId 2-124
CvCreatePMPCktRootId 2-126
CvCreatePMPSpvcLeafId 2-128
CvCreatePMPSpvcRootId 2-129
CvCreatePPortId 2-131
CvCreateServiceNameId 2-132
CvCreateSmdsAlienGroupAddrId 2-133
CvCreateSmdsAlienIndivAddrId 2-135
CvCreateSmdsCountryCodeId 2-137
CvCreateSmdsGroupAddrId 2-138
CvCreateSmdsGroupScreenId 2-140
CvCreateSmdsIndivScreenId 2-141
CvCreateSmdsLocalIndivAddrId 2-142
CvCreateSmdsNetwideGroupAddrId 2-144
CvCreateSmdsPrefixId 2-146
CvCreateSmdsSsiIndivAddrId 2-148
CvCreateSpvcId 2-150
CvCreateSvcAddressId 2-152
CvCreateSvcConfigId 2-154
CvCreateSvcCUGId 2-155
CvCreateSvcCUGMbrId 2-156
CvCreateSvcCUGMbrRuleId 2-158
CvCreateSvcNodePrefixId 2-159
CvCreateSvcPrefixId 2-161
CvCreateSvcSecScnId 2-163
CvCreateSvcSecScnActParamId 2-164
CvCreateSvcUserPartId 2-165
CvCreateSwitchId 2-167
CvCreateSwitchIdFromString 2-168
CvCreateTrafficDescId 2-169
CvCreateTrafficShaperId 2-170
CvCreateVirtualUniLPortIId 2-172
CvCreateVPNId 2-174
CvGetAddress 2-175
CvGetArgumentName 2-176
CvGetCardNumber 2-177
CvGetChannelNumber 2-178
CvGetDlciNumber 2-179
CvGetEnumName 2-180
CvGetLPortNumber 2-181
CvGetName 2-182
CvGetName2 2-183
CvGetNetworkNumber 2-184
CvGetObjectType 2-185
CvGetObjectTypeName 2-186
CvGetPMPSpvcLeafId 2-187
CvGetPPortNumber 2-188
CvGetPrefix 2-189
CvGetStartVpi 2-190
CvGetSvcAddress 2-191
CvGetSwitchId 2-192
CvGetTrafficShaperId 2-193
CvGetVciNumber 2-194
CvGetVpiNumber 2-195
CvObjectIdPrint 2-196
CvObjectIdToString 2-197
CvParseObjectId 2-198
CvParseObjectType 2-199
CvStringFree 2-200

Session Control Functions

CvClose 2-5
CvConnect 2-6
CvSetModifyType 2-8

Purpose

Terminates a session with the Provisioning Server.

Syntax

CvClose(CvServerHandle);

Description

CvClose terminates a session previously established with the Provisioning Server.

Handle is the server handle returned by **CvConnect**. This server handle is no longer valid if **CvClose** is successful.

Prerequisites

You must establish a session with the Provisioning Server with **CvConnect**.

Post-processing

None.

See Also

CvConnect.

CvConnect

Purpose

Establishes a session with the Provisioning Server.

Syntax

CvServerHandle CvConnect(
CvAddrName *Host*,
CvIpPort *RemotePort*,
**const char ***_UserName_,
const char ***_Password_ **);

Description

CvConnect establishes a session between the Provisioning client and the Provisioning Server.

Host is the IP address of the host on which the server is running. Specify the address as a string. To use a local host, set this value to null.

RemotePort identifies the remote port to be used for communication between the client and server. To use a default port, set this value to **CvDefaultServerPort**.

UserName is a pointer to the name of the user starting the session. This value is the same as that required by CascadeView.

Password is a pointer to the password required to start the session. This value is the same as that required by CascadeView.

Notes

Each of the operational functions takes a CvServerHandle argument that identifies the session with the Provisioning Server.

You can have multiple sessions open at a time.

UserName and *Password* are optional parameters. By default, the Provisioning Server accepts requests from clients without requiring authorization. To implement a security feature that authenticates user logins, set the environment variable CV_PSRV_USE_LOGINS (refer to Chapter 2 of the *NavisXtend Provisioning Server User's Guide* for details). When issuing this function, pass in the character strings required by CascadeView for the *UserName* and *Password* arguments.

The UserName and Password character strings are sent over the network as nonencrypted text.

Prerequisites None.

Post-processing None.

See Also **CvClose**.

CvSetModifyType

Purpose

Specifies whether updates are made to the network component and database, or to the database only.

Syntax

CvModifyType CvSetModifyType(
CvServerHandle *Handle***,**
CvModifyType *ModifyType* **);**

Description

CvSetModifyType specifies whether updates are made to the network component and database, or to the database only. The function returns the update setting that was previously in effect.

Handle is the CvServerHandle variable that identifies this session with the Provisioning Server. This handle is returned by **CvConnect**.

ModifyType indicates the update method, as follows:

modifyNormal — Sends updates to both the network component and the database. The database is updated only if the network component updates successfully.

modifyDbOnly — Sends updates to the database only.

modifyDbOnlySetOutSync — Sends updates to the database only and sets a flag indicating that the object is out of synchronization in the database.

Notes

Once you issue this function, the update setting remains in effect until either the session is terminated or the function is issued with another setting.

To determine if the setting was changed since your last session, compare *ModifyType* to the **CvModifyType** returned by the function.

Prerequisites You must establish a session with the Provisioning Server with **CvConnect**.

Post-processing None.

Operational Functions

CvaAddMember 2-11
CvaAddObject 2-13
CvaDeleteMember 2-16
CvaDeleteObject 2-18
CvaGetObject 2-20
CvaListAllContainedObjects 2-22
CvaListContainedObjects 2-26
CvaModifyObject 2-33
CvGetErrorMsg 2-36
CvGetResponseArgs 2-37
CvsAddMember 2-38
CvsAddObject 2-40
CvsDeleteMember 2-43
CvsDeleteObject 2-45
CvsGetObject 2-47
CvsListAllContainedObjects 2-49
CvsListContainedObjects 2-54
CvsModifyObject 2-60
CvsNextObject 2-62

Purpose

Adds a member to a list of objects.

Syntax

CvReqId CvaAddMember(
CvServerHandle *Handle*,
**const CvObjectId ***Container*,*
**const CvObjectId ***Member*,*
CVCBH *Callback*,
void ***UserArg* **);

Description

CvaAddMember is an asynchronous function that adds an address to a screen or netwide group address. Upon completion of the function, the address represented by the second object ID argument is added to the object specified by the first object ID argument.

Handle is the CvServerHandle variable that identifies this session with the Provisioning Server. This handle is returned by **CvConnect**.

Container is a pointer to the CvObjectId structure that specifies the container object, which is the screen or netwide group address.

Member is a pointer to the CvObjectId structure that specifies the member object, which is the address to become a member of the screen or netwide group address.

Callback is the callback function.

UserArg is a user-context argument. Use this argument to pass any value to the callback handler.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

When you specify the object CVT_SmdsGroupScreen as the container object, the member to be added must be either a CVT_SmdsAlienGroupAddress or a CVT_SmdsSwitchGroupAddress.

When you specify the object CVT_SmdsIndividualScreen as the container object, the member to be added must be either a CVT_SmdsLocalIndividualAddress or a CVT_SmdsAlienIndividualAddress.

When you specify the object CVT_SmdsNetwideGroupAddress as the container object, the member to be added must be a CVT_SmdsLocalIndividualAddress.

Prerequisites

You must establish a session with the Provisioning Server with **CvConnect**.

Make sure that the object IDs represented by *Container* and *Member* are valid.

Post-processing

The application must return to the **select** loop. Invoke **CvProcessEvents** after activity is found on the file descriptor.

When issuing multiple requests, use the returned **CvReqId** to verify or look up the original request. Compare this **CvReqId** to the **CvReqId** of the callback function to associate a response with its original request.

See Also

CvsAddMember.

Purpose

Creates an object in the database and (optionally) in the switch.

Syntax

```
CvReqId CvaAddObject(
        CvServerHandle Handle,
        const CvObjectId *ObjId,
        CVCBH Callback,
        void *UserArg,
        CvConstArgs Args );
```

Description

CvaAddObject is an asynchronous function that creates an object in the database and (optionally) in the switch. The attributes specified by the function are used to initialize the object.

Handle is the CvServerHandle variable that identifies this session with the Provisioning Server. This handle is returned by **CvConnect**.

ObjId is a pointer to the CvObjectId structure that specifies the object to be added.

Callback is the callback function.

UserArg is a user-context argument. Use this argument to pass any value to the callback handler.

Args is a constant CvArgs structure that specifies the attributes for the object to be added. The API does not modify or delete this argument. This structure is created by either **CvArgsMake** or **CvArgMakeVals**. Specify only those attributes appropriate for the object type (refer to the *NavisXtend Provisioning Server Object Attribute Definitions*). You can specify any attributes except those with the Read-Only access restriction.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

To create a card or PPort, use a Modify function (**CvaModifyObject** or **CvsModifyObject**). The CascadeView database automatically populates each switch with cards of type "empty." Use the Modify function to modify the card's type from "empty" to a specified type. Likewise, once a card has been configured, CascadeView automatically populates the card with all necessary physical ports. Use the Modify function to change the PPort specifications. In the case of the channelized DS3 card, once the card has been configured, CascadeView automatically populates the card with all necessary channels. Use the Modify function to change the channel specifications.

If you omit a required parameter, the function response will have the status **cveArgumentNotPresent**. Check the argument list returned in the response. Any attribute that is missing a valid value is a required attribute that you omitted.

Prerequisites

You must establish a session with the Provisioning Server with **CvConnect**.

Make sure that the object ID is valid.

You must build the argument list. To do so, either:

- Issue a single function (**CvArgsMakeVals**) that takes a variable number of arguments and builds the CvArgs structure.
- Issue a series of utility functions that create (**CvArgsMake**) and fill in (**CvArgsSet***AttrType*) the CvArgs structure.

Post-processing

When you are finished with the function's returned value, you must issue **CvArgsFree** to free the memory used by the argument list.

The application must return to the **select** loop. Invoke **CvProcessEvents** after activity is found on the file descriptor.

When issuing multiple requests, use the returned **CvReqId** to verify or look up the original request. Compare this **CvReqId** to the **CvReqId** of the callback function to associate a response with its original request.

See Also

CvaModifyObject, **CvsAddObject**, and **CvsModifyObject**.

CvaDeleteMember

Purpose

Removes a member from a list of objects.

Syntax

CvReqId CvaDeleteMember(
CvServerHandle *Handle*,
**const CvObjectId ***Container*,*
**const CvObjectId ***Member*,*
CVCBH *Callback*,
void ***UserArg **);

Description

CvaDeleteMember is an asynchronous function that removes an address from a screen or netwide group address. Upon completion of the function, the address represented by the second object ID argument is removed from the object specified by the first object ID argument.

Handle is the CvServerHandle variable that identifies this session with the Provisioning Server. This handle is returned by **CvConnect**.

Container is a pointer to the CvObjectId structure that specifies the container object, which is the screen or netwide group address.

Member is a pointer to the CvObjectId structure that specifies the address to be removed.

Callback is the callback function.

UserArg is a user-context argument. Use this argument to pass any value to the callback handler.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

When you specify the object CVT_SmdsGroupScreen as the container object, the member to be removed must be either a CVT_SmdsAlienGroupAddress or a CVT_SmdsSwitchGroupAddress.

When you specify the object CVT_SmdsIndividualScreen as the container object, the member to be removed must be either a CVT_SmdsLocalIndividualAddress or a CVT_SmdsAlienIndividualAddress.

When you specify the object CVT_SmdsNetwideGroupAddress as the container object, the member to be removed must be a CVT_SmdsLocalIndividualAddress.

Prerequisites

You must establish a session with the Provisioning Server with **CvConnect**.

Make sure that the object IDs represented by *Container* and *Member* are valid.

Post-processing

The application must return to the **select** loop. Invoke **CvProcessEvents** after activity is found on the file descriptor.

When issuing multiple requests, use the returned **CvReqId** to verify or look up the original request. Compare this **CvReqId** to the **CvReqId** of the callback function to associate a response with its original request.

See Also

CvsDeleteMember.

CvaDeleteObject

Purpose

Deletes an object from the database and (optionally) from the switch.

Syntax

CvReqId CvaDeleteObject(
CvServerHandle *Handle*,
**const CvObjectId ***_ObjId_,
CVCBH *Callback*,
void ***_UserArg_ **);

Description

CvaDeleteObject is an asynchronous function that deletes an object from the database and (optionally) from the switch.

Handle is the CvServerHandle variable that identifies this session with the Provisioning Server. This handle is returned by **CvConnect**.

ObjId is a pointer to the CvObjectId structure that specifies the object to be deleted.

Callback is the callback function.

UserArg is a user-context argument. Use this argument to pass any value to the callback handler.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

You only need to delete an SMDS switch group address if the database shows an SMDS switch group address that should not exist.

To remove a card, use a Modify function (**CvaModifyObject** or **CvsModifyObject**) to modify the card's type to "empty."

You cannot delete some objects until the objects they contain have been deleted. For example, you cannot delete an LPort until all circuits and addresses defined on that LPort have been deleted.

Prerequisites

You must establish a session with the Provisioning Server with **CvConnect**.

Make sure that the object ID is valid.

Post-processing

The application must return to the **select** loop. Invoke **CvProcessEvents** after activity is found on the file descriptor.

When issuing multiple requests, use the returned **CvReqId** to verify or look up the original request. Compare this **CvReqId** to the **CvReqId** of the callback function to associate a response with its original request.

See Also

CvaModifyObject, **CvsDeleteObject**, and **CvsModifyObject**.

CvaGetObject

Purpose

Retrieves the values of specific attributes. The values are retrieved from the database.

Syntax

CvReqId CvaGetObject(
CvServerHandle *Handle*,
**const CvObjectId ***ObjId*,*
CVCBH *Callback*,
**void ***UserArg*,*
CvConstArgs *Args* **);**

Description

CvaGetObject is an asynchronous function that retrieves the values of specific attributes.

Handle is the CvServerHandle variable that identifies this session with the Provisioning Server. This handle is returned by **CvConnect**.

ObjId is a pointer to the CvObjectId structure that specifies the object whose attributes are to be retrieved.

Callback is the callback function.

UserArg is a user-context argument. Use this argument to pass any value to the callback handler.

Args is a constant CvArgs structure that specifies the attributes to be retrieved. The API does not modify or delete this argument. This structure is created by either **CvArgsMake** or **CvArgsMakeIds**. Specify only those attributes appropriate for the object type (refer to the *NavisXtend Provisioning Server Object Attribute Definitions*). Specify only attribute IDs with null values.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

Prerequisites

You must establish a session with the Provisioning Server with **CvConnect**.

Make sure that the object ID is valid.

You must build the argument list. To do so, either:

- Issue a single function (**CvArgsMakeIds**) that takes a variable number of arguments and builds the CvArgs structure.
- Issue a series of utility functions that create (**CvArgsMake**) and fill in (**CvArgsSet***AttrType*) the CvArgs structure.

Post-processing

When you are finished with the function's returned value, you must issue **CvArgsFree** to free the memory used by the argument list.

The application must return to the **select** loop. Invoke **CvProcessEvents** after activity is found on the file descriptor.

When issuing multiple requests, use the returned **CvReqId** to verify or look up the original request. Compare this **CvReqId** to the **CvReqId** of the callback function to associate a response with its original request.

See Also

CvsGetObject.

CvaListAllContainedObjects

Purpose

Retrieves a list of objects of any type that are immediate children of a specified object.

Syntax

CvReqId CvaListAllContainedObjects(
CvServerHandle *Handle*,
**const CvObjectId ***_ObjId_,
CVCBH *Callback*,
void ***_UserArg_ **);

Description

CvaListAllContainedObjects is an asynchronous function that retrieves a list of objects of any type that are immediate children of a specified object. Each object is returned in a separate response.

Handle is the CvServerHandle variable that identifies this session with the Provisioning Server. This handle is returned by **CvConnect**.

ObjId is a pointer to the CvObjectId structure that specifies the parent object. The parent object is always the immediate parent of the contained objects (such as a PPort that is a parent of multiple LPorts).

Callback is the callback function.

UserArg is a user-context argument. Use this argument to pass any value to the callback handler.

Notes

You can issue the function on either:

Network level — The function retrieves a list of objects on a network, including all subnets. To issue the function on a network level, specify an IP address, such as 128.100.0.0.

Subnet level — The function retrieves a list of objects on a particular subnet. To issue the function on a subnet level, specify an IP address with a subnet number, such as 128.100.111.0.

Table 2-1 lists the valid parent and child object types you can specify with this function.

.

Table 2-1. Valid Parent and Child Object Types

Parent Object Type	Child Object Types
Network	Switch VPN Customer SmdsCountryCode SmdsNetwideGroupAddress SvcSecScn SvcCUG SvcCUGMbrRule TrafficDesc NetCac ServiceName
Customer	LPort Circuit
VPN	LPort Circuit
SvcCUGMbrRule	SvcCUGMbr
SvcCUG	SvcCUGMbr
Switch	Card SvcNodePrefix SmdsSwitchGroupAddress SmdsAddressPrefix SmdsAlienIndividualAddress SmdsAlienGroupAddress
Card	PPort
PPort	LPort Channel PFdl (8-port ATM T1 card only) PerformanceMonitor Aps (1-port OC-12c/STM-4 card only)

Table 2-1. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
Channel (valid for channelized DS3 card only)	LPort
LPort	Circuit SvcPrefix SvcAddress SvcUserPart SvcConfig SmdsLocalIndividualAddress SmdsIndividualScreen SmdsGroupScreen PMPCktRoot AssignedSvcSecScn SvcSecScnActParam Spvc PMPSpvcRoot
SmdsIndividualScreen	SmdsLocalIndividualAddress SmdsAlienIndividualAddress
SmdsGroupScreen	SmdsSwitchGroupAddress SmdsAlienGroupAddress
SmdsNetwideGroupAddress	SmdsSwitchGroupAddress SmdsLocalIndividualAddress
PMPCktRoot	PMPCktLeaf
PMPSpvcRoot	PMPSpvcLeaf
ServiceName	Circuit

Prerequisites

You must establish a session with the Provisioning Server with **CvConnect**.

Make sure that the object ID is valid.

Post-processing

The application must return to the **select** loop. Invoke **CvProcessEvents** after activity is found on the file descriptor.

This function makes multiple callbacks to the callback handler, each representing an object contained by the parent:

- In the callback function, the *ObjId* parameter represents the listed object.
- The last call to the callback handler returns the status **cveListDone**. The call does not include a valid object.
- A call to the callback handler that returns a status other than **cveNoError** or **cveListDone** indicates that an error has occurred. No more responses will be made for this request.

When issuing multiple requests, use the returned **CvReqId** to verify or look up the original request. Compare this **CvReqId** to the **CvReqId** of the callback function to associate a response with its original request.

See Also

CvaListContainedObjects, **CvsListAllContainedObjects**, and **CvsListContainedObjects**.

CvaListContainedObjects

Purpose

Retrieves a list of objects of a specified object type that are contained by another object.

Syntax

CvReqId CvaListContainedObjects(
CvServerHandle *Handle*,
**const CvObjectId ***_ObjId_,
CvObjectType *ObjType*,
CVCBH *Callback*,
**void ***_UserArg_,
CvConstArgs *Args* **);**

Description

CvaListContainedObjects is an asynchronous function that retrieves a list of objects contained by another object. The returned objects are those that match a specified object type. Each object is returned in a separate response.

Handle is the CvServerHandle variable that identifies this session with the Provisioning Server. This handle is returned by **CvConnect**.

ObjId is a pointer to the CvObjectId structure that specifies the parent object. The parent object can be the immediate parent of the contained objects (such as a PPort that is a parent of multiple LPorts) or the parent object can be positioned higher in the containment hierarchy (such as a switch that is a parent of multiple LPorts).

ObjType is the enumerated value that specifies the type of the objects to be retrieved.

Callback is the callback function.

UserArg is a user-context argument. Use this argument to pass any value to the callback handler.

Args is a constant CvArgs structure that specifies the attributes to be retrieved. The API does not modify or delete this argument. This structure is created by either **CvArgsMake** or **CvArgsMakeIds**. If you want all attributes to be retrieved, pass in a null pointer for this argument.

Notes

You can issue the function on either:

Network level — The function retrieves a list of objects on a network, including all subnets. To issue the function on a network level, specify an IP address, such as 128.100.0.0.

Subnet level — The function retrieves a list of objects on a particular subnet. To issue the function on a subnet level, specify an IP address with a subnet number, such as 128.100.111.0.

Table 2-2 lists the valid parent and child object types you can specify with this function.

Table 2-2. Valid Parent and Child Object Types

Parent Object Type	Child Object Types
Network	Circuit Customer NetCac PMPCktLeaf PMPCktRoot PMPSpvcRoot SmdsAddressPrefix SmdsCountryCode SmdsLocalIndividualAddress SmdsNetwideGroupAddress SvcCUG SvcCUGMbrRule SvcSecScn Switch TrafficDesc ServiceName VPN
Customer	LPort Circuit
VPN	LPort Circuit
SvcCUGMbrRule	SvcCUGMbr
SvcCUG	SvcCUGMbr

Table 2-2. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
Switch	Aps Card Channel Circuit LPort Performance Monitor PFdl PMPSpvcRoot PPort Spvc SmdsAddressPrefix SmdsAlienGroupAddress SmdsAlienIndividualAddress SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress SmdsSwitchGroupAddress SvcAddress SvcConfig SvcNodePrefix SvcPrefix SvcUserPart

Table 2-2. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
Card	Aps Channel Circuit LPort Performance Monitor PFdl PMPSpvcRoot PPort Spvc SmdsAlienGroupAddress SmdsAlienIndividualAddress SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress SmdsSwitchGroupAddress SvcConfig SvcNodePrefix SvcUserPart
PPort	Aps Channel Circuit LPort PerformanceMonitor PFdl (8-port ATM T1 card only)
Channel (valid for channelized DS3 card only)	LPort

Table 2-2. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
LPort	AssignedSvcSecScn Circuit PMPCktRoot PMPSpvcRoot SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress Spvc SvcAddress SvcConfig SvcPrefix SvcUserPart SvcSecScnActParam
SmdsIndividualScreen	SmdsLocalIndividualAddress SmdsAlienIndividualAddress
SmdsGroupScreen	SmdsSwitchGroupAddress SmdsAlienGroupAddress
SmdsNetwideGroupAddress	SmdsSwitchGroupAddress SmdsLocalIndividualAddress
PMPCktRoot	PMPCktLeaf
PMPSpvcRoot	PMPSpvcLeaf

Prerequisites

You must establish a session with the Provisioning Server with **CvConnect**.

Make sure that the object ID is valid.

To specify which attributes you want retrieved for the object, you must build the argument list. To do so, either:

- Issue a single function (**CvArgsMakeIds**) that takes a variable number of arguments and builds the CvArgs structure.

- Issue a series of utility functions that create (**CvArgsMake**) and fill in (**CvArgsSet***AttrType*) the CvArgs structure.

Post-processing

When you are finished with the function's returned value, you must issue **CvArgsFree** to free the memory used by the argument list.

The application must return to the **select** loop. Invoke **CvProcessEvents** after activity is found on the file descriptor.

This function makes multiple callbacks to the callback handler, each representing an object contained by the parent:

- In the callback function, the *ObjId* parameter represents the listed object. The *Args* parameter represents the object's attributes.
- The last call to the callback handler returns the status **cveListDone**. The call does not include a valid object.
- A call to the callback handler that returns a status other than **cveNoError** or **cveListDone** indicates that an error has occurred. No more responses will be made for this request.

When issuing multiple requests, use the returned **CvReqId** to verify or look up the original request. Compare this **CvReqId** to the **CvReqId** of the callback function to associate a response with its original request.

See Also

CvaListAllContainedObjects, **CvsListAllContainedObjects**, and **CvsListContainedObjects**.

Purpose

Modifies specific attributes of an object in the database and (optionally) in the switch.

Syntax

CvReqId CvaModifyObject(
CvServerHandle *Handle*,
**const CvObjectId ***_ObjId_,
CVCBH *Callback*,
**void ***_UserArg_,
CvConstArgs *Args* **);**

Description

CvaModifyObject is an asynchronous function that modifies specific attributes of an object.

Handle is the CvServerHandle variable that identifies this session with the Provisioning Server. This handle is returned by **CvConnect**.

ObjId is a pointer to the CvObjectId structure that specifies the object to be modified.

Callback is the callback function.

UserArg is a user-context argument. Use this argument to pass any value to the callback handler.

Args is a constant CvArgs structure that specifies the attributes to be modified. The API does not modify or delete this argument. This structure is created by either **CvArgsMake** or **CvArgsMakeVals**. Specify only those attributes appropriate for the object type (refer to the *NavisXtend Provisioning Server Object Attribute Definitions*). You can modify any attributes except those with either the Read-Only or Create-Only access restriction.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

Use this function to create a card or PPort. The CascadeView database automatically populates each switch with cards of type "empty." Use this function to modify the card's type from "empty" to a specified type. Likewise, once a card has been configured, CascadeView automatically populates the card with all necessary physical ports. Use this function to change the PPort specifications. In the case of the channelized DS3 card, once the card has been configured, CascadeView automatically populates the card with all necessary channels. Use this function to change the channel specifications.

Use this function to remove a card. To do so, modify the card type to "empty."

Prerequisites

You must establish a session with the Provisioning Server with **CvConnect**.

Make sure that the object ID is valid.

You must build the argument list. To do so, either:

- Issue a single function (**CvArgsMakeVals**) that takes a variable number of arguments and builds the CvArgs structure.
- Issue a series of utility functions that create (**CvArgsMake**) and fill in (**CvArgsSet***AttrType*) the CvArgs structure.

Post-processing

When you are finished with the function's returned value, you must issue **CvArgsFree** to free the memory used by the argument list.

The application must return to the **select** loop. Invoke **CvProcessEvents** after activity is found on the file descriptor.

When issuing multiple requests, use the returned **CvReqId** to verify or look up the original request. Compare this **CvReqId** to the **CvReqId** of the callback function to associate a response with its original request.

See Also

CvsModifyObject.

CvGetErrorMsg

Purpose

Obtains the help string associated with a function's error status code.

Syntax

const char *CvGetErrorMsg(
CvServerHandle *Handle* **);**

Description

CvGetErrorMsg obtains the help string associated with a function's error status code. When an operational function returns an error status code, issue this function to obtain a help string that provides details on the error.

Handle is the CvServerHandle variable that identifies this session with the Provisioning Server. This handle is returned by **CvConnect**.

Notes

This function returns a null pointer if the last function issued did not return an error.

Prerequisites

You must establish a session with the Provisioning Server with **CvConnect**.

Use this function in conjunction with another function that returns an error status:

1. Issue the function (such as **CvaAddObject**).
2. If the function returns an error status, issue **CvGetErrorMsg** to obtain the help string associated with the status code.

Post-processing

None.

See Also

CvArgsErrorIndex.

Purpose

Reads the argument list returned by a synchronous function.

Syntax

CvConstArgs CvGetResponseArgs(CvServerHandle *Handle* **);**

Description

CvGetResponseArgs reads the argument list returned by a synchronous function. Use this function to determine if you omitted any required parameters when you issued the synchronous operational function. The returned argument list indicates which attributes have values and which do not. Any attribute that is missing a value is a required attribute that you omitted.

Handle is the CvServerHandle variable that identifies this session with the Provisioning Server. This handle is returned by **CvConnect**.

Prerequisites

You must establish a session with the Provisioning Server with **CvConnect**.

Use this function in conjunction with a synchronous operational function to determine if a required attribute was omitted:

1. Issue the function (such as **CvsAddObject**).
2. If the response has the status **cveArgumentNotPresent**, issue **CvGetResponseArgs** to obtain the argument list returned by the function. Any attribute that is missing a value is a required attribute that you omitted.

Post-processing

None.

See Also

CvArgsErrorIndex.

CvsAddMember

Purpose

Adds a member to a list of objects.

Syntax

void CvsAddMember(
CvServerHandle *Handle*,
const CvObjectId * *Container*,
const CvObjectId * *Member*,
CvStatus * *Status* **);**

Description

CvsAddMember is a synchronous function that adds an address to a screen or netwide group address. Upon completion of the function, the address represented by the second object ID argument is added to the object specified by the first object ID argument.

Handle is the CvServerHandle variable that identifies this session with the Provisioning Server. This handle is returned by **CvConnect**.

Container is a pointer to the CvObjectId structure that specifies the container object, which is the screen or netwide group address.

Member is a pointer to the CvObjectId structure that specifies the member object, which is the address to become a member of the screen or netwide group address.

Status, an output parameter, is a pointer to the CvStatus variable that contains the status returned by the function. Initialize this argument to **cveNoError**.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

When you specify the object CVT_SmdsGroupScreen as the container object, the member to be added must be either a CVT_SmdsAlienGroupAddress or a CVT_SmdsSwitchGroupAddress.

When you specify the object CVT_SmdsIndividualScreen as the container object, the member to be added must be either a CVT_SmdsLocalIndividualAddress or a CVT_SmdsAlienIndividualAddress.

When you specify the object CVT_SmdsNetwideGroupAddress as the container object, the member to be added must be a CVT_SmdsLocalIndividualAddress.

Prerequisites

You must establish a session with the Provisioning Server with **CvConnect**.

You must initialize the *Status* argument to **cveNoError**.

Make sure that the object IDs represented by *Container* and *Member* are valid.

Post-processing

None.

See Also

CvaAddMember.

CvsAddObject

Purpose

Creates an object in the database and (optionally) in the switch.

Syntax

void CvsAddObject(
CvServerHandle *Handle*,
**const CvObjectId ***_ObjId_,
**CvStatus ***_Status_,
CvConstArgs *Args* **);**

Description

CvsAddObject is a synchronous function that creates an object in the database and (optionally) in the switch. The attributes specified by the function are used to initialize the object.

Handle is the CvServerHandle variable that identifies this session with the Provisioning Server. This handle is returned by **CvConnect**.

ObjId is a pointer to the CvObjectId structure that specifies the object to be added.

Status, an output parameter, is a pointer to the CvStatus variable that contains the status returned by the function. Initialize this argument to **cveNoError**.

Args is a constant CvArgs structure that specifies the attributes for the object to be added. The API does not modify or delete this argument. This structure is created by either **CvArgsMake** or **CvArgsMakeVals**. Specify only those attributes appropriate for the object type (refer to the *NavisXtend Provisioning Server Object Attribute Definitions*). You can specify any attributes except those with the Read-Only access restriction.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

To create a card or PPort, use a Modify function (**CvaModifyObject** or **CvsModifyObject**). The CascadeView database automatically populates each switch with cards of type “empty.” Use the Modify function to modify the card’s type from “empty” to a specified type. Likewise, once a card has been configured, CascadeView automatically populates the card with all necessary physical ports. Use the Modify function to change the PPort specifications. In the case of the channelized DS3 card, once the card has been configured, CascadeView automatically populates the card with all necessary channels. Use the Modify function to change the channel specifications.

If you omit a required parameter, the function response will have the status **cveArgumentNotPresent**. Issue **CvGetResponseArgs** to obtain the argument list returned by the function. Any attribute that is missing a valid value is a required attribute that you omitted.

Prerequisites

You must establish a session with the Provisioning Server with **CvConnect**.

You must initialize the *Status* argument to **cveNoError**.

Make sure that the object ID is valid.

You must build the argument list. To do so, either:

- Issue a single function (**CvArgsMakeVals**) that takes a variable number of arguments and builds the CvArgs structure.
- Issue a series of utility functions that create (**CvArgsMake**) and fill in (**CvArgsSet***AttrType*) the CvArgs structure.

Post-processing

When you are finished with the function’s returned value, you must issue **CvArgsFree** to free the memory used by the argument list.

If the response has the status **cveArgumentNotPresent**, issue **CvGetResponseArgs** to obtain the argument list returned by the function. Any attribute that is missing a valid value is a required attribute that you omitted.

See Also

CvaAddObject, **CvaModifyObject**, and **CvsModifyObject**.

Purpose

Removes a member from a list of objects.

Syntax

```
void CvsDeleteMember(
        CvServerHandle Handle,
        const CvObjectId *Container,
        const CvObjectId *Member,
        CvStatus *Status );
```

Description

CvsDeleteMember is a synchronous function that removes an address from a screen or group address. Upon completion of the function, the address represented by the second object ID argument is removed from the object specified by the first object ID argument.

Handle is the CvServerHandle variable that identifies this session with the Provisioning Server. This handle is returned by **CvConnect**.

Container is a pointer to the CvObjectId structure that specifies the container object, which is the screen or netwide group address.

Member is a pointer to the CvObjectId structure that specifies the address to be removed.

Status, an output parameter, is a pointer to the CvStatus variable that contains the status returned by the function. Initialize this argument to **cveNoError**.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

When you specify the object CVT_SmdsGroupScreen as the container object, the member to be removed must be either a CVT_SmdsAlienGroupAddress or a CVT_SmdsSwitchGroupAddress.

When you specify the object CVT_SmdsIndividualScreen as the container object, the member to be removed must be either a CVT_SmdsLocalIndividualAddress or a CVT_SmdsAlienIndividualAddress.

When you specify the object CVT_SmdsNetwideGroupAddress as the container object, the member to be removed must be a CVT_SmdsLocalIndividualAddress.

Prerequisites

You must establish a session with the Provisioning Server with **CvConnect**.

You must initialize the *Status* argument to **cveNoError**.

Make sure that the object IDs represented by *Container* and *Member* are valid.

Post-processing

None.

See Also

CvaDeleteMember.

Purpose

Deletes an object in the database and (optionally) in the switch.

Syntax

void CvsDeleteObject(
CvServerHandle *Handle*,
const CvObjectId ***ObjId*,
CvStatus ***Status* **);

Description

CvsDeleteObject is a synchronous function that deletes an object from the database and (optionally) in the switch.

Handle is the CvServerHandle variable that identifies this session with the Provisioning Server. This handle is returned by **CvConnect**.

ObjId is a pointer to the CvObjectId structure that specifies the object to be added.

Status, an output parameter, is a pointer to the CvStatus variable that contains the status returned by the function. Initialize this argument to **cveNoError**.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

You only need to delete an SMDS switch group address if the database shows an SMDS switch group address that should not exist.

To remove a card, use a Modify function (**CvaModifyObject** or **CvsModifyObject**) to modify the card's type to "empty."

You cannot delete some objects until the objects they contain have been deleted. For example, you cannot delete an LPort until all circuits and addresses defined on that LPort have been deleted.

Prerequisites

You must establish a session with the Provisioning Server with **CvConnect**.

You must initialize the *Status* argument to **cveNoError**.

Make sure that the object ID is valid.

Post-processing

None.

See Also

CvaDeleteObject, **CvaModifyObject**, and **CvsModifyObject**.

Purpose

Retrieves the values of specific attributes.

Syntax

CvArgs CvsGetObject(
CvServerHandle *Handle*,
**const CvObjectId ***_ObjId_,
**CvStatus ***_Status_,
CvConstArgs *Args* **);**

Description

CvsGetObject is a synchronous function that retrieves the values of specific attributes.

Handle is the CvServerHandle variable that identifies this session with the Provisioning Server. This handle is returned by **CvConnect**.

ObjId is a pointer to the CvObjectId structure that specifies the object whose attributes are to be retrieved.

Status, an output parameter, is a pointer to the CvStatus variable that contains the status returned by the function. Initialize this argument to **cveNoError**.

Args is a constant CvArgs structure that specifies the attributes to be retrieved. The API does not modify or delete this argument. This structure is created by either **CvArgsMake** or **CvArgsMakeIds**. Specify only those attributes appropriate for the object type (refer to the *NavisXtend Provisioning Server Object Attribute Definitions*). Specify only attribute IDs with null values.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

Prerequisites

You must establish a session with the Provisioning Server with **CvConnect**.

You must initialize the *Status* argument to **cveNoError**.

Make sure that the object ID is valid.

You must build the argument list. To do so, either:

- Issue a single function (**CvArgsMakeVals**) that takes a variable number of arguments and builds the CvArgs structure.
- Issue a series of utility functions that create (**CvArgsMake**) and fill in (**CvArgsSet***AttrType*) the CvArgs structure.

Post-processing

When you are finished with the function's returned value, you must issue **CvArgsFree** to free the memory used by the returned argument list.

See Also

CvaGetObject.

Purpose

Retrieves a list of objects of any type that are immediate children of a specified object.

Syntax

CvListHandle CvsListAllContainedObjects(
CvServerHandle *Handle*,
const CvObjectId **ObjId*,
CvStatus ***Status*,
CvConstArgs *Args* **);**

Description

CvsListAllContainedObjects is a synchronous function that retrieves a list of objects of any type that are immediate children of a specified object.

Handle is the CvServerHandle variable that identifies this session with the Provisioning Server. This handle is returned by **CvConnect**.

ObjId is a pointer to the CvObjectId structure that specifies the parent object. The parent object is always the immediate parent of the contained objects (such as a PPort that is a parent of multiple LPorts).

Status, an output parameter, is a pointer to the CvStatus variable that contains the status returned by the function. Initialize this argument to **cveNoError**.

Args is a constant CvArgs structure that specifies the attributes to be retrieved. The API does not modify or delete this argument. This structure is created by **CvArgsMake**. If you want all attributes to be retrieved, pass in a null pointer for this argument.

Notes

You can issue the function on either:

Network level — The function retrieves a list of objects on a network, including all subnets. To issue the function on a network level, specify an IP address, such as 128.100.0.0.

Subnet level — The function retrieves a list of objects on a particular subnet. To issue the function on a subnet level, specify an IP address with a subnet number, such as 128.100.111.0.

Table 2-3 lists the valid parent and child object types you can specify with this function.

.

Table 2-3. Valid Parent and Child Object Types

Parent Object Type	Child Object Types
Network	Switch VPN Customer SmdsCountryCode SmdsNetwideGroupAddress SvcSecScn SvcCUG SvcCUGMbrRule TrafficDesc NetCac ServiceName
Customer	LPort Circuit
VPN	LPort Circuit
SvcCUGMbrRule	SvcCUGMbr
SvcCUG	SvcCUGMbr

Table 2-3. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
Switch	Card SvcNodePrefix SmdsSwitchGroupAddress SmdsAddressPrefix SmdsAlienIndividualAddress SmdsAlienGroupAddress
Card	PPort
PPort	LPort Channel PFdl (8-port ATM T1 card only) PerformanceMonitor Aps (1-port OC-12c/STM-4 card only)
Channel (valid for channelized DS3 card only)	LPort
LPort	Circuit SvcPrefix SvcAddress SvcUserPart SvcConfig SmdsLocalIndividualAddress SmdsIndividualScreen SmdsGroupScreen PMPCktRoot AssignedSvcSecScn SvcSecScnActParam Spvc PMPSpvcRoot
SmdsIndividualScreen	SmdsLocalIndividualAddress SmdsAlienIndividualAddress
SmdsGroupScreen	SmdsSwitchGroupAddress SmdsAlienGroupAddress
SmdsNetwideGroupAddress	SmdsSwitchGroupAddress SmdsLocalIndividualAddress
PMPCktRoot	PMPCktLeaf

Table 2-3. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
PMPSpvcRoot	PMPSpvcLeaf
ServiceName	Circuit

Prerequisites

You must establish a session with the Provisioning Server with **CvConnect**.

You must initialize the *Status* argument to **cveNoError**.

Make sure that the object ID is valid.

You must build the argument list. To do so, either:

- Issue a single function (**CvArgsMakeVals**) that takes a variable number of arguments and builds the CvArgs structure.
- Issue a series of utility functions that create (**CvArgsMake**) and fill in (**CvArgsSet***AttrType*) the CvArgs structure.

Post-processing

When you are finished with the function's returned value, you must issue **CvArgsFree** to free the memory used by the argument list.

Once you receive a CvListHandle, issue **CvsNextObject** to retrieve the next object. Pass in the CvListHandle value returned by the previous **CvsListContainedObjects**:

- When a call to **CvsNextObject** returns the status **cveNoError**, continue issuing **CvsNextObject**.
- When a call to **CvsNextObject** returns the status **cveListDone**, stop issuing **CvsNextObject**. This last call does not include a valid object.

- When a call to **CvsNextObject** returns a status other than **cveNoError** or **cveListDone**, an error has occurred.

Keep in mind that the CvListHandle has a limited lifetime. If more than 10 minutes elapse between requests with the same handle, the next request may return with an error status.

See Also

CvaListAllContainedObjects, **CvaListContainedObjects**, and **CvsListContainedObjects**.

CvsListContainedObjects

Purpose

Retrieves a list of objects of a specified object type that are contained by another object.

Syntax

CvListHandle CvsListContainedObjects(
CvServerHandle *Handle*,
const CvObjectId **ObjId*,
CvObjectType *ObjType*,
CvStatus ***Status*,
CvConstArgs *Args* **);**

Description

CvsListContainedObjects is a synchronous function that retrieves a list of objects contained by another object. The returned objects are those that match a specified object type.

Handle is the CvServerHandle variable that identifies this session with the Provisioning Server. This handle is returned by **CvConnect**.

ObjId is a pointer to the CvObjectId structure that specifies the parent object. The parent object can be the immediate parent of the contained objects (such as a PPort that is a parent of multiple LPorts) or the parent object can be positioned higher in the containment hierarchy (such as a switch that is a parent of multiple LPorts).

ObjType is the enumerated value that specifies the type of the objects to be retrieved.

Status, an output parameter, is a pointer to the CvStatus variable that contains the status returned by the function. Initialize this argument to **cveNoError**.

Args is a constant CvArgs structure that specifies the attributes to be retrieved. The API does not modify or delete this argument. This structure is created by either **CvArgsMake** or **CvArgsMakeIds**. If you want all attributes to be retrieved, pass in a null pointer for this argument.

Notes

You can issue the function on either:

Network level — The function retrieves a list of objects on a network, including all subnets. To issue the function on a network level, specify an IP address, such as 128.100.0.0.

Subnet level — The function retrieves a list of objects on a particular subnet. To issue the function on a subnet level, specify an IP address with a subnet number, such as 128.100.111.0.

Table 2-4 lists the valid parent and child object types you can specify with this function.

Table 2-4. Valid Parent and Child Object Types

Parent Object Type	Child Object Types
Network	Circuit Customer NetCac PMPCktLeaf PMPCktRoot PMPSpvcRoot SmdsAddressPrefix SmdsCountryCode SmdsLocalIndividualAddress SmdsNetwideGroupAddress SvcCUG SvcCUGMbrRule SvcSecScn Switch TrafficDesc ServiceName VPN
Customer	LPort Circuit
VPN	LPort Circuit

Table 2-4. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
SvcCUGMbrRule	SvcCUGMbr
SvcCUG	SvcCUGMbr
Switch	Aps Card Channel Circuit LPort Performance Monitor PFdl PMPSpvcRoot PPort Spvc SmdsAddressPrefix SmdsAlienGroupAddress SmdsAlienIndividualAddress SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress SmdsSwitchGroupAddress SvcAddress SvcConfig SvcNodePrefix SvcPrefix SvcUserPart

Table 2-4. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
Card	Aps Channel Circuit LPort Performance Monitor PFdl PMPSpvcRoot PPort Spvc SmdsAlienGroupAddress SmdsAlienIndividualAddress SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress SmdsSwitchGroupAddress SvcConfig SvcNodePrefix SvcUserPart
PPort	Aps Channel Circuit LPort PerformanceMonitor PFdl (8-port ATM T1 card only)
Channel (valid for channelized DS3 card only)	LPort

Table 2-4. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
LPort	AssignedSvcSecScn Circuit PMPCktRoot PMPSpvcRoot SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress Spvc SvcAddress SvcConfig SvcPrefix SvcUserPart SvcSecScnActParam
SmdsIndividualScreen	SmdsLocalIndividualAddress SmdsAlienIndividualAddress
SmdsGroupScreen	SmdsSwitchGroupAddress SmdsAlienGroupAddress
SmdsNetwideGroupAddress	SmdsSwitchGroupAddress SmdsLocalIndividualAddress
PMPCktRoot	PMPCktLeaf
PMPSpvcRoot	PMPSpvcLeaf

Prerequisites

You must establish a session with the Provisioning Server with **CvConnect**.

You must initialize the *Status* argument to **cveNoError**.

Make sure that the object ID is valid.

To specify which attributes you want retrieved for the object, you must build the argument list. To do so, either:

- Issue a single function (**CvArgsMakeIds**) that takes a variable number of arguments and builds the CvArgs structure.

- Issue a series of utility functions that create (**CvArgsMake**) and fill in (**CvArgsSet***AttrType*) the CvArgs structure.

Post-processing

When you are finished with the function's returned value, you must issue **CvArgsFree** to free the memory used by the argument list.

Once you receive a CvListHandle, issue **CvsNextObject** to retrieve the next object. Pass in the list handle value returned by the previous **CvsListContainedObjects**:

- When a call to **CvsNextObject** returns the status **cveNoError**, continue issuing **CvsNextObject**.
- When a call to **CvsNextObject** returns the status **cveListDone**, stop issuing **CvsNextObject**. This last call does not include a valid object.
- When a call to **CvsNextObject** returns a status other than **cveNoError** or **cveListDone**, an error has occurred.

Keep in mind that the CvListHandle has a limited lifetime. If more than 10 minutes elapse between requests with the same handle, the next request may return with an error status.

See Also

CvaListAllContainedObjects, **CvaListContainedObjects**, and **CvsListAllContainedObjects**.

CvsModifyObject

Purpose

Modifies specific attributes of an object in the database and (optionally) in the switch.

Syntax

void CvsModifyObject(
CvServerHandle *Handle*,
**const CvObjectId ***_ObjId_,
**CvStatus ***_Status_,
CvArgs *Args* **);**

Description

CvsModifyObject is a synchronous function that modifies specific attributes of an object.

Handle is the CvServerHandle variable that identifies this session with the Provisioning Server. This handle is returned by **CvConnect**.

ObjId is a pointer to the CvObjectId structure that specifies the object to be modified.

Status, an output parameter, is a pointer to the CvStatus variable that contains the status returned by the function. Initialize this argument to **cveNoError**.

Args is the CvArgs structure that specifies the attributes for the object to be added. This structure is created by either **CvArgsMake** or **CvArgsMakeVals**. Specify only those attributes appropriate for the object type (refer to the *NavisXtend Provisioning Server Object Attribute Definitions*). You can modify any attributes except those with either the Read-Only or Create-Only access restriction.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

Use this function to create a card or PPort. The CascadeView database automatically populates each switch with cards of type “empty.” Use this function to modify the card’s type from “empty” to a specified type. Likewise, once a card has been configured, CascadeView automatically populates the card with all necessary physical ports. Use this function to change the PPort specifications. In the case of the channelized DS3 card, once the card has been configured, CascadeView automatically populates the card with all necessary channels. Use this function to change the channel specifications.

Use this function to remove a card. To do so, modify the card type to “empty.”

Prerequisites

You must establish a session with the Provisioning Server with **CvConnect**.

You must initialize the *Status* argument to **cveNoError**.

Make sure that the object ID is valid.

You must build the argument list. To do so, either:

- Issue a single function (**CvArgsMakeVals**) that takes a variable number of arguments and builds the CvArgs structure.
- Issue a series of utility functions that create (**CvArgsMake**) and fill in (**CvArgsSet***AttrType*) the CvArgs structure.

Post-processing

When you are finished with the function’s returned value, you must issue **CvArgsFree** to free the memory used by the argument list.

See Also

CvaModifyObject.

CvsNextObject

Purpose

Retrieves the next object in a list.

Syntax

CvArgs CvsNextObject(
CvServerHandle *Handle***,**
CvListHandle *ListHandle***,**
CvStatus ***Status **);

Description

CvsNextObject is a synchronous function that retrieves the next object in a list. When you request a list of objects contained by another object, use this function to retrieve the next object in the list.

Handle is the CvServerHandle variable that identifies this session with the Provisioning Server. This handle is returned by **CvConnect**.

ListHandle is the CvListHandle variable returned by a synchronous List Contained function (**CvsListAllContainedObjects** or **CvsListContainedObjects**).

Status, an output parameter, is a pointer to the CvStatus variable that contains the status returned by the function. Initialize this argument to **cveNoError**.

Prerequisites

You must establish a session with the Provisioning Server with **CvConnect**.

You must initialize the *Status* argument to **cveNoError**.

Use a List Contained function (**CvsListAllContainedObjects** or **CvsListContainedObjects**) to obtain a CvListHandle.

Post-processing

When you are finished with the function's returned value, you must issue **CvArgsFree** to free the memory used by the returned argument list.

Once you receive a CvListHandle, issue **CvsNextObject** to retrieve the next object:

- When a call to **CvsNextObject** returns the status **cveNoError**, continue issuing **CvsNextObject**.
- When a call to **CvsNextObject** returns the status **cveListDone**, stop issuing **CvsNextObject**. This last call does not include a valid object.
- When a call to **CvsNextObject** returns a status other than **cveNoError** or **cveListDone**, an error has occurred.

Keep in mind that the CvListHandle has a limited lifetime. If more than 10 minutes elapse between requests with the same handle, the next request may return with an error status.

See Also

CvsListAllContainedObjects and **CvsListContainedObjects**.

Select Loop Processing Functions

CVCBH (Client Callback Handler)..... 2-65
CvGetSelectInfo 2-67
CvProcessEvents 2-69
CvTimeout 2-71

Purpose

Is the prototype for a function supplied by the client. The API calls this callback handler to deliver a response to an asynchronous request.

Syntax

void (*CVCBH) (
CvObjectId *ObjId*,
CvStatus *Status*,
CvReqId *RequestId*,
void * *UserArg*,
CvConstArgs *Args* **);**

Description

CVCBH is the prototype for a function supplied by the client. The API calls this callback handler to deliver a response to an asynchronous request.

ObjId is a pointer to the CvObjectId structure that specifies the object for the response.

Status contains the status returned by the function.

RequestId is the request ID associated with the asynchronous request.

UserArg is the user-context argument associated with the asynchronous request.

Args is a constant CvArgs structure that contains the attributes for the response. The client must not modify or delete this argument.

Prerequisites

None.

Post-processing

Upon return, the function returns back into the API library and then into the **select** loop.

The application must return to the **select** loop. Invoke **CvProcessEvents** after activity is found on the file descriptor.

Purpose

Obtains information needed to pass to a **select** system call after the completion of an asynchronous function.

Syntax

void CvGetSelectInfo(
int *WaitTimeMs*,
**int ***Numfds*,
fd_set **Fdset*,
struct timeval **Timeout* **);**

Description

CvGetSelectInfo obtains information needed to pass to a **select** system call after the completion of an asynchronous function. You should issue this call before each call to **select**. The function provides information that **select** uses to notify your application of pending messages from the Provisioning Server.

WaitTimeMs is the maximum amount of time (in milliseconds) that **select** should wait for an event. Use this timeout argument as follows:

0 – For applications designed to perform polling in the **select** loop.

> 0 – For applications designed to time out. This value is useful for complex applications that use other kinds of timeouts. This timeout represents the time to wait for an event, not the time to wait for a specific response.

Numfd and *Fdset* indicate the file descriptors used by the application. *Numfd* is the number of file descriptors indicated by the fd_set structure. *Fdset* is a pointer to the fd_set structure that contains the set of file descriptors to be passed to **select**. Either initialize this argument as empty or set to a value to indicate the application's own file descriptors.

Timeout, an output parameter, is a pointer to the timeval structure. When the function returns, timeval is set to either of the following values, whichever is smaller:

- *WaitTimeMs*
- Preferred timeout of the API

Notes

This function augments the *Numfd* and *Fdset* arguments with its own file descriptor information and sets the timeval structure to the smallest requested timeout. Your application should pass this information to the **select** call.

Prerequisites

None.

Post-processing

Use this function in conjunction with a **select** loop:

1. Issue this function to obtain information for the **select** call.
2. If **select** returns 0, issue **CvTimeout** to determine whether a timeout occurred from the server.
3. When **select** returns a non-zero value, issue **CvProcessEvents** to receive and process a message from the server.

See Also

CvProcessEvents and **CvTimeout**.

Purpose

Processes activity on file descriptors in order to receive responses to an asynchronous function.

Syntax

int CvProcessEvents(
long *TimeOutInMs***,**
int *ReturnOnError* **);**

Description

CvProcessEvents processes activity on file descriptors in order to receive responses to an asynchronous function. The function returns the value from the last **select** call.

TimeOutInMs is the maximum amount of time (in milliseconds) that is used to control how **CvProcessEvents** times out. Use this timeout argument as follows:

< 0 (Blocking) — Signals **CvProcessEvents** to never time out. Set this value if you want **CvProcessEvents** to block until a message arrives.

0 (Polling) — Signals **CvProcessEvents** to process a single message with no blocking. The function returns immediately if no events are pending. Set this value if you want **CvProcessEvents** to poll for an event. This value is useful for applications that use their own **select** loop and already know that an event is waiting.

> 0 (Blocking with timeout) — Signals **processEvents** to time out. Set this value if you want **CvProcessEvents** to return when either a single event occurs or the timeout expires. This value is useful for complex applications that use other kinds of timeouts. You can set this value higher or lower than the timeout that the client stack uses to wait for responses from the server.

ReturnOnError signals **processEvents** to return when an error return occurs in the **select** loop (such as when a signal is caught). Use this argument when *TimeOutInMs* is a non-zero value. Set to 0 if you do not want to be notified when an error occurs. Set to 1 to be notified when an error occurs.

Notes

The return value from this function is the same as that from the last **select** call that it makes. The value **errno** is untouched for examination by the caller.

Prerequisites

Use this function in conjunction with a **select** loop:

1. Before you issue this function, issue **CvGetSelectInfo** to obtain information needed to pass to a **select** system call.
2. Issue the **select** call.
3. If **select** returns 0, issue **CvTimeout** to determine whether a timeout occurred from the server.
4. When **select** returns a non-zero value, issue this function to receive and process a message from the server.

Post-processing

None.

See Also

CvGetSelectInfo and **CvTimeout**.

Purpose Determines if a timeout occurred for an outstanding asynchronous request.

Syntax **void CvTimeout( )**

Description **CvTimeout** determines if a timeout occurred for an outstanding asynchronous request. The function calls the client's callback handler appropriately if any timeouts occurred.

Notes Issue this call when **select** returns 0.

Prerequisites None.

Post-processing None.

See Also **CvProcessEvents**.

Utility Functions

CvAddArgumentByName 2-75
CvAddArgumentByNameValue 2-76
CvArgsCombine 2-77
CvArgsCount 2-78
CvArgsEqual 2-79
CvArgsErrorIndex 2-80
CvArgsFree 2-81
CvArgsGetInt 2-82
CvArgsGetIpAddr 2-83
CvArgsGetObjectId 2-84
CvArgsGetStatus 2-85
CvArgsGetStr 2-86
CvArgsIdAt 2-87
CvArgsMake 2-88
CvaArgsMakeIds 2-89
CvaArgsMakeVals 2-90
CvArgsPrint 2-92
CvArgsRemove 2-93
CvArgsSelect 2-94
CvArgsSetInt 2-95
CvArgsSetIpAddr 2-96
CvArgsSetNull 2-97
CvArgsSetObjectId 2-98
CvArgsSetStr 2-99
CvArgsStatusAt 2-100
CvArgsToString 2-101
CvArgsTypeAt 2-102
CvCreateApsId 2-103
CvCreateAssignedSvcSecScnId 2-104
CvCreateAtmEndpointId 2-106
CvCreateAtmLPortId 2-108
CvCreateCardId 2-110
CvCreateChannelId 2-111
CvCreateCustomerId 2-112
CvCreateDlciId 2-113
CvCreateFdlId 2-115
CvCreateLPortId 2-116
CvCreateNetCacId 2-118
CvCreateNetworkId 2-119

CvCreateNetworkIdFromString 2-120
CvCreateNullId 2-122
CvCreatePerformanceMonitorId 2-123
CvCreatePMPCktLeafId 2-124
CvCreatePMPCktRootId 2-126
CvCreatePMPSpvcLeafId 2-128
CvCreatePMPSpvcRootId 2-129
CvCreatePPortId 2-131
CvCreateServiceNameId 2-132
CvCreateSmdsAlienGroupAddrId 2-133
CvCreateSmdsAlienIndivAddrId 2-135
CvCreateSmdsCountryCodeId 2-137
CvCreateSmdsGroupAddrId 2-138
CvCreateSmdsGroupScreenId 2-140
CvCreateSmdsIndivScreenId 2-141
CvCreateSmdsLocalIndivAddrId 2-142
CvCreateSmdsNetwideGroupAddrId 2-144
CvCreateSmdsPrefixId 2-146
CvCreateSmdsSsiIndivAddrId 2-148
CvCreateSpvcId 2-150
CvCreateSvcAddressId 2-152
CvCreateSvcConfigId 2-154
CvCreateSvcCUGId 2-155
CvCreateSvcCUGMbrId 2-156
CvCreateSvcCUGMbrRuleId 2-158
CvCreateSvcNodePrefixId 2-159
CvCreateSvcPrefixId 2-161
CvCreateSvcSecScnId 2-163
CvCreateSvcSecScnActParamId 2-164
CvCreateSvcUserPartId 2-165
CvCreateSwitchId 2-167
CvCreateSwitchIdFromString 2-168
CvCreateTrafficDescId 2-169
CvCreateTrafficShaperId 2-170
CvCreateVirtualUniLPortIId 2-172
CvCreateVPNId 2-174
CvGetAddress 2-175
CvGetArgumentName 2-176
CvGetCardNumber 2-177
CvGetChannelNumber 2-178
CvGetDlciNumber 2-179
CvGetEnumName 2-180

CvGetLPortNumber 2-181
CvGetName 2-182
CvGetName2 2-183
CvGetNetworkNumber 2-184
CvGetObjectType 2-185
CvGetObjectTypeName 2-186
CvGetPMPSpvcLeafId 2-187
CvGetPPortNumber 2-188
CvGetPrefix 2-189
CvGetStartVpi 2-190
CvGetSvcAddress 2-191
CvGetSwitchId 2-192
CvGetTrafficShaperId 2-193
CvGetVciNumber 2-194
CvGetVpiNumber 2-195
CvObjectIdPrint 2-196
CvObjectIdToString 2-197
CvParseObjectId 2-198
CvParseObjectType 2-199
CvStringFree 2-200

Purpose

Adds an argument to an argument list by specifying a textual argument name.

Syntax

void CvAddArgumentByName(
CvArgs *Args*,
CvObjectType *ObjType*,
const char **ArgName* **);**

Description

CvAddArgumentByName adds an argument to an argument list by specifying a textual argument name.

Args is the CvArgs structure to which to add the argument.

ObjType is the enumerated value that specifies the object type.

ArgName is the text argument description of the argument to be read. The function looks up the corresponding argument ID value for that attribute. The argument ID value is defined in the file CvArgId.H.

Prerequisites

Make sure that the argument list represented by *Args* is valid.

Post-processing

None.

See Also

CvAddArgumentByNameValue.

CvAddArgumentByNameValue

Purpose

Adds an argument to an argument list by specifying a textual argument name and a textual argument value.

Syntax

void CvAddArgumentByNameValue(
CvArgs *Args*,
CvObjectType *ObjType*,
const char **ArgName*,
const char **ValString* **);**

Description

CvAddArgumentByNameValue adds an argument to an argument list by specifying a textual argument name and a textual argument value. Use this function to process object descriptions stored in a text file or relational database.

Args is the CvArgs structure to which to add the argument and value.

ObjType is the enumerated value that specifies the object type.

ArgName is the text argument description of the argument to be added. The function looks up the corresponding argument ID value for that attribute. The argument ID value is defined in the file CvArgId.H.

ValString is the text argument description of the value for the argument.

Prerequisites

Use **CvArgsMake** or **CvArgsMakeVals** to create the argument list (CvArgs structure).

Post-processing

None.

See Also

CvAddArgumentByName.

Purpose

Adds two argument lists together.

Syntax

CvArgs CvArgsCombine(
CvArgs *Args1***,**
CvArgs *Args2* **);**

Description

CvArgsCombine adds two argument lists together. The result is a new argument list that combines both sets of arguments. When an argument exists in both argument lists, the value from the second list is used.

Args1 and *Args2* are the CvArgs structures to be added.

Prerequisites

Make sure that the argument lists represented by *Args1* and *Args2* are valid.

Post-processing

When you are finished with the function's returned value, you must issue **CvArgsFree** to free the memory used by the returned argument list.

See Also

CvArgsRemove and **CvArgsSelect**.

CvArgsCount

Purpose

Retrieves the number of arguments in an argument list.

Syntax

int CvArgsCount(
CvConstArgs *Args* **);**

Description

CvArgsCount retrieves the number of arguments in an argument list.

Args is a constant CvArgs structure. The API does not modify or delete this argument.

Prerequisites

Make sure that the argument list represented by *Args* is valid.

Post-processing

None.

See Also

CvArgsIdAt and **CvArgsTypeAt**.

Purpose

Compares two argument lists for equality.

Syntax

int CvArgsEqual(
CvArgs *Args1***,**
CvArgs *Args2* **);**

Description

CvArgsEqual compares two argument lists for equality. Argument lists are equal if their argument numbers, IDs, values, and status are equal. If the argument lists are equal, the function returns a non-zero value.

Args1 and *Args2* are the CvArgs structures to be compared.

Prerequisites

Make sure that the argument lists represented by *Args1* and *Args2* are valid.

Post-processing

None.

CvArgsErrorIndex

Purpose

Determines if any argument in an argument list has an error status.

Syntax

int CvArgsErrorIndex(
CvConstArgs *Args* **);**

Description

CvArgsErrorIndex determines if any argument in an argument list has an error status. If an error status exists, the function returns the one-based index to the first argument with an error.

Args is a constant CvArgs structure. The API does not modify or delete this argument.

Notes

The function's returned value is one-based. The value 0 indicates a non-error status. If you use the returned value in another function (such as **CvArgsIdAt**), subtract 1 from the value.

Prerequisites

Make sure that the argument list represented by *Args* is valid.

Post-processing

To return the error status for the argument, issue **CvArgsStatusAt**.

See Also

CvArgsStatusAt.

Purpose

Deletes a pointer to an argument list.

Syntax

CvArgsFree(
CvArgs *Args* **);**

Description

CvArgsFree deletes an opaque pointer to a CvArgs structure. It frees the memory used by an argument list that was either created explicitly or implicitly by another function.

Args is a pointer to a CvArgs structure. This pointer is no longer valid if **CvArgsFree** is successful.

Prerequisites

None.

Post-processing

None.

See Also

CvArgsMake, **CvArgsMakeIds**, and **CvArgsMakeValues**.

CvArgsGetInt

Purpose

Reads an integer value of a particular attribute out of an argument list.

Syntax

unsigned long CvArgsGetInt(
CvArgs *Args***,**
CvArgId *Id***,**
CvStatus ***Status **);

Description

CvArgsGetInt reads the integer value of a particular attribute out of an argument list.

Args is the CvArgs structure that holds the attribute to be read.

Id is the argument ID symbol for the attribute to be read. The argument ID symbol is defined in the file CvArgId.H.

Status, an output parameter, is a pointer to the CvStatus variable that contains the status returned by the function. Initialize this argument to **cveNoError**. Upon exit, if this argument has any other value, the function's returned value is invalid.

Notes

Use this function to read values returned by queries, such as **CvaGetObject** or **CvaListContainedObjects**.

Prerequisites

You must initialize the *Status* argument to **cveNoError**.

Make sure that the value of the specified attribute is an integer.

Post-processing

None.

See Also

CvArgsGetIpAddr, **CvArgsGetObjectId**, and **CvArgsGetStr**, **CvArgsSetInt**, **CvArgsSetIPAddr**, **CvArgsSetNull**, **CvArgsSetObjectId**, and **CvArgsSetStr**.

Purpose

Reads an IP address value of a particular attribute out of an argument list.

Syntax

CvIpAddr CvArgsGetIpAddr(
CvArgs *Args***,**
CvArgId *Id***,**
CvStatus ***Status **);

Description

CvArgsGetIpAddr reads the IP address value of a particular attribute out of an argument list.

Args is the CvArgs structure that holds the attribute to be read.

Id is the argument ID symbol for the attribute to be read. The argument ID symbol is defined in the file CvArgId.H.

Status, an output parameter, is a pointer to the CvStatus variable that contains the status returned by the function. Initialize this argument to **cveNoError**. Upon exit, if this argument has any other value, the function's returned value is invalid.

Notes

Use this function to read values returned by queries, such as **CvaGetObject** or **CvaListContainedObjects**.

Prerequisites

You must initialize the *Status* argument to **cveNoError**.

Make sure that the value of the specified attribute is an IP address.

Post-processing

None.

See Also

CvArgsGetInt, **CvArgsGetObjectId**, and **CvArgsGetStr**, **CvArgsSetInt**, **CvArgsSetIPAddr**, **CvArgsSetNull**, **CvArgsSetObjectId**, and **CvArgsSetStr**.

CvArgsGetObjectId

Purpose

Reads an object ID value of a particular attribute out of an argument list.

Syntax

CvObjectId CvArgsGetObjectId(
CvArgs *Args***,**
CvArgId *Id***,**
CvStatus ***Status **);

Description

CvArgsGetObjectId reads the object ID value of a particular attribute out of an argument list.

Args is the CvArgs structure that holds the attribute to be read.

Id is the argument ID symbol for the attribute to be read. The argument ID symbol is defined in the file CvArgId.H.

Status, an output parameter, is a pointer to the CvStatus variable that contains the status returned by the function. Initialize this argument to **cveNoError**. Upon exit, if this argument has any other value, the function's returned value is invalid.

Notes

Use this function to read values returned by queries, such as **CvaGetObject** or **CvaListContainedObjects**.

Prerequisites

You must initialize the *Status* argument to **cveNoError**.

Make sure that the value of the specified attribute is an object ID.

Post-processing

None.

See Also

CvArgsGetInt, **CvArgsGetIpAddr**, **CvArgsGetStr**, **CvArgsSetInt**, **CvArgsSetIPAddr**, **CvArgsSetNull**, **CvArgsSetObjectId**, and **CvArgsSetStr**.

Purpose

Retrieves the error status code of an argument in an argument list.

Syntax

CvStatus CvArgsGetStatus(
CvConstArgs *Args*,
CvArgId *Id* **);**

Description

CvArgsGetStatus retrieves the error status code of an argument in an argument list.

Args is a constant CvArgs structure. The API does not modify or delete this argument.

Id is the argument ID symbol for the argument in the argument list. The argument ID symbol is defined in the file CvArgId.H. If *Id* is not in the argument list, the status -1 is returned. Status is defined in the file CvErrors.H.

Prerequisites

Use this function in conjunction with other utility functions that traverse the argument list:

1. Obtain the argument list (CvArgs structure) by appropriate means, such as with **CvArgsMake** or **CvaGetObject**.
2. Issue **CvArgsErrorIndex** to determine if any argument in the argument list has an error status. If you use the returned value in another function, subtract 1 from the value.
3. Issue this function to retrieve the error status code of the argument.

Post-processing

None.

See Also

CvArgsStatusAt.

CvArgsGetStr

Purpose

Reads a string value of a particular attribute out of an argument list.

Syntax

const char *CvArgsGetStr(
CvArgs *Args***,**
CvArgId *Id***,**
CvStatus ***Status* **);

Description

CvArgsGetStr reads the string value of a particular attribute out of an argument list.

Args is the CvArgs structure that specifies the attribute to be read.

Id is the argument ID symbol for the attribute to be read. The argument ID symbol is defined in the file CvArgId.H.

Status, an output parameter, is a pointer to the CvStatus variable that contains the status returned by the function. Initialize this argument to **cveNoError**. Upon exit, if this argument has any other value, the function's returned value is invalid.

Notes

Use this function to read values returned by queries, such as **CvaGetObject** or **CvaListContainedObjects**.

Prerequisites

You must initialize the *Status* argument to **cveNoError**.

Make sure that the value of the specified attribute is a string.

Post-processing

None.

See Also

CvArgsGetInt, **CvArgsGetIpAddr**, **CvArgsGetObjectId**, **CvArgsSetInt**, **CvArgsSetIPAddr**, **CvArgsSetNull**, **CvArgsSetObjectId**, and **CvArgsSetStr**.

Purpose

Retrieves the argument ID at a specified location in an argument list.

Syntax

CvArgId CvArgsIdAt(
CvArgs *Args***,**
int *Index* **);**

Description

CvArgsIdAt retrieves the argument ID at a specified location in an argument list. If the index is invalid, the function returns 0.

Args is a CvArgs structure.

Index is the zero-based index to the argument list.

Notes

Use this function to read values returned by queries, such as **CvaGetObject** or **CvaListContainedObjects**.

Prerequisites

Make sure that the argument list represented by *Args* is valid.

To count the number of arguments in an argument list, issue **CvArgsCount**.

Post-processing

None.

See Also

CvArgsCount, **CvArgsStatusAt**, and **CvArgsTypeAt**.

CvArgsMake

Purpose

Creates an argument list.

Syntax

CvArgs CvArgsMake( )

Description

CvArgsMake creates an argument list. It returns an opaque pointer to a CvArgs structure that specifies the attributes for an object.

Prerequisites

None.

Post-processing

Use this function in conjunction with operational functions that manipulate an object's attributes:

1. Issue this function to create the argument list (CvArgs structure).
2. Use **CvArgsSet***AttrType* to fill in the CvArgs structure.
3. Issue one or more operational functions (such as **CvaAddObject**).
4. Issue **CvArgsFree** to free the memory used by the argument list.

See Also

CvArgsMakeIds and **CvArgsMakeVals**.

Purpose

Creates and sets the IDs for the attributes in an argument list.

Syntax

CvArgs CvArgsMakeIds(
CvArgId *AttrId***,**
...
ENDARG);

Description

CvArgsMakeIds creates and sets the IDs for the attributes in an argument list. It returns an opaque pointer to a CvArgs structure that specifies the attributes for an object.

AttrId is an attribute ID to be set in the argument list. The attribute is specified by its argument ID symbol, as defined in the file CvArgId.H. You can specify up to 160 attributes. The last parameter must be **ENDARG**. Specify only those attributes appropriate for the object type (refer to the *NavisXtend Provisioning Server Object Attribute Definitions*).

ENDARG is the last parameter in the sequence of attributes.

Prerequisites

None.

Post-processing

Use this function in conjunction with operational functions that manipulate an object's attributes:

1. Issue this function to create and fill in the argument list (CvArgs structure).
2. Issue one or more operational functions (such as **CvaAddObject**).
3. Issue **CvArgsFree** to free the memory used by the argument list.

See Also

CvArgsMake and **CvArgsMakeVals**.

CvaArgsMakeVals

Purpose

Creates and sets the values for the attributes in an argument list.

Syntax

```
CvArgs CvArgsMakeVals(
        CvArgId AttrId,
        Value,
        ...
        ENDARG );
```

Description

CvArgsMakeVals creates and sets the values for the attributes in an argument list. It returns an opaque pointer to a CvArgs structure that specifies the attributes for an object.

AttrId is an attribute to be set in the argument list. The attribute is specified by its argument ID symbol, as defined in the file CvArgId.H.

Value is an attribute's value to be set in the argument list. The value uses a data type appropriate for the argument.

You can specify up to 160 attribute-value pairs. Specify only those attributes and values appropriate for the object type (refer to the *NavisXtend Provisioning Server Object Attribute Definitions*).

ENDARG is the last parameter in the sequence of attribute-value pairs.

Prerequisites

None.

Post-processing

Use this function in conjunction with operational functions that manipulate an object's attributes:

1. Issue this function to create and fill in the argument list (CvArgs structure).

2. Issue one or more operational functions (such as **CvaAddObject**).

3. Issue **CvArgsFree** to free the memory used by the argument list.

See Also

CvArgsMake and **CvArgsMakeIds**.

CvArgsPrint

Purpose

Prints the text description of an argument list to a file.

Syntax

void CvArgsPrint(
FILE ****Fileptr*,**
CvConstArgs *Args* **);**

Description

CvArgsPrint prints text argument descriptions of an argument list to a file. No newline characters are inserted into the file.

Fileptr is a pointer to the file to which the text descriptions are written.

Args is a constant CvArgs structure that holds the attributes to be printed to a file. The API does not modify or delete this argument.

Prerequisites

Make sure that the argument list represented by *Args* is valid.

Post-processing

None.

See Also

CvArgsToString.

Purpose

Subtracts two argument lists.

Syntax

CvArgs CvArgsRemove(
CvArgs *Args1***,**
CvArgs *Args2* **);**

Description

CvArgsRemove subtracts two argument lists. The result is a new argument list that contains the arguments from the first set that are not contained in the second set.

Args1 and *Args2* are the CvArgs structures to be subtracted. Each argument list should be a valid, non-null pointer.

Prerequisites

Make sure that the argument lists represented by *Args1* and *Args2* are valid.

Post-processing

When you are finished with the function's returned value, you must issue **CvArgsFree** to free the memory used by the returned argument list.

See Also

CvArgsCombine and **CvArgsSelect**.

CvArgsSelect

Purpose

Returns the intersection of two argument lists.

Syntax

CvArgs CvArgsSelect(
CvArgs *Args1***,**
CvArgs *Args2* **);**

Description

CvArgsSelect returns the intersection of two argument lists. The result is a new argument list containing arguments that exist in both sets of arguments. Values from the first list are used.

Args1 and *Args2* are the CvArgs structures to be added.

Prerequisites

Make sure that the argument lists represented by *Args1* and *Args2* are valid.

Post-processing

When you are finished with the function's returned value, you must issue **CvArgsFree** to free the memory used by the returned argument list.

See Also

CvArgsRemove.

Purpose

Sets an integer value for a particular attribute in an argument list.

Syntax

void CvArgsSetInt(
CvArgs *Args***,**
CvArgId *Id***,**
int *Integer* **);**

Description

CvArgsSetInt sets the integer value for a particular attribute in an argument list. If a value already exists for that attribute, the function replaces the existing value with the given value.

Args is the CvArgs structure that holds the attribute to be set.

Id is the argument ID symbol for the attribute to be set. The argument ID symbol is defined in the file CvArgId.H.

Integer is the integer value to be set, in integer form.

Prerequisites

Make sure that the value of the specified attribute is an integer.

Post-processing

None.

See Also

CvArgsGetInt, **CvArgsGetIpAddr**, **CvArgsGetObjectId**, **CvArgsGetStr**, **CvArgsSetIpAddr**, **CvArgsSetNull**, **CvArgsSetObjectId**, and **CvArgsSetStr**.

CvArgsSetIpAddr

Purpose

Sets an IP address value for a particular attribute in an argument list.

Syntax

void CvArgsSetIpAddr(
CvArgs *Args***,**
CvArgId *Id***,**
CvIpAddr *IpAddr* **);**

Description

CvArgsSetIpAddr sets the IP address value for a particular attribute in an argument list. If a value already exists for that attribute, the function replaces the existing value with the given value.

Args is the CvArgs structure that holds the attribute to be set.

Id is the argument ID symbol for the attribute to be set. The argument ID symbol is defined in the file CvArgId.H.

IpAddr is the IP address value to be set.

Prerequisites

Make sure that the value of the specified attribute is an IP address.

Post-processing

None.

See Also

CvArgsGetInt, **CvArgsGetIpAddr**, **CvArgsGetObjectId**, **CvArgsGetStr**, **CvArgsSetInt**, **CvArgsSetNull**, **CvArgsSetObjectId**, and **CvArgsSetStr**.

Purpose

Sets a null value for a particular attribute in an argument list.

Syntax

void CvArgsSetNull(
CvArgs *Args***,**
CvArgId *Id* **);**

Description

CvArgsSetNull sets a null value for a particular attribute in an argument list. Use this function to add an attribute with no value to the argument list in preparation for a query.

Args is the CvArgs structure that holds the attribute to be set.

Id is the argument ID symbol for the attribute to be set. The argument ID symbol is defined in the file CvArgId.H.

Prerequisites

Use **CvArgsMake** to create the argument list (CvArgs structure).

Post-processing

Use this function in conjunction with an operational function that queries an object's attributes:

1. Issue this function to fill in the CvArgs structure.
2. Issue the operational function (such as **CvaGetObject**).
3. Issue **CvArgsFree** to free the memory used by the argument list.

See Also

CvArgsGetInt, **CvArgsGetIpAddr**, **CvArgsGetObjectId**, **CvArgsGetStr**, **CvArgsSetInt**, **CvArgsSetIpAddr**, **CvArgsSetObjectId**, and **CvArgsSetStr**.

CvArgsSetObjectId

Purpose

Sets an object ID value for a particular attribute in an argument list.

Syntax

void CvArgsSetObjectId(
CvArgs *Args*,
CvArgId *Id*,
const CvObjectId ***_ObjId_ **);

Description

CvArgsSetObjectId sets the object ID value for a particular attribute in an argument list. If a value already exists for that attribute, the function replaces the existing value with the given value.

Args is the CvArgs structure that holds the attribute to be set.

Id is the argument ID symbol for the attribute to be set. The argument ID symbol is defined in the file CvArgId.H.

ObjId is a pointer to the CvObjectId structure that specifies the object ID to be set.

Prerequisites

Make sure that the value of the specified attribute is an object ID.

Post-processing

None.

See Also

CvArgsGetInt, **CvArgsGetIpAddr**, **CvArgsGetObjectId**, **CvArgsGetStr**, **CvArgsSetInt**, **CvArgsSetIpAddr**, **CvArgsSetNull**, and **CvArgsSetStr**.

Purpose

Sets a string value for a particular attribute in an argument list.

Syntax

void CvArgsSetStr(
CvArgs *Args***,**
CvArgId *Id***,**
const char ***_String_ **);

Description

CvArgsSetStr sets the string value for a particular attribute in an argument list. If a value already exists for that attribute, the function replaces the existing value with the given value.

Args is the CvArgs structure that holds the attribute to be set.

Id is the argument ID symbol for the attribute to be set. The argument ID symbol is defined in the file CvArgId.H.

String is the string value to be set.

Prerequisites

Make sure that the value of the specified attribute is a string.

Post-processing

None.

See Also

CvArgsGetInt, **CvArgsGetIpAddr**, **CvArgsGetObjectId**, **CvArgsGetStr**, **CvArgsSetInt**, **CvArgsSetIpAddr**, **CvArgsSetNull**, and **CvArgsSetObjectId**.

CvArgsStatusAt

Purpose

Retrieves the error status code of an argument at a specified location in an argument list.

Syntax

CvStatus CvArgsStatusAt(
CvConstArgs *Args*,
int *N* **);**

Description

CvArgsStatusAt retrieves the error status code of an argument at a specified location in an argument list.

Args is a constant CvArgs structure. The API does not modify or delete this argument.

N is the zero-based index of the argument in the argument list. If *N* is out of range of the argument list, the status -1 is returned.

Prerequisites

Use this function in conjunction with other utility functions that traverse the argument list:

1. Obtain the argument list (CvArgs structure) by appropriate means, such as with **CvArgsMake** or **CvaGetObject**.
2. Issue **CvArgsErrorIndex** to determine if any argument in the argument list has an error status. If you use the returned value in another function, subtract 1 from the value.
3. Issue this function to retrieve the error status code of the argument.

Post-processing

None.

See Also

CvArgsErrorStatus and **CvArgsGetStatus**.

Purpose

Converts an argument list to a printable string format.

Syntax

char *CvArgsToString(
CvConstArgs *Args* **);**

Description

CvArgsToString converts an argument list to a printable string format.

Args is a constant CvArgs structure that specifies the attributes to be converted to text. The API does not modify or delete this argument.

Prerequisites

Make sure that the argument list represented by *Args* is valid.

Post-processing

Once the result has been used, you must issue **CvStringFree** to free the memory used by the returned text string.

To print the arguments to a file, issue **CvArgsPrint**.

See Also

CvArgsPrint and **CvStringFree**.

CvArgsTypeAt

Purpose

Retrieves a value's type at a specified location in an argument list.

Syntax

CvValueType CvArgsTypeAt(
CvArgs *Args*,
int *Index* **);**

Description

CvArgsTypeAt retrieves a value's type at a specified location in an argument list.

Args is a CvArgs structure.

Index is the zero-based index to the argument list.

Notes

Valid return values are as follows:

- INTEGER
- STRING
- OBJID
- IPADDR
- NULLOBJ

These values are defined in the file CvDefs.H.

Prerequisites

Make sure that the argument list represented by *Args* is valid.

To count the number of arguments in an argument list, issue **CvArgsCount**.

Post-processing

None.

See Also

CvArgsCount and **CvArgsIdAt**.

Purpose

Fills in a CvObjectId structure with a value for an object's APS ID.

Syntax

CvObjectId CvCreateApsId(
CvObjectId *PportID* **);**

Description

CvCreateApsId fills in the CvObjectId structure with a value for an object's APS ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

PportID is the CvObjectId structure that specifies the PPort that contains the APS. This structure is built by **CvCreatePPortId**.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvCreate***ObjectType***Id** functions.

CvCreateAssignedSvcSecScnId

Purpose

Fills in a CvObjectId structure with a value for an object's AssignedSvcSecScn ID.

Syntax

CvObjectId CvCreateAssignedSvcSecScnId(
CvObjectId *LportID*,
const char ***_Name_ **);

Description

CvCreateAssignedSvcSecScnId fills in the CvObjectId structure with a value for an object's AssignedSvcSecScn. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

LportID is the CvObjectId structure that specifies the LPort that contains the AssignedSvcSecScn. This structure is built by **CvCreateLPortId**.

Name is a pointer to the name string referring to an SvcSecScn object. That name is created by **CvCreateSvcSecScnId**.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also **CvCreate***ObjectType***Id** functions.

CvCreateAtmEndpointId

Purpose

Fills in a CvObjectId structure with a value for an object's circuit ID.

Syntax

CvObjectId CvCreateAtmEndpointId(
CvObjectId *ObjectID***,**
int *Vpi***,**
int *Vci* **);**

Description

CvCreateAtmEndpointId fills in the CvObjectId structure with a value for an object's ATM circuit ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

ObjectID is the CvObjectId structure that specifies the object that contains the circuit. The object could be an LPort (in which case this structure is built by **CvCreateLPortId**) or a ServiceName (in which case this structure is built by **CvCreateServiceNameId**).

Vpi and *Vci* are the VPI and VCI values of an endpoint of the circuit. Valid values for the VPI endpoint are 0-15.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing None.

See Also Other **CvCreate***ObjectType***Id** functions.

CvCreateAtmLPortId

Purpose

Fills in a CvObjectId structure with a value for an object's ATM LPort ID.

Syntax

CvObjectId CvCreateAtmLPortId(
CvObjectId *PportID*,
int *Vpi*,
int *Vci* **);**

Description

CvCreateAtmLPortId fills in the CvObjectId structure with a value for an atmXportForFrNni LPort. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

PPortID is the CvObjectId structure that specifies the PPort that contains the LPort. This structure is built by **CvCreatePPortId**. When the card is a channelized DS3 card, specify the channel that contains the LPort. In this case, the structure is built by **CvCreateChannelId**.

Vpi and *Vci* are the VPI and VCI values of an endpoint of the circuit.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing None.

See Also Other **CvCreate***ObjectType***Id** functions.

CvCreateCardId

Purpose

Fills in a CvObjectId structure with a value for an object's card ID.

Syntax

CvObjectId CvCreateCardId(
CvObjectId *SwitchID*,
int *Slot* **);**

Description

CvCreateCardId fills in the CvObjectId structure with a value for an object's card ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

SwitchID is the CvObjectId structure that specifies the switch that contains the card. This structure is built by **CvCreateSwitchId**.

Slot is the slot number of the card.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvCreate***ObjectType***Id** functions.

Purpose

Fills in a CvObjectId structure with a value for an object's channel ID.

Syntax

CvObjectId CvCreateChannelId(
CvObjectId *PportID*,
int *ChannelNumber* **);**

Description

CvCreateChannelId fills in the CvObjectId structure with a value for an object's channel ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

PPortID is the CvObjectId structure that specifies the PPort that contains the channel. This structure is built by **CvCreatePPortId**.

ChannelNumber is the number of the channel.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvCreate***ObjectType***Id** functions.

CvCreateCustomerId

Purpose

Fills in a CvObjectId structure with a value for an object's Customer ID.

Syntax

CvObjectId CvCreateCustomerId(
CvObjectId *NetworkID*,
const char ***_Name_ **);

Description

CvCreateCustomerId fills in the CvObjectId structure with a value for an object's Customer ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

NetworkID is the CvObjectId structure that specifies the network that contains the Customer. This structure is built by **CvCreateNetworkId** or **CvCreateNetworkIdFromString**.

Name is a pointer to the Customer name string.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

CvCreate*ObjectType***Id** functions.

Purpose

Fills in a CvObjectId structure with a value for an object's circuit ID.

Syntax

CvObjectId CvCreateDlciId(
CvObjectId *ObjectID*,
int *DlciNumber* **);**

Description

CvCreateDlciId fills in the CvObjectId structure with a value for an object's Frame Relay circuit ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

ObjectID is the CvObjectId structure that specifies the object that contains the circuit. The object could be an LPort (in which case this structure is built by **CvCreateLPortId**) or a ServiceName (in which case this structure is built by **CvCreateServiceNameId**).

DlciNumber is the DLCI number of an endpoint of the circuit.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also Other **CvCreate***ObjectType***Id** functions.

Purpose

Fills in a CvObjectId structure with a value for an object's Pfdl ID.

Syntax

CvObjectId CvCreateFdlId(
CvObjectId *PportID* **);**

Description

CvCreateFdlId fills in the CvObjectId structure with a value for an object's Pfdl ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

PportID is the CvObjectId structure that specifies the PPort that contains the Pfdl. This structure is built by **CvCreatePPortId**.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

CvCreateLPortId

Purpose

Fills in a CvObjectId structure with a value for an object's Frame Relay LPort ID.

Syntax

CvObjectId CvCreateLPortId(
CvObjectId *PPortID***,**
int *LPort* **);**

Description

CvCreateLPortId fills in the CvObjectId structure with a value for an object's Frame Relay LPort ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

PportID is the CvObjectId structure that specifies the PPort that contains the LPort. This structure is built by **CvCreatePPortId**. When the card is a channelized DS3 card, specify the channel that contains the LPort. In this case, the structure is built by **CvCreateChannelId**.

LPort is the port number of the LPort.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also Other **CvCreate***ObjectType***Id** functions.

CvCreateNetCacId

Purpose

Fills in a CvObjectId structure with a value for an object's NetCac ID.

Syntax

CvObjectId CvCreateNetCacId(
CvObjectId *NetworkID* **);**

Description

CvCreateNetCacId fills in the CvObjectId structure with a value for an object's NetCac ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

NetworkID is the CvObjectId structure that specifies the network that contains the NetCac. This structure is built by **CvCreateNetworkId** or **CvCreateNetworkIdFromString**.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

CvCreate*ObjectType***Id** functions.

Purpose

Fills in a CvObjectId structure with a value for an object's network ID.

Syntax

CvObjectId CvCreateNetworkId(
CvIpAddr *NetNumber* **);**

Description

CvCreateNetworkId fills in the CvObjectId structure with a value for an object's network ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

NetNumber is the network IP address of the object being specified, with the last byte set to 0.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

CvCreateNetworkIdFromString and other **CvCreate***ObjectType***Id** functions.

CvCreateNetworkIdFromString

Purpose

Fills in a CvObjectId structure with a value for an object's network ID, based on a text description of the network ID.

Syntax

CvObjectId CvCreateNetworkIdFromString(
const char ***_NetName_ **);

Description

CvCreateNetworkIdFromString fills in the CvObjectId structure with a value for an object's network ID, based on a text description of the network ID. The function issues the **gethostbyname()** function to obtain the IP address of the network. The last byte is set to 0.

CvCreateNetworkIdFromString is a member of the **CvCreate**_ObjectType_**Id** family of functions used to create an object ID based on the containment hierarchy.

NetName is a pointer to the name of the network.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate**_ObjectType_**Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

CvCreateNetworkId and other **CvCreate***ObjectType***Id** functions.

CvCreateNullId

Purpose

Fills in a CvObjectId structure with a null value for an object's ID.

Syntax

CvObjectId CvCreateNullId( )

Description

CvCreateNullId creates the CvObjectId structure for an unspecified object (CVT_Null). This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvCreate***ObjectType***Id** functions.

Purpose

Fills in a CvObjectId structure with a value for an object's Frame Relay PerformanceMonitor ID.

Syntax

CvObjectId CvCreatePerformanceMonitorId(
CvObjectId *PPortID* **);**

Description

CvCreatePerformanceMonitorId fills in the CvObjectId structure with a value for an object's PerformanceMonitor ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

PportID is the CvObjectId structure that specifies the PPort that contains the PerformanceMonitor. This structure is built by **CvCreatePPortId**.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvCreate***ObjectType***Id** functions.

CvCreatePMPCktLeafId

Purpose

Fills in a CvObjectId structure with a value for an object's PMP circuit leaf ID.

Syntax

CvObjectId CvCreatePMPCktLeafId(
CvObjectId *LportID*,
int *Vpi*,
int *Vci* **);**

Description

CvCreatePMPCktLeafId fills in the CvObjectId structure with a value for an object's PMP circuit leaf ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

LPortID is the CvObjectId structure that specifies the LPort that contains the PMP circuit leaf endpoint. This structure is built by **CvCreateLPortId**.

Vpi and *Vci* are the VPI and VCI values of an endpoint of the PMP circuit leaf. Valid values for the VPI endpoint are 0-15.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also Other **CvCreate***ObjectType***Id** functions.

CvCreatePMPCktRootId

Purpose

Fills in a CvObjectId structure with a value for an object's PMP circuit root ID.

Syntax

CvObjectId CvCreatePMPCktRootId(
CvObjectId *LportID*,
int *Vpi*,
int *Vci* **);**

Description

CvCreatePMPCktRootId fills in the CvObjectId structure with a value for an object's PMP circuit root ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

LPortID is the CvObjectId structure that specifies the LPort that contains the PMP circuit root endpoint. This structure is built by **CvCreateLPortId**.

Vpi and *Vci* are the VPI and VCI values of an endpoint of the PMP circuit root. Valid values for the VPI endpoint are 0-15.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also Other **CvCreate***ObjectType***Id** functions.

CvCreatePMPSpvcLeafId

Purpose

Fills in a CvObjectId structure with a value for an object's PMP SpvcLeaf ID.

Syntax

CvObjectId CvCreatePMPSpvcLeafId(
CvObjectId *LPortID*,
unsigned int *LeafNumber* **);**

Description

CvCreatePMPSpvcLeafId fills in the CvObjectId structure with a value for an object's SpvcLeaf ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

LPortID is the CvObjectId structure that specifies the LPort that contains the leaf. This structure is built by **CvCreateLPortId**.

LeafNumber is the leaf number, which is assigned internally.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvCreate***ObjectType***Id** functions.

Purpose

Fills in a CvObjectId structure with a value for an object's PMP soft PVC root ID.

Syntax

CvObjectId CvCreatePMPSpvcRootId(
CvObjectId *LportID*,
int *Vpi*,
int *Vci* **);**

Description

CvCreatePMPSpvcRootId fills in the CvObjectId structure with a value for an object's PMP soft PVC root ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

LPortID is the CvObjectId structure that specifies the LPort that contains the PMP soft PVC root endpoint. This structure is built by **CvCreateLPortId**.

Vpi and *Vci* are the VPI and VCI values of an endpoint of the PMP soft PVC root. Valid values for the VPI endpoint are 0-15.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also Other **CvCreate***ObjectType***Id** functions.

Purpose

Fills in a CvObjectId structure with a value for an object's PPort ID.

Syntax

CvObjectId CvCreatePPortId(
CvObjectId *CardID*,
int *Port* **);**

Description

CvCreatePPortId fills in the CvObjectId structure with a value for an object's PPort ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

CardID is the CvObjectId structure that specifies the card that contains the PPort. This structure is built by **CvCreateCardId**.

Port is the port number of the PPort.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvCreate***ObjectType***Id** functions.

CvCreateServiceNameId

Purpose

Fills in a CvObjectId structure with a value for an object's Service Name ID.

Syntax

CvObjectId CvCreateServiceNameId(
CvObjectId *NetworkID*,
const char ***ServiceName* **);

Description

CvCreateServiceNameId fills in the CvObjectId structure with a value for an object's Service Name ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

NetworkID is the CvObjectId structure that specifies the network that contains the Service Name. This structure is built by **CvCreateNetworkId** or **CvCreateNetworkIdFromString**.

ServiceName is a pointer to the Service Name string.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

CvCreate*ObjectType***Id** functions.

Purpose

Fills in a CvObjectId structure with a value for an object's SMDS alien group address ID.

Syntax

CvObjectId CvCreateSmdsAlienGroupAddrId(
CvObjectId *SwitchID*,
const char ***_Addr_ **);

Description

CvCreateSmdsAlienGroupAddrId fills in the CvObjectId structure with a value for an object's SMDS alien group address ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

SwitchID is the CvObjectId structure that specifies the switch that contains the SMDS alien group address. This structure is built by **CvCreateSwitchId**.

Addr is a pointer to the E.164 address string of the SMDS alien group address. The address must be from 10 to 16 digits long.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also Other **CvCreate***ObjectType***Id** functions.

Purpose

Fills in a CvObjectId structure with a value for an object's SMDS alien individual address ID.

Syntax

CvObjectId CvCreateSmdsAlienIndivAddrId(
CvObjectId *SwitchID*,
const char ***_Addr_ **);

Description

CvCreateSmdsAlienIndivAddrId fills in the CvObjectId structure with a value for an object's SMDS alien individual address ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

SwitchID is the CvObjectId structure that specifies the switch that contains the SMDS alien individual address. This structure is built by **CvCreateSwitchId**.

Addr is a pointer to the E.164 address string of the SMDS alien individual address. The address must be from 10 to 16 digits long.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also Other **CvCreate***ObjectType***Id** functions.

Purpose

Fills in a CvObjectId structure with a value for an object's SMDS country code ID.

Syntax

CvObjectId CvCreateSmdsCountryCodeId(
CvObjectId *NetworkID*,
const char ***Ccode **);

Description

CvCreateSmdsCountryCodeId fills in the CvObjectId structure with a value for an object's country code ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

NetworkID is the CvObjectId structure that specifies the network that contains the SMDS country code. This structure is built by **CvCreateNetworkId** or **CvCreateNetworkIdFromString**.

Ccode is a pointer to the SMDS country code.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

CvCreate*ObjectType***Id** functions.

CvCreateSmdsGroupAddrId

Purpose

Fills in a CvObjectId structure with a value for an object's SMDS group address ID.

Syntax

CvObjectId CvCreateSmdsGroupAddrId(
CvObjectId *SwitchID*,
const char ***_Addr_ **);

Description

CvCreateSmdsGroupAddrId fills in the CvObjectId structure with a value for an object's SMDS group address ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

SwitchID is the CvObjectId structure that specifies the switch that contains the SMDS group address. This structure is built by **CvCreateSwitchId**.

Addr is a pointer to the E.164 address string of the SMDS group address. The address must be from 10 to 16 digits long.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also Other **CvCreate***ObjectType***Id** functions.

CvCreateSmdsGroupScreenId

Purpose

Fills in a CvObjectId structure with an object's SMDS group screen ID.

Syntax

CvObjectId CvCreateSmdsGroupScreenId(
CvObjectId *LPortID* **);**

Description

CvCreateSmdsGroupScreenId fills in the CvObjectId structure with an object's SMDS group screen ID. No value is required for an SMDS group screen. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

LPortID is the CvObjectId structure that specifies the LPort that contains the SMDS group screen. This structure is built by **CvCreateLPortId**.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvCreate***ObjectType***Id** functions.

Purpose

Fills in a CvObjectId structure with an object's SMDS individual screen ID.

Syntax

CvObjectId CvCreateSmdsIndivScreenId(
CvObjectId *LPortID* **);**

Description

CvCreateSmdsIndivScreenId fills in the CvObjectId structure with an object's SMDS individual screen ID. No value is required for an SMDS individual screen. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

LPortID is the CvObjectId structure that specifies the LPort that contains the SMDS individual screen. This structure is built by **CvCreateLPortId**.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvCreate***ObjectType***Id** functions.

CvCreateSmdsLocalIndivAddrId

Purpose

Fills in a CvObjectId structure with a value for an object's SMDS local individual address ID.

Syntax

CvObjectId CvCreateSmdsLocalIndivAddrId(
CvObjectId *LPortID*,
const char ***_Addr_ **);

Description

CvCreateSmdsLocalIndivAddrId fills in the CvObjectId structure with a value for an object's SMDS local individual address ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

LPortID is the CvObjectId structure that specifies the LPort that contains the SMDS local individual address. This structure is built by **CvCreateLPortId**.

Addr is a pointer to the E.164 address string of the SMDS local individual address. The address must be from 10 to 16 digits long.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also Other **CvCreate***ObjectType***Id** functions.

CvCreateSmdsNetwideGroupAddrId

Purpose

Fills in a CvObjectId structure with a value for an object's SMDS netwide group address ID.

Syntax

CvObjectId CvCreateSmdsNetwideGroupAddrId(
CvObjectId *NetID*,
const char ***_Addr_ **);

Description

CvCreateSmdsNetwideGroupAddrId fills in the CvObjectId structure with a value for an object's SMDS netwide group address ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

NetID is the CvObjectId structure that specifies the network that contains the SMDS netwide group address. This structure is built by **CvCreateNetworkId**.

Addr is a pointer to the E.164 address string of the SMDS netwide group address. The address must be from 10 to 16 digits long.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also Other **CvCreate***ObjectType***Id** functions.

CvCreateSmdsPrefixId

Purpose

Fills in a CvObjectId structure with a value for an object's SMDS address prefix ID.

Syntax

CvObjectId CvCreateSmdsPrefixId(
CvObjectId *SwitchID*,
const char ***_Prefix_ **);

Description

CvCreateSmdsPrefixId fills in the CvObjectId structure with a value for an object's SMDS address prefix ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

SwitchID is the CvObjectId structure that specifies the switch that contains the SMDS address prefix ID. This structure is built by **CvCreateSwitchId**.

Prefix is a pointer to the E.164 address string of the SMDS address prefix (3 to 6 characters).

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also Other **CvCreate***ObjectType***Id** functions.

CvCreateSmdsSsiIndivAddrId

Purpose

Fills in a CvObjectId structure with a value for an object's SMDS SSI individual address ID.

Syntax

CvObjectId CvCreateSmdsSsiIndivAddrId(
CvObjectId *LPortID*,
const char ***_Addr_ **);

Description

CvCreateSmdsSsiIndivAddrId fills in the CvObjectId structure with a value for an object's SMDS SSI individual address ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

LPortID is the CvObjectId structure that specifies the LPort that contains the SMDS SSI individual address. This structure is built by **CvCreateLPortId**.

Addr is a pointer to the E.164 address string of the SMDS SSI individual address. The address must be from 10 to 16 digits long.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also Other **CvCreate***ObjectType***Id** functions.

CvCreateSpvcId

Purpose

Fills in a CvObjectId structure with a value for an object's soft PVC circuit ID.

Syntax

CvObjectId CvCreateSpvcId(
CvObjectId *LPortID*,
int *Vpi*,
int *Vci* **);**

Description

CvCreateSpvcId fills in the CvObjectId structure with a value for an object's soft PVC circuit ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

LPortID is the CvObjectId structure that specifies the LPort that contains the soft PVC circuit endpoint. This structure is built by **CvCreateLPortId**.

Vpi and *Vci* are the VPI and VCI values of an endpoint of the soft PVC circuit. Valid values for the VPI endpoint are 0-15.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also Other **CvCreate***ObjectType***Id** functions.

CvCreateSvcAddressId

Purpose

Fills in a CvObjectId structure with a value for an object's SvcAddress ID.

Syntax

CvObjectId CvCreateSvcAddressId(
CvObjectId *LPortID*,
const char ***_Addr_ **);

Description

CvCreateSvcAddressId fills in the CvObjectId structure with an object's SvcAddress ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

LPortID is the CvObjectId structure that specifies the LPort that contains the SvcAddress. This structure is built by **CvCreateLPortId**.

Addr is a pointer to the SVC address string. The string must conform to the convention used to specify SVC addresses (refer to Chapter 1 of the *NavisXtend Provisioning Server User's Guide* for details).

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also Other **CvCreate***ObjectType***Id** functions.

CvCreateSvcConfigId

Purpose

Fills in a CvObjectId structure with a value for an object's SvcConfig ID.

Syntax

CvObjectId CvCreateSvcConfigId(
CvObjectId *LPortID* **);**

Description

CvCreateSvcConfigId fills in the CvObjectId structure with an object's SvcConfig ID. No value is required for an SVCConfig. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

LPortID is the CvObjectId structure that specifies the LPort that contains the SvcConfig. This structure is built by **CvCreateLPortId**.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvCreate***ObjectType***Id** functions.

Purpose

Fills in a CvObjectId structure with a value for an object's SvcCUG ID.

Syntax

CvObjectId CvCreateSvcCUGId(
CvObjectId *NetworkID*,
const char ***_Name_ **);

Description

CvCreateSvcCUGId fills in the CvObjectId structure with a value for an object's SvcCUG ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

NetworkID is the CvObjectId structure that specifies the network that contains the SvcCUG. This structure is built by **CvCreateNetworkId** or **CvCreateNetworkIdFromString**.

Name is a pointer to the name string of the SvcCUG.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

CvCreate*ObjectType***Id** functions.

CvCreateSvcCUGMbrId

Purpose

Fills in a CvObjectId structure with a value for an object's SvcCUGMbr ID.

Syntax

CvObjectId CvCreateSvcCUGMbrId(
CvObjectId *CUGID*,
const char ***_Name_ **);

Description

CvCreateSvcCUGMbrId fills in the CvObjectId structure with a value for an object's SvcCUGMbr ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

CUGID is the CvObjectId structure that specifies the SvcCUG that contains the SvcCUGMbr. This structure is built by **CvCreateSvcCUGId**.

Name is a pointer to the member name string. The name must be an existing name for the CUG member rule. The function binds the name string of the parent SvcCUG object to this member name string.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also **CvCreate***ObjectType***Id** functions.

CvCreateSvcCUGMbrRuleId

Purpose

Fills in a CvObjectId structure with a value for an object's SvcCUGMbrRule ID.

Syntax

CvObjectId CvCreateSvcCUGMbrRuleId(
CvObjectId *NetworkID*,
const char ***_Name_ **);

Description

CvCreateSvcCUGMbrRuleId fills in the CvObjectId structure with a value for an object's SvcCUGMbrRule ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

NetworkID is the CvObjectId structure that specifies the network that contains the SvcCUGMbrRule. This structure is built by **CvCreateNetworkId** or **CvCreateNetworkIdFromString**.

Name is a pointer to the name string of the SvcCUGMbrRule.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

CvCreate*ObjectType***Id** functions.

Purpose

Fills in a CvObjectId structure with a value for an object's SvcNodePrefix ID.

Syntax

CvObjectId CvCreateSvcNodePrefixId(
CvObjectId *SwitchID*,
const char ***Addr **);

Description

CvCreateSvcNodePrefixId fills in the CvObjectId structure with an object's SvcNodePrefix ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

SwitchID is the CvObjectId structure that specifies the switch that contains the SvcNodePrefix. This structure is built by **CvCreateSwitchId** or **CvCreateSwitchIdFromString**.

Addr is a pointer to the SVC address string. The string must conform to the convention used to specify SVC addresses (refer to Chapter 1 of the *NavisXtend Provisioning Server User's Guide* for details).

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also Other **CvCreate***ObjectType***Id** functions.

Purpose

Fills in a CvObjectId structure with a value for an object's SvcPrefix ID.

Syntax

CvObjectId CvCreateSvcPrefixId(
CvObjectId *LPortID*,
const char ***Addr **);

Description

CvCreateSvcPrefixId fills in the CvObjectId structure with an object's SvcPrefix ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

LPortID is the CvObjectId structure that specifies the LPort that contains the SvcPrefix. This structure is built by **CvCreateLPortId**.

Addr is a pointer to the SVC address string. The string must conform to the convention used to specify SVC addresses (refer to Chapter 1 of the *NavisXtend Provisioning Server User's Guide* for details).

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also Other **CvCreate***ObjectType***Id** functions.

Purpose

Fills in a CvObjectId structure with a value for an object's SvcSecScn ID.

Syntax

CvObjectId CvCreateSvcSecScnId(
CvObjectId *NetworkID*,
const char ***Name **);

Description

CvCreateSvcSecScnId fills in the CvObjectId structure with a value for an object's SvcSecScn. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

NetworkID is the CvObjectId structure that specifies the network that contains the SvcSecScn. This structure is built by **CvCreateNetworkId** or **CvCreateNetworkIdFromString**.

Name is a pointer to the name string of the SvcSecScn.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

CvCreate*ObjectType***Id** functions.

CvCreateSvcSecScnActParamId

Purpose

Fills in a CvObjectId structure with a value for an object's SvcSecScnActParam ID.

Syntax

CvObjectId CvCreateSvcSecScnActParamId(
CvObjectId *LportID* **);**

Description

CvCreateSvcSecScnActParamId fills in the CvObjectId structure with an object's SvcSecScnActParam ID. No value is required for an SvcSecScnActParam. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

LPortID is the CvObjectId structure that specifies the LPort that contains the SvcSecScnActParam. This structure is built by **CvCreateLPortId**.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvCreate***ObjectType***Id** functions.

Purpose

Fills in a CvObjectId structure with a value for an object's SvcUserPart ID.

Syntax

CvObjectId CvCreateSvcUserPartId(
CvObjectId *LPortID*,
const char ***Addr **);

Description

CvCreateSvcUserPartId fills in the CvObjectId structure with an object's SvcUserPart ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

LPortID is the CvObjectId structure that specifies the LPort that contains the SvcUserPart. This structure is built by **CvCreateLPortId**.

Addr is a pointer to the SVC address string. The string must conform to the convention used to specify SVC addresses (refer to Chapter 1 of the *NavisXtend Provisioning Server User's Guide* for details).

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also Other **CvCreate***ObjectType***Id** functions.

Purpose

Fills in a CvObjectId structure with a value for an object's switch ID.

Syntax

CvObjectId CvCreateSwitchId(
CvIpAddr *SwitchNumber* **);**

Description

CvCreateSwitchId fills in the CvObjectId structure with a value for an object's switch ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

SwitchNumber is the network IP address of the object being specified.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

CvCreateSwitchIdFromString and other **CvCreate***ObjectType***Id** functions.

CvCreateSwitchIdFromString

Purpose

Fills in a CvObjectId structure with a value for an object's switch ID, based on a text description of the switch.

Syntax

CvObjectId CvCreateSwitchIdFromString(
const char ****SwitchName* **);

Description

CvCreateSwitchIdFromString fills in the CvObjectId structure with a value for an object's switch ID, based on a text description of the switch. The function issues the **gethostbyname()** function to obtain the IP address of the switch.

CvCreateSwitchIdFromString is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

SwitchName is a pointer to the name of the switch.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

CvCreateSwitchId and other **CvCreate***ObjectType***Id** functions.

Purpose

Fills in a CvObjectId structure with a value for an object's Traffic Descriptor ID.

Syntax

CvObjectId CvCreateTrafficDescId(
CvObjectId *NetworkID*,
const char ***_Name_ **);

Description

CvCreateTrafficDescId fills in the CvObjectId structure with a value for an object's Traffic Descriptor ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

NetworkID is the CvObjectId structure that specifies the network that contains the Traffic Descriptor. This structure is built by **CvCreateNetworkId** or **CvCreateNetworkIdFromString**.

Name is a pointer to the name of the Traffic Descriptor.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvCreate***ObjectType***Id** functions.

CvCreateTrafficShaperId

Purpose

Fills in a CvObjectId structure with a value for an object's TrafficShaper ID.

Syntax

CvObjectId CvCreateTrafficShaperId(
CvObjectId *PportID*,
unsigned int *TrafficShaperId* **);**

Description

CvCreateTrafficShaperId fills in the CvObjectId structure with a value for an object's TrafficShaper ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

PportID is the CvObjectId structure that specifies the PPort that contains the Traffic Shaper. This structure is built by **CvCreatePPortId**.

TrafficShaperId is the one-based identifier into the traffic shaper attributes (range 1 - 15).

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also Other **CvCreate***ObjectType***Id** functions.

CvCreateVirtualUniLPortId

Purpose

Fills in a CvObjectId structure with a value for an object's ATM Virtual UNI LPort start VPI value.

Syntax

CvObjectId CvCreateVirtualUniLPortId(
CvObjectId *PportID*,
unsigned int *StartVpi* **);**

Description

CvCreateVirtualUniLPortId fills in the CvObjectId structure with a value for an object's ATM Virtual UNI LPort start VPI value. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

PportID is the CvObjectId structure that specifies the PPort that contains the LPort. This structure is built by **CvCreatePPortId**.

StartVpi is the number that represents the lower end of the range of VPI values of the virtual UNI LPort.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also Other **CvCreate***ObjectType***Id** functions.

CvCreateVPNId

Purpose

Fills in a CvObjectId structure with a value for an object's VPN ID.

Syntax

CvObjectId CvCreateVPNId(
CvObjectId *NetworkID*,
const char ***_Name_ **);

Description

CvCreateVPNId fills in the CvObjectId structure with a value for an object's VPN ID. This function is a member of the **CvCreate***ObjectType***Id** family of functions used to create an object ID based on the containment hierarchy.

NetworkID is the CvObjectId structure that specifies the network that contains the VPN. This structure is built by **CvCreateNetworkId** or **CvCreateNetworkIdFromString**.

Name is a pointer to the name string of the VPN.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build a CvObjectId structure with the object's values. Use this function in conjunction with other **CvCreate***ObjectType***Id** functions to build the structure based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

Prerequisites

None.

Post-processing

None.

See Also

CvCreate*ObjectType***Id** functions.

Purpose

Reads the SMDS address out of a CvObjectId structure.

Syntax

const char *CvGetAddress(
CvObjectId *ObjId* **);**

Description

CvGetAddress reads the 10-character E.164 address string of the SMDS address out of a CvObjectId structure. This function is a member of the **CvGet***ObjectTypeValue* family of functions used to read specified values out of the object ID.

ObjId is the CvObjectId structure that specifies the object.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvGet***ObjectTypeValue* functions.

CvGetArgumentName

Purpose

Converts an argument to a printable string format.

Syntax

const char *CvGetArgumentName(
CvArgId *Id* **);**

Description

CvGetArgumentName returns the name for a given argument.

Id is the argument ID symbol for the attribute to be converted to text. The argument ID symbol is defined in the file CvArgId.H.

Prerequisites

None.

Post-processing

Do not free or alter the string returned by the function.

See Also

CvGetEnumName, **CvGetObjectType**, **CvParseObjectId**, and **CvParseObjectType**.

Purpose

Reads the card slot number out of a CvObjectId structure.

Syntax

int CvGetCardNumber(
CvObjectId *ObjId* **);**

Description

CvGetCardNumber reads the card slot number out of a CvObjectId structure. This function is a member of the **CvGet***ObjectTypeValue* family of functions used to read specified values out of the object ID.

ObjId is the CvObjectId structure that specifies the object.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvGet***ObjectTypeValue* functions.

CvGetChannelNumber

Purpose

Reads the channel number out of a CvObjectId structure.

Syntax

int CvGetChannelNumber(
CvObjectId *ObjId* **);**

Description

CvGetChannelNumber reads the channel number (range 1 - 28) out of a CvObjectId structure. This function is a member of the **CvGet***ObjectTypeValue* family of functions used to read specified values out of the object ID.

ObjId is the CvObjectId structure that specifies the object.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvGet***ObjectTypeValue* functions.

Purpose

Reads the circuit DLCI number out of a CvObjectId structure.

Syntax

int CvGetDlciNumber(
CvObjectId *ObjId* **);**

Description

CvGetDlciNumber reads the DLCI number of an endpoint of the Frame Relay circuit out of a CvObjectId structure. This function is a member of the **CvGet***ObjectTypeValue* family of functions used to read specified values out of the object ID.

ObjId is the CvObjectId structure that specifies the object.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvGet***ObjectTypeValue* functions.

CvGetEnumName

Purpose

Converts an enumerated value to a printable string format.

Syntax

const char *CvGetEnumName(
CvArgId *Id*,
int *Value* **);**

Description

CvGetEnumName returns the name of a given enumerated value.

Id is the argument ID symbol for the attribute whose enumerated value is to be converted to text. The argument ID symbol is defined in the file CvArgId.H.

Value is the integer value of the enumerated type to be converted to text.

Notes

If the argument ID does not represent an enumerated argument, the function returns 0.

Prerequisites

None.

Post-processing

Do not free or alter the string returned by the function.

See Also

CvGetArgumentName, **CvGetObjectType**, **CvParseObjectId**, and **CvParseObjectType**.

Purpose

Reads the LPort number out of a CvObjectId structure.

Syntax

int CvGetLPortNumber(
CvObjectId *ObjId* **);**

Description

CvGetLPortNumber reads the LPort number out of a CvObjectId structure. This function is a member of the **CvGet***ObjectTypeValue* family of functions used to read specified values out of the object ID.

ObjId is the CvObjectId structure that specifies the object.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvGet***ObjectTypeValue* functions.

CvGetName

Purpose

Reads the name string out of a CvObjectId structure.

Syntax

const char *CvGetName(
CvObjectId *ObjId* **);**

Description

CvGetName reads the name string of a CvObjectId structure. This function is a member of the **CvGet***ObjectTypeValue* family of functions used to read specified values out of the object ID.

ObjId is the CvObjectId structure that specifies the object.

Notes

Valid object types for this function are as follows:

CVT_AssignedSvcSecScn
CVT_Customer
CVT_ServiceName
CVT_SvcCUG
CVT_SvcCUGMbr
CVT_SvcCUGMbrRule
CVT_SvcSecScn
CVT_TrafficDesc
CVT_Vpn

For an SvcCUGMbr object, this function reads the CUG name string. To read the member name string out of an SvcCUGMbr object, use **CvGetName2**.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvGet***ObjectTypeValue* functions.

Purpose

Reads the member name string of an SvcCUGMbr object out of a CvObjectId structure.

Syntax

const char *CvGetName2(
CvObjectId *ObjId* **);**

Description

CvGetName2 reads the member name string of an SvcCUGMbr object out of a CvObjectId structure. This function is a member of the **CvGet***ObjectTypeValue* family of functions used to read specified values out of the object ID.

ObjId is the CvObjectId structure that specifies the object.

Notes

To read the name string of other types of objects (including the CUG name string of an SvcCUGMbr object), use **CvGetName**.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvGet***ObjectTypeValue* functions.

CvGetNetworkNumber

Purpose

Reads the network IP address out of a CvObjectId structure.

Syntax

CvIpAddr CvGetNetworkNumber(
CvObjectId *ObjId* **);**

Description

CvGetNetworkNumber reads the network IP address out of a CvObjectId structure. This function is a member of the **CvGet***ObjectTypeValue* family of functions used to read specified values out of the object ID.

ObjId is the CvObjectId structure that specifies the object.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvGet***ObjectTypeValue* functions.

Purpose

Reads the object type out of a CvObjectId structure.

Syntax

CvObjectType CvGetObjectType(
CvObjectId *ObjId* **);**

Description

CvGetObjectType reads the object type out of a CvObjectId structure. This function is a member of the **CvGet***ObjectTypeValue* family of functions used to read specified values out of the object ID.

ObjId is the CvObjectId structure that specifies the object.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvGet***ObjectTypeValue* functions.

CvGetObjectTypeName

Purpose

Converts an object type to a printable string format.

Syntax

const char *CvGetObjectTypeName(
CvObjectType *ObjType* **);**

Description

CvGetObjectTypeName converts an object type to a printable string format.

ObjType is the enumerated value that specifies the object type to be converted to text.

Prerequisites

Use **CvGetObjectType** to read the object type out of a CvObjectId structure.

Post-processing

Do not free or alter the string returned by the function.

See Also

CvGetArgumentName, **CvGetEnumName**, **CvGetObjectType**, **CvParseObjectId**, and **CvParseObjectType**.

Purpose Reads the PMP SpvcLeaf number out of a CvObjectId structure.

Syntax

```
int CvGetPMPSpvcLeafId(
        CvObjectId ObjId );
```

Description **CvGetPMPSpvcLeafId** reads the PMP SpvcLeaf number out of a CvObjectId structure. This function is a member of the **CvGet***ObjectTypeValue* family of functions used to read specified values out of the object ID.

ObjId is the CvObjectId structure that specifies the object.

Prerequisites None.

Post-processing None.

See Also Other **CvGet***ObjectTypeValue* functions.

CvGetPPortNumber

Purpose

Reads the PPort number out of a CvObjectId structure.

Syntax

int CvGetPPortNumber(
CvObjectId *ObjId* **);**

Description

CvGetPPortNumber reads the PPort number out of a CvObjectId structure. This function is a member of the **CvGet***ObjectTypeValue* family of functions used to read specified values out of the object ID.

ObjId is the CvObjectId structure that specifies the object.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvGet***ObjectTypeValue* functions.

Purpose

Reads the SMDS address prefix out of a CvObjectId structure.

Syntax

const char *CvGetPrefix(
CvObjectId *ObjId* **);**

Description

CvGetPrefix reads the E.164 address string of the SMDS address prefix out of a CvObjectId structure (3 to 6 characters). This function is a member of the **CvGet***ObjectTypeValue* family of functions used to read specified values out of the object ID.

ObjId is the CvObjectId structure that specifies the object.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvGet***ObjectTypeValue* functions.

CvGetStartVpi

Purpose

Reads the number that represents the lower end of the range of VPI values out of a CvObjectId structure for a virtual UNI LPort.

Syntax

int CvGetStartVpi(
CvObjectId *ObjId* **);**

Description

CvGetStartVpi reads the lower end of the range of VPI values out of a CvObjectId structure for a virtual UNI LPort. This function is a member of the **CvGet***ObjectTypeValue* family of functions used to read specified values out of the object ID.

ObjId is the CvObjectId structure that specifies the object.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvGet***ObjectTypeValue* functions.

Purpose

Reads an SVC address out of a CvObjectId structure.

Syntax

const char *CvGetSvcAddress(
CvObjectId *ObjId* **);**

Description

CvGetSvcAddress reads the SVC address out of a CvObjectId structure. This function is a member of the **CvGet***ObjectTypeValue* family of functions used to read specified values out of the object ID.

ObjId is the CvObjectId structure that specifies the object.

Notes

Valid object types for this function are as follows:

CVT_SvcNodePrefix
CVT_SvcPrefix
CVT_SvcAddress
CVT_SvcUserPart

Prerequisites

None.

Post-processing

None.

See Also

Other **CvGet***ObjectTypeValue* functions.

CvGetSwitchId

Purpose

Reads the switch IP address out of a CvObjectId structure.

Syntax

CvIpAddr CvGetSwitchId(
CvObjectId *ObjId* **);**

Description

CvGetSwitchId reads the switch IP address out of a CvObjectId structure. This function is a member of the **CvGet***ObjectTypeValue* family of functions used to read specified values out of the object ID.

ObjId is the CvObjectId structure that specifies the object.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvGet***ObjectTypeValue* functions.

Purpose

Reads the Traffic Shaper identifier value out of a CvObjectId structure.

Syntax

int CvGetTrafficShaperId(
CvObjectId *ObjId* **);**

Description

CvGetTrafficShaperId reads the Traffic Shaper identifier value out of a CvObjectId structure. This function is a member of the **CvGet***ObjectTypeValue* family of functions used to read specified values out of the object ID.

ObjId is the CvObjectId structure that specifies the object.

Prerequisites

None.

Post-processing

None.

See Also

Other **CvGet***ObjectTypeValue* functions.

CvGetVciNumber

Purpose

Reads the circuit VCI number out of a CvObjectId structure.

Syntax

int CvGetVciNumber(
CvObjectId *ObjId* **);**

Description

CvGetVciNumber reads the VCI number of an endpoint of the ATM circuit out of a CvObjectId structure. This function is a member of the **CvGet***ObjectTypeValue* family of functions used to read specified values out of the object ID.

ObjId is the CvObjectId structure that specifies the object.

Notes

Valid object types for this function are as follows:

CVT_Circuit
CVT_Spvc
CVT_PMPCktRoot
CVT_PMPCktLeaf
CVT_PMPSpvcRoot

Prerequisites

None.

Post-processing

None.

See Also

Other **CvGet***ObjectTypeValue* functions.

Purpose

Reads the circuit VPI number out of a CvObjectId structure.

Syntax

int CvGetVpiNumber(
CvObjectId *ObjId* **);**

Description

CvGetVpiNumber reads the VPI number of an endpoint of the ATM circuit out of a CvObjectId structure. This function is a member of the **CvGet***ObjectTypeValue* family of functions used to read specified values out of the object ID.

ObjId is the CvObjectId structure that specifies the object.

Notes

Valid object types for this function are as follows:

CVT_Circuit
CVT_Spvc
CVT_PMPCktRoot
CVT_PMPCktLeaf
CVT_PMPSpvcRoot

Prerequisites

None.

Post-processing

None.

See Also

Other **CvGet***ObjectTypeValue* functions.

CvObjectIdPrint

Purpose

Prints the text description of a CvObjectId structure to a file.

Syntax

void CvObjectIdPrint(
FILE ****Fileptr*,**
CvObjectId *ObjId* **);**

Description

CvObjectIdPrint prints text description of a CvObjectId structure to a file. No newline characters are inserted into the file.

Fileptr is a pointer to the file to which the text descriptions are written.

ObjId is the CvObjectId structure that specifies the object.

Prerequisites

Use **CvObjectIdToString** to convert a CvObjectId structure to text argument descriptions.

Post-processing

None.

See Also

CvObjectIdToString and **CvParseObjectId**.

Purpose

Converts a CvObjectId structure to text descriptions.

Syntax

char *CvObjectIdToString(
CvObjectId *ObjId* **);**

Description

CvObjectIdToString converts a CvObjectId structure to text descriptions.

ObjId is the CvObjectId structure that specifies the object.

Prerequisites

None.

Post-processing

Once the function has been issued, you must issue **CvStringFree** to free the memory used by the returned text string.

To print the text string to a file, issue **CvObjectIdPrint**.

See Also

CvObjectIdPrint, **CvParseObjectId**, and **CvStringFree**.

CvParseObjectId

Purpose

Converts text descriptions of an object ID to a CvObjectId structure.

Syntax

CvObjectId CvParseObjectId(
const char **ValString* **);**

Description

CvParseObjectId converts text descriptions of an object ID to a CvObjectId structure. Use this function to process object descriptions stored in a text file or relational database.

ValString is the text argument description of the object.

Prerequisites

None.

Post-processing

None.

See Also

CvObjectIdToString and **CvParseObjectType**.

Purpose

Converts a text argument description of an object type to a CvObjectType.

Syntax

CvObjectType CvParseObjectType(
const char **ValString* **);**

Description

CvParseObjectType converts a text argument description of an object type to a CvObjectType. Use this function to process an object type stored in a text file or relational database.

ValString is the text description of the object type.

Notes

If the function cannot convert the text description to an object type (for example, if the specified text description is invalid), the function returns the value CVT_Unknown.

Prerequisites

None.

Post-processing

None.

See Also

CvGetObjectType, **CvGetObjectTypeName**, and **CvParseObjectId**.

CvStringFree

Purpose

Deletes a pointer to a string.

Syntax

CvStringFree(
char ****String* **);

Description

CvStringFree deletes a pointer to a string.

String is a pointer to a string. This pointer is returned by **CvArgsToString** or **CvObjectIdToString**.

Prerequisites

Before you issue this function, use either:

- **CvArgsToString** to convert an argument list to a string.
- **CvObjectIdToString** to convert an object Id to a string.

Post-processing

None.

See Also

CvArgsToString and **CvObjectIdToString**.

3

C++ Functions

This chapter describes the C++ classes and member functions of the Provisioning Server API.

C++ Classes

The **CvClient** class serves as the session context for the client. **CvClient** contains the information needed by the API to route requests to the Provisioning server and receive responses back from the server. And, **CvClient** contains operational member functions that perform provisioning operations on the network components.

The **CvClient** class has several nested classes:

CvClient::Callback — A derived class that implements a virtual response function.

CvClient::ObjectId — A class that represents an object identifier, which identifies the objects to be acted on and contains member functions to access and specify the identifying information.

CvClient::Args — A class that represents the argument list and contains member functions to add values to and read values from the argument list.

CvClient::Value — A helper class used to support multiple data types. This class contains constructors and cast operators for the supported data types and contains a member function to declare a null value.

CvClient::IpAddr — A helper class used to create IP addresses using either string or integer representations. This class contains constructors and cast operators for the supported data types.

CvClient::ApiStatus — A class that represents status codes. This class ensures that objects of type CvStatus are initilized (to **cveNoError** by default). This class has a constructor that performs this initialization.

This chapter presents **CvClient** and its nested classes. Each class section starts with a contents listing of the member functions, followed by the alphabetic presentation of the member functions.

This chapter also describes the **CvSVCAddress** class, which is used to create SVC addresses. The class enforces the basic syntax rules of the seven different formats of SVC addresses.

The master contents list of all the C++ functions follows.

Master List of Functions

CvClient

Session Control

Establishing a Session 3-6
setModifyType 3-8
Terminating a Session 3-9

Operational Functions

addMember (asynchronous) 3-11
addMember (synchronous) 3-13
addObject (asynchronous) 3-15
addObject (synchronous) 3-17
deleteMember (asynchronous) 3-19
deleteMember (synchronous) 3-21
deleteObject (asynchronous) 3-23
deleteObject (synchronous) 3-25
getErrorMsg 3-27
getObject (asynchronous) 3-28
getObject (synchronous) 3-30
getResponseArgs 3-32
getResponseErrorIndex 3-33
listContainedObjects (asynchronous; list all) 3-34
listContainedObjects (synchronous; list all) 3-38
listContainedObjects (asynchronous; list) 3-42
listContainedObjects (synchronous: list) 3-48
modifyObject (asynchronous) 3-54
modifyObject (synchronous) 3-56
nextObject 3-58

CvClient::Callback

CvClient::Callback (Client Callback Handler class) 3-61
getSelectInfo 3-63
processEvents 3-65
timeout 3-67

CvClient::ObjectId

CvClient::ObjectId 3-69
getCard 3-71
getChannel 3-72
getDlci 3-73
getE164Address 3-74
getLPort 3-76
getName 3-77
getName2 3-78
getNetwork 3-79
getPMPSpvcLeafNo 3-80
getPPort 3-81
getStartVpi 3-82
getSVCAddress 3-83
getSwitch 3-85
getTrafficShaperId 3-86
getVci 3-87
getVpi 3-88
set 3-89
setAtmInfo 3-92
setCard 3-94
setChannel 3-96
setDlci 3-98
setE164Address 3-100
setLPort 3-102
setName 3-104
setName2 3-106
setNameAndType 3-108
setNetwork 3-110
setPMPSpvcLeafNo 3-112
setPPort 3-114
setSVCAddressAndType 3-116
setSwitch 3-118
setTrafficShaper 3-120
setType 3-122
setVirtualUniLPort 3-123
type 3-125

C++ Functions

CvClient::Args

CvClient::Args 3-127
count 3-130
errorIndex 3-131
exists 3-132
getStatus 3-133
getType 3-134
idAt 3-135
setNull 3-136
statusAt 3-137
valueAt 3-138

CvClient::IpAddr

CvClient::IpAddr 3-140

CvSVCAddr

CvSVCAddr 3-143

Session Control

Establishing a Session 3-6
setModifyType 3-8
Terminating a Session 3-9

Establishing a Session

Purpose

Create a **CvClient** class object to establish a session between the Provisioning client and the Provisioning Server. If the session is to be global or static, use the null constructor to declare the object:

Syntax

CvClient::CvClient ( )

Then, in your program, use the **CvClient::open** member function to pass **CvClient** arguments that provide session context:

Syntax

int CvClient::open (CvClient::IpAddr *Host*,
CvIpPort *RemotePort*,
**const char ***_UserName_,
const char ***_Password_ **);

Description

Host is the **CvClient::IpAddr** class that specifies the IP Address of the host on which the server is running. Specify the address as a string or an unsigned long. To use a local host, set this value to an unsigned long.

RemotePort identifies the remote port to be used for communication between the client and server. To use a default port, set this value to **CvDefaultServerPort**, which sets the port to 4001.

UserName is a pointer to the name of the user starting the session. This value is the same as that required by CascadeView.

Password is a pointer to the password required to start the session. This value is the same as that required by CascadeView.

Notes

You can have multiple sessions open at a time.

UserName and *Password* are optional parameters. By default, the Provisioning Server accepts requests from clients without requiring authorization. To implement a security feature that authenticates user logins, set the environment variable CV_PSRV_USE_LOGINS (refer to Chapter 2 of the *NavisXtend Provisioning Server User's Guide* for details). When issuing this function, pass in the character strings required by CascadeView for the *UserName* and *Password* arguments.

The UserName and Password character strings are sent over the network as nonencrypted text.

The API also provides the following constructor to pass **CvClient** arguments that provide session context:

Syntax

CvClient::CvClient (CvClient::IpAddr *Host***,**
CvIpPort *RemotePort***,**
const char ***_UserName_,**
const char ***_Password_ **);

The parameters are the same as those of the **CvClient::open** member function described earlier in this section.

Keep in mind that this constructor may fail if you use it to establish a static or global session. For such a session, use the null constructor and the **CvClient::open** member function as described earlier in this section.

Prerequisites

None.

Post-processing

None.

See Also

Terminating a Session.

setModifyType

Purpose

Specifies whether updates are made to the network component and database, or to the database only.

Syntax

CvModifyType CvClient::setModifyType(
CvModifyType *ModifyType* **);**

Description

setModifyType specifies whether updates are made to the network component and database, or to the database only. The function returns the update setting that was in effect previously.

ModifyType indicates the update method, as follows:

modifyNormal — Sends updates to both the network component and the database. The database is updated only if the complete update suceeds.

modifyDbOnly — Sends updates to the database only.

modifyDbOnlySetOutSync — Sends updates to the database only and sets a flag indicating that the object is out of synchronization in the database.

Notes

Once you issue this function, the update setting remains in effect until either the session is terminated or the function is issued with another setting.

To determine if the setting was changed since your last session, compare *ModifyType* to the **CvModifyType** returned by the function.

Prerequisites

You must establish a session with the Provisioning Server.

Post-processing

None.

Purpose

Use one of the following methods to terminate a session between the Provisioning client and the Provisioning Server:

- Issue **CvClient::close**. This function does not delete the **CvClient** class object, but does terminate the session with the Provisioning server.
- Use the **CvClient** destructor.

Syntax

CvClient::close ( )

Prerequisites

You must establish a session with the Provisioning Server.

Post-processing

None.

See Also

Establishing a Session.

Operational Functions

addMember (asynchronous) 3-11
addMember (synchronous) 3-13
addObject (asynchronous) 3-15
addObject (synchronous) 3-17
deleteMember (asynchronous) 3-19
deleteMember (synchronous) 3-21
deleteObject (asynchronous) 3-23
deleteObject (synchronous) 3-25
getErrorMsg 3-27
getObject (asynchronous) 3-28
getObject (synchronous) 3-30
getResponseArgs 3-32
getResponseErrorIndex 3-33
listContainedObjects (asynchronous; list all) 3-34
listContainedObjects (synchronous; list all) 3-38
listContainedObjects (asynchronous; list) 3-42
listContainedObjects (synchronous: list) 3-48
modifyObject (asynchronous) 3-54
modifyObject (synchronous) 3-56
nextObject 3-58

Purpose

Adds a member to a list of objects.

Syntax

CvReqId CvClient::addMember(
const CvClient::CvObjectId& *Container*,
const CvClient::CvObjectId& *Member*,
Callback ***_Callback_ **);

Description

addMember is an asynchronous function that adds an address to a screen or netwide group address. Upon completion of the function, the address represented by the second object ID argument is added to the object specified by the first object ID argument.

Container is the **CvClient::CvObjectId** class that specifies the container object, which is the screen or netwide group address.

Member is the **CvClient::CvObjectId** class that specifies the member object, which is the address to become a member of the screen or netwide group address.

Callback is a pointer to the callback function of the class **CvClient::Callback**.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

When you specify the object CVT_SmdsGroupScreen as the container object, the member to be added must be either a CVT_SmdsAlienGroupAddress or a CVT_SmdsSwitchGroupAddress.

When you specify the object CVT_SmdsIndividualScreen as the container object, the member to be added must be either a CVT_SmdsLocalIndividualAddress or a CVT_SmdsAlienIndividualAddress.

When you specify the object CVT_SmdsNetwideGroupAddress as the container object, the member to be added must be a CVT_SmdsLocalIndividualAddress.

Prerequisites

You must establish a session with the Provisioning Server.

You must use valid object IDs for each of the following:

- The object identifier represented by *Container*.
- The object identifier represented by *Member*.

Post-processing

The application must return to the **select** loop. Invoke **CvClient::processEvents** after activity is found on the file descriptor.

When issuing multiple requests, use the returned **CvReqId** to verify or look up the original request. Compare this **CvReqId** to the **CvReqId** of the callback function to associate a response with its original request.

See Also

addMember (synchronous).

Purpose

Adds a member to a list of objects.

Syntax

void CvClient::addMember(
const CvClient::CvObjectId& *Container*,
const CvClient::CvObjectId& *Member*,
CvClient::ApiStatus* *Status* **);**

Description

addMember is a synchronous function that adds an address to a screen or netwide group address. Upon completion of the function, the address represented by the second object ID argument is added to the object specified by the first object ID argument.

Container is the **CvClient::CvObjectId** class that specifies the container object, which is the screen or netwide group address.

Member is the **CvClient::CvObjectId** class that specifies the member object, which is the address to become a member of the screen or netwide group address.

Status, an output parameter, is a pointer to the **CvClient::ApiStatus** class that contains the status returned by the function.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

When you specify the object CVT_SmdsGroupScreen as the container object, the member to be added must be either a CVT_SmdsAlienGroupAddress or a CVT_SmdsSwitchGroupAddress.

When you specify the object CVT_SmdsIndividualScreen as the container object, the member to be added must be either a CVT_SmdsLocalIndividualAddress or a CVT_SmdsAlienIndividualAddress.

When you specify the object CVT_SmdsNetwideGroupAddress as the container object, the member to be added must be a CVT_SmdsLocalIndividualAddress.

Prerequisites

You must establish a session with the Provisioning Server.

Make sure to use valid object IDs for each of the following:

- The object identifier represented by *Container*.
- The object identifier represented by *Member*.

Post-processing

None.

See Also

addMember (asynchronous).

Purpose

Creates an object in the database and (optionally) in the switch.

Syntax

CvReqId CvClient::addObject(
const CvClient::ObjectId &*ObjId***,**
CvClient::Callback ***Callback*,**
const CvClient::Args& *Args* **);**

Description

addObject is an asynchronous function that creates an object in the database and (optionally) in the switch. The attributes specified by the function are used to initialize the object.

ObjId is the **CvClient::CvObjectId** class that specifies the object to be added.

Callback is a pointer to the callback function of the class **CvClient::Callback**.

Args is the **CvClient::Args** class that specifies the attributes for the object to be added. The API does not modify or delete this argument. Specify only those attributes appropriate for the object type (refer to the *NavisXtend Provisioning Server Object Attribute Definitions*). You can specify any attributes except those with the Read-Only access restriction.

If you omit a required parameter, the function response will have the status **cveArgumentNotPresent**. Check the argument list returned in the response. Any attribute that is missing a valid value is a required attribute that you omitted.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

To create a card or PPort, use a Modify function (asynchronous or synchronous version of **modifyObject**). The CascadeView database automatically populates each switch with cards of type “empty”. Use the Modify function to modify the card’s type from “empty” to a specified type. Likewise, once a card has been configured, CascadeView automatically populates the card with all necessary physical ports. Use the Modify function to change the PPort specifications. In the case of the channelized DS3 card, once the card has been configured, CascadeView automatically populates the card with all necessary channels. Use the Modify function to change the channel specifications.

Prerequisites

You must establish a session with the Provisioning Server.

Create each of the following:

- A class **CvClient::ObjectId** for the object to be added.
- A class **CvClient::Args** for the list of attributes for the object to be added.

Post-processing

The application must return to the **select** loop. Invoke **CvClient::processEvents** after activity is found on the file descriptor.

When issuing multiple requests, use the returned **CvReqId** to verify or look up the original request. Compare this **CvReqId** to the **CvReqId** of the callback function to associate a response with its original request.

See Also

addObject (synchronous), **modifyObject** (asynchronous), and **modifyObject** (synchronous).

Purpose

Creates an object in the database and (optionally) in the switch.

Syntax

void CvClient::addObject(
const CvClient::ObjectId &*ObjId***,**
const CvClient::Args& *Args***,**
CvClient::ApiStatus ***Status* **);

Description

addObject is a synchronous function that creates an object in the database and (optionally) in the switch. The attributes specified by the function are used to initialize the object.

ObjId is the **CvClient::ObjectId** class that specifies the object to be added.

Args is the **CvClient::Args** class that specifies the attributes for the object to be added. The API does not modify or delete this argument. Specify only those attributes appropriate for the object type (refer to the *NavisXtend Provisioning Server Object Attribute Definitions*). You can specify any attributes except those with the Read-Only access restriction.

Status, an output parameter, is a pointer to the **CvClient::ApiStatus** class that contains the status returned by the function.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

To create a card or PPort, use a Modify function (asynchronous or synchronous version of **modifyObject**). The CascadeView database automatically populates each switch with cards of type "empty". Use the Modify function to modify the card's type from "empty" to a specified type. Likewise, once a card has been configured, CascadeView automatically populates the

card with all necessary physical ports. Use the Modify function to change the PPort specifications. In the case of the channelized DS3 card, once the card has been configured, CascadeView automatically populates the card with all necessary channels. Use the Modify function to change the channel specifications.

If you omit a required parameter, the function response will have the status **cveArgumentNotPresent**. Issue **CvGetResponseArgs** to obtain the argument list returned by the function. Any attribute that is missing a valid value is a required attribute that you omitted.

Prerequisites

You must establish a session with the Provisioning Server.

Create each of the following:

- A class **CvClient::ObjectId** for the object to be added.
- A class **CvClient::Args** for the list of attributes for the object to be added.

Post-processing

If the response has the status **cveArgumentNotPresent**, issue **CvGetResponseArgs** to obtain the argument list returned by the function. Any attribute that is missing a valid value is a required attribute that you omitted.

See Also

addObject (asynchronous), **modifyObject** (asynchronous), and **modifyObject** (synchronous).

Purpose

Removes a member from a list of objects.

Syntax

CvReqId CvClient::deleteMember(
const CvClient::CvObjectId& *Container*,
const CvClient::CvObjectId& *Member*,
Callback ***_Callback_ **);

Description

deleteMember is an asynchronous function that removes an address from a screen or netwide group address. Upon completion of the function, the address represented by the second object ID argument is removed from the object specified by the first object ID argument.

Container is the **CvClient::CvObjectId** class that specifies the container object, which is the screen or netwide group address.

Member is the **CvClient::CvObjectId** class that specifies the address to be removed.

Callback is a pointer to the callback function of the class **CvClient::Callback**.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

When you specify the object CVT_SmdsGroupScreen as the container object, the member to be removed must be either a CVT_SmdsAlienGroupAddress or a CVT_SmdsSwitchGroupAddress.

When you specify the object CVT_SmdsIndividualScreen as the container object, the member to be removed must be either a CVT_SmdsLocalIndividualAddress or a CVT_SmdsAlienIndividualAddress.

When you specify the object CVT_SmdsNetwideGroupAddress as the container object, the member to be removed must be a CVT_SmdsLocalIndividualAddress.

Prerequisites

You must establish a session with the Provisioning Server.

Create a class **CvClient::ObjectId** for each of the following:

- The object identifier represented by *Container*.
- The object identifier represented by *Member*.

Post-processing

The application must return to the **select** loop. Invoke **CvClient::processEvents** after activity is found on the file descriptor.

When issuing multiple requests, use the returned **CvReqId** to verify or look up the original request. Compare this **CvReqId** to the **CvReqId** of the callback function to associate a response with its original request.

See Also

deleteMember (synchronous).

Purpose

Removes a member from a list of objects.

Syntax

void CvClient::deleteMember(
const CvClient::CvObjectId& *Container*,
const CvClient::CvObjectId& *Member*,
CvClient::ApiStatus * *Status* **);**

Description

deleteMember is a synchronous function that removes an address from a screen or group address. Upon completion of the function, the address represented by the second object ID argument is removed from the object specified by the first object ID argument.

Container is the **CvClient::CvObjectId** class that specifies the container object, which is the screen or netwide group address.

Member is the **CvClient::CvObjectId** class that specifies the address to be removed.

Status, an output parameter, is a pointer to the **CvClient::ApiStatus** class that contains the status returned by the function.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

When you specify the object CVT_SmdsGroupScreen as the container object, the member to be removed must be either a CVT_SmdsAlienGroupAddress or a CVT_SmdsSwitchGroupAddress.

When you specify the object CVT_SmdsIndividualScreen as the container object, the member to be removed must be either a CVT_SmdsLocalIndividualAddress or a CVT_SmdsAlienIndividualAddress.

When you specify the object CVT_SmdsNetwideGroupAddress as the container object, the member to be removed must be a CVT_SmdsLocalIndividualAddress.

Prerequisites

You must establish a session with the Provisioning Server.

Create a class **CvClient::ObjectId** for each of the following:

- The object identifier represented by *Container*.
- The object identifier represented by *Member*.

Post-processing

None.

See Also

deleteMember (asynchronous).

Purpose

Deletes an object from the database and (optionally) from the switch.

Syntax

CvReqId CvClient::deleteObject(
const CvClient::ObjectId &*ObjId***,**
CvClient::Callback ***Callback* **);

Description

deleteObject is an asynchronous function that deletes an object from the database and (optionally) from the switch.

ObjId is a pointer to the **CvClient::ObjectId** class that specifies the object to be deleted.

Callback is a pointer to the callback function of the class **CvClient::Callback**.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

You only need to delete an SMDS switch group address if the database shows an SMDS switch group address that should not exist.

To remove a card, use a Modify function (asynchronous or synchronous version of **modifyObject**) to modify the card's type to "empty".

You cannot delete some objects until the objects they contain have been deleted. For example, you cannot delete an LPort until all circuits and addresses defined on the LPort have been deleted.

Prerequisites

You must establish a session with the Provisioning Server.

Create a **CvClient::ObjectId** class for the object identifier.

Post-processing

The application must return to the **select** loop. Invoke **CvClient::processEvents** after activity is found on the file descriptor.

When issuing multiple requests, use the returned **CvReqId** to verify or look up the original request. Compare this **CvReqId** to the **CvReqId** of the callback function to associate a response with its original request.

See Also

deleteObject (synchronous), **modifyObject** (asynchronous), and **modifyObject** (synchronous).

Purpose

Deletes an object from the database and (optionally) from the switch.

Syntax

void CvClient::deleteObject(
const CvClient::CvObjectId& *ObjId***,**
CvClient::ApiStatus* *Status* **);**

Description

deleteObject is a synchronous function that deletes an object from the database and (optionally) from the switch.

ObjId is a pointer to the **CvClient::ObjectId** class that specifies the object to be deleted.

Status, an output parameter, is a pointer to the **CvClient::ApiStatus** class that contains the status returned by the function.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

You only need to delete an SMDS switch group address if the database shows an SMDS switch group address that should not exist.

To remove a card, use a Modify function (asynchronous or synchronous version of **modifyObject**) to modify the card's type to "empty".

You cannot delete some objects until the objects they contain have been deleted. For example, you cannot delete an LPort until all circuits and addresses defined on the LPort have been deleted.

Prerequisites

You must establish a session with the Provisioning Server.

Create a **CvClient::ObjectId** class for the object identifier.

Post-processing

None.

See Also

deleteObject (asynchronous), **modifyObject** (asynchronous), and **modifyObject** (asynchronous).

Purpose

Obtains the help string associated with a function's error status code.

Syntax

const char *CvClient::getErrorMsg( );

Description

getErrorMsg obtains the help string associated with a function's error status code. When an operational function returns an error status code, issue this function to obtain a help string that provides details on the error.

Notes

This function returns a null pointer if the last function issued did not return an error.

Prerequisites

You must establish a session with the Provisioning Server.

Use this function in conjunction with another function that returns an error status:

1. Issue the function (such as **CvClient::addObject**).
2. If the function returns an error status, issue **CvClient::getErrorMsg** to obtain the help string associated with the status code.

Post-processing

None.

See Also

errorIndex and **getResponseErrorIndex**.

getObject (asynchronous)

Purpose

Retrieves the values of specific attributes. The values are retrieved from the database.

Syntax

CvReqId CvClient::getObject(
const CvClient::ObjectId &*ObjId***,**
CvClient::Callback ***Callback*,**
const CvClient::Args &*Args* **);**

Description

getObject is an asynchronous function that retrieves the values of specific attributes.

ObjId is a pointer to the **CvClient::ObjectId** class that specifies the object whose attributes are to be retrieved.

Callback is a pointer to the callback function of the class **CvClient::Callback**.

Args is the **CvClient::Args** class that specifies the attributes to be retrieved. The API does not modify or delete this argument. Specify only those attributes appropriate for the object type (refer to the *NavisXtend Provisioning Server Object Attribute Definitions*). Specify only attribute IDs with null values. Use either:

- The value-less constructor for the **CvClient::Args** class
- The **CvClient::Value::setNull** function

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

Prerequisites

You must establish a session with the Provisioning Server.

Create each of the following:

- A class **CvClient::ObjectId** for the object to be retrieved.
- A class **CvClient::Args** for the list of attributes to be retrieved for the object.

Post-processing

The application must return to the **select** loop. Invoke **CvClient::processEvents** after activity is found on the file descriptor.

When issuing multiple requests, use the returned **CvReqId** to verify or look up the original request. Compare this **CvReqId** to the **CvReqId** of the callback function to associate a response with its original request.

See Also

getObject (synchronous).

getObject (synchronous)

Purpose

Retrieves the values of specific attributes. The values are retrieved from the database.

Syntax

CvClient::Args CvClient::getObject(
const CvClient::ObjectId &*ObjId***,**
const CvClientArgs &*Args***,**
CvClient::ApiStatus* *Status* **);**

Description

getObject is a synchronous function that retrieves the values of specific attributes.

ObjId is a pointer to the **CvClient::ObjectId** class that specifies the object whose attributes are to be retrieved.

Args is the **CvClient::Args** class that specifies the attributes to be retrieved. The API does not modify or delete this argument. Specify only those attributes appropriate for the object type (refer to the *NavisXtend Provisioning Server Object Attribute Definitions*). Specify only attribute IDs with null values. Use either:

- The value-less constructor for the **CvClient::Args** class
- The **CvClient::Value::setNull** function

Status, an output parameter, is a pointer to the **CvClient::ApiStatus** class that contains the status returned by the function.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

Prerequisites

You must establish a session with the Provisioning Server.

Create each of the following:

- A class **CvClient::ObjectId** for the object to be retrieved.
- A class **CvClient::Args** for the list of attributes to be retrieved for the object.

Post-processing None.

See Also **getObject** (asynchronous).

getResponseArgs

Purpose

Reads the argument list returned by a synchronous function.

Syntax

const CvClient::Args& CvClient::getResponseArgs( );

Description

getResponseArgs reads the argument list returned by a synchronous function. Use this function to determine if you omitted any required parameters when you issued the synchronous operational function. The returned argument list indicates which attributes have values and which do not. Any attribute that is missing a value is a required attribute that you omitted.

Prerequisites

You must establish a session with the Provisioning Server.

Use this function in conjunction with a synchronous operational function to determine if a required attribute was omitted:

1. Issue the function (such as **CvClient::addObject**).
2. If the response has the status **cveArgumentNotPresent**, issue **CvClient::getResponseArgs** to obtain the argument list returned by the function. Any attribute that is missing a value is a required attribute that you omitted.

Post-processing

None.

See Also

errorIndex.

Purpose Returns the error index of the response for a synchronous request.

Syntax **const CvClient::getResponseErrorIndex( );**

Description **getResponseErrorIndex** returns the one-based index to the first parameter that has an error in the response of a synchronous request.

Notes The function's returned value is one-based. The value 0 indicates a non-error status.

Prerequisites You must establish a session with the Provisioning Server.

Use this function in conjunction with a synchronous operational function to determine if an error occurred:

1. Issue the function (such as **CvClient::addObject**).
2. Once the response returns, issue **CvClient::getResponseErrorIndex** to obtain the one-based index to the error of the response.

Post-processing None.

See Also **errorIndex**.

listContainedObjects (asynchronous; list all)

Purpose

Retrieves a list of objects of any type that are immediate children of a specified object.

Syntax

CvReqId CvClient::listContainedObjects(
const CvClient::ObjectId &*ObjId***,**
CvClient::Callback* *Callback* **);**

Description

listContainedObjects is an asynchronous function that retrieves a list of objects of any type that are immediate children of a specified object. Each object is returned in a separate response.

ObjId is the **CvClient::ObjectId** class that specifies the parent object. The parent object is always the immediate parent of the contained objects (such as a PPort that is a parent of multiple LPorts).

Callback is a pointer to the callback function of the class **CvClient::Callback**.

Notes

You can issue the function on either:

Network level — The function retrieves a list of objects on a network, including all subnets. To issue the function on a network level, specify an IP address, such as 128.100.0.0.

Subnet level — The function retrieves a list of objects on a particular subnet. To issue the function on a subnet level, specify an IP address with a subnet number, such as 128.100.111.0.

Table 3-1 lists the valid parent and child object types you can specify with this function.

Table 3-1. Valid Parent and Child Object Types

Parent Object Type	Child Object Types
Network	Switch VPN Customer SmdsCountryCode SmdsNetwideGroupAddress SvcSecScn SvcCUG SvcCUGMbrRule TrafficDesc NetCac ServiceName
Customer	LPort Circuit
VPN	LPort Circuit
SvcCUGMbrRule	SvcCUGMbr
SvcCUG	SvcCUGMbr
Switch	Card SvcNodePrefix SmdsSwitchGroupAddress SmdsAddressPrefix SmdsAlienIndividualAddress SmdsAlienGroupAddress
Card	PPort
PPort	LPort Channel PFdl (8-port ATM T1 card only) PerformanceMonitor Aps (1-port OC-12c/STM-4 card only)
Channel (valid for channelized DS3 card only)	LPort

Table 3-1. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
LPort	Circuit SvcPrefix SvcAddress SvcUserPart SvcConfig SmdsLocalIndividualAddress SmdsIndividualScreen SmdsGroupScreen PMPCktRoot AssignedSvcSecScn SvcSecScnActParam Spvc PMPSpvcRoot
SmdsIndividualScreen	SmdsLocalIndividualAddress SmdsAlienIndividualAddress
SmdsGroupScreen	SmdsSwitchGroupAddress SmdsAlienGroupAddress
SmdsNetwideGroupAddress	SmdsSwitchGroupAddress SmdsLocalIndividualAddress
PMPCktRoot	PMPCktLeaf
PMPSpvcRoot	PMPSpvcLeaf
ServiceName	Circuit

Prerequisites

You must establish a session with the Provisioning Server.

Create a class **CvClient::ObjectId** for the object identifier.

Post-processing

The application must return to the **select** loop. Invoke **CvClient::processEvents** after activity is found on the file descriptor.

This function makes multiple callbacks to the callback handler, each representing an object contained by the parent:

- In the **CvClient::Callback** function, the *ObjId* parameter represents the listed object.
- The last call to the callback handler returns the status **cveListDone**. The call does not include a valid object.
- A call to the callback handler that returns a status other than **cveNoError** or **cveListDone** indicates that an error has occurred. No more responses will be made for this request.

When issuing multiple requests, use the returned **CvReqId** to verify or look up the original request. Compare this **CvReqId** to the **CvReqId** of the callback function to associate a response with its original request.

See Also

listContainedObjects (asynchronous; list), **listContainedObjects** (synchronous; list), and **listContainedObjects** (synchronous; list all).

listContainedObjects (synchronous; list all)

Purpose

Retrieves a list of objects of any type that are immediate children of a specified object.

Syntax

CvListHandle CvClient::listContainedObjects(
const CvClient::CvObjectId& *ObjId***,**
CvClient::ApiStatus* *Status* **);**

Description

listContainedObjects is a synchronous function that retrieves a list of objects of any type that are immediate children of a specified object.

ObjId is the **CvClient::ObjectId** class that specifies the parent object. The parent object is always the immediate parent of the contained objects (such as a PPort that is a parent of multiple LPorts).

Status, an output parameter, is a pointer to the **CvClient::ApiStatus** class that contains the status returned by the function.

Notes

You can issue the function on either:

Network level — The function retrieves a list of objects on a network, including all subnets. To issue the function on a network level, specify an IP address, such as 128.100.0.0.

Subnet level — The function retrieves a list of objects on a particular subnet. To issue the function on a subnet level, specify an IP address with a subnet number, such as 128.100.111.0.

Table 3-2 lists the valid parent and child object types you can specify with this function.

.

Table 3-2. Valid Parent and Child Object Types

Parent Object Type	Child Object Types
Network	Switch VPN Customer SmdsCountryCode SmdsNetwideGroupAddress SvcSecScn SvcCUG SvcCUGMbrRule TrafficDesc NetCac ServiceName
Customer	LPort Circuit
VPN	LPort Circuit
SvcCUGMbrRule	SvcCUGMbr
SvcCUG	SvcCUGMbr
Switch	Card SvcNodePrefix SmdsSwitchGroupAddress SmdsAddressPrefix SmdsAlienIndividualAddress SmdsAlienGroupAddress
Card	PPort
PPort	LPort Channel PFdl (8-port ATM T1 card only) PerformanceMonitor Aps (1-port OC-12c/STM-4 card only)
Channel (valid for channelized DS3 card only)	LPort

Table 3-2. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
LPort	Circuit SvcPrefix SvcAddress SvcUserPart SvcConfig SmdsLocalIndividualAddress SmdsIndividualScreen SmdsGroupScreen PMPCktRoot AssignedSvcSecScn SvcSecScnActParam Spvc PMPSpvcRoot
SmdsIndividualScreen	SmdsLocalIndividualAddress SmdsAlienIndividualAddress
SmdsGroupScreen	SmdsSwitchGroupAddress SmdsAlienGroupAddress
SmdsNetwideGroupAddress	SmdsSwitchGroupAddress SmdsLocalIndividualAddress
PMPCktRoot	PMPCktLeaf
PMPSpvcRoot	PMPSpvcLeaf
ServiceName	Circuit

Prerequisites

You must establish a session with the Provisioning Server.

Create a class **CvClient::ObjectId** for the object identifier.

Post-processing

Once you receive a CvListHandle, issue **CvClient::nextObject** to retrieve the next object. Pass in the CvListHandle value returned by the previous **listContainedObjects**:

- When a call to **CvClient::nextObject** returns the status **cveNoError**, continue issuing **CvClient::nextObject**.
- When a call to **CvClient::nextObject** returns the status **cveListDone**, stop issuing **CvClient::nextObject**. This last call does not include a valid object.
- When a call to **CvClient::nextObject** returns a status other than **cveNoError** or **cveListDone**, an error has occurred.

Keep in mind that the CvListHandle has a limited lifetime. If more than 10 minutes elapse between requests with the same handle, the next request may return with an error status.

See Also

listContainedObjects (asynchronous; list all), **listContainedObjects** (asynchronous; list), and **listContainedObjects** (synchronous; list).

listContainedObjects (asynchronous; list)

Purpose

Retrieves a list of objects of a specified object type that are contained by another object.

Syntax

CvReqId CvClient::listContainedObjects(
const CvClient::ObjectId& *ObjId***,**
CvObjectType *ObjType***,**
CvClient::Callback* *Callback***,**
const CvClient::Args &*Args* **);**

Description

listContainedObjects is an asynchronous function that retrieves a list of objects contained by another object. The returned objects are those that match a specified object type. Each object is returned in a separate response.

ObjId is the **CvClient::ObjectId** class that specifies the parent object. The parent object can be the immediate parent of the contained objects (such as a PPort that is a parent of multiple LPorts) or the parent object can be positioned higher in the containment hierarchy (such as a switch that is a parent of multiple LPorts).

ObjType is the enumerated value that specifies the type of the objects to be retrieved.

Callback is a pointer to the callback function of the class **CvClient::Callback**.

Args is the **CvClient::Args** class that specifies the attributes to be retrieved. The API does not modify or delete this argument. If you want all attributes to be retrieved, pass in a null pointer for this argument.

Notes

You can issue the function on either:

Network level — The function retrieves a list of objects on a network, including all subnets. To issue the function on a network level, specify an IP address, such as 128.100.0.0.

Subnet level — The function retrieves a list of objects on a particular subnet. To issue the function on a subnet level, specify an IP address with a subnet number, such as 128.100.111.0.

Table 3-3 lists the valid parent and child object types you can specify with this function.

Table 3-3. Valid Parent and Child Object Types

Parent Object Type	Child Object Types
Network	Circuit Customer NetCac PMPCktLeaf PMPCktRoot PMPSpvcRoot SmdsAddressPrefix SmdsCountryCode SmdsLocalIndividualAddress SmdsNetwideGroupAddress SvcCUG SvcCUGMbrRule SvcSecScn Switch TrafficDesc ServiceName VPN
Customer	LPort Circuit
VPN	LPort Circuit
SvcCUGMbrRule	SvcCUGMbr

Table 3-3. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
SvcCUG	SvcCUGMbr
Switch	Aps Card Channel Circuit LPort PerformanceMonitor PFdl PMPSpvcRoot PPort Spvc SmdsAddressPrefix SmdsAlienGroupAddress SmdsAlienIndividualAddress SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress SmdsSwitchGroupAddress SvcAddress SvcConfig SvcNodePrefix SvcPrefix SvcUserPart

Table 3-3. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
Card	Aps Channel Circuit LPort PerformanceMonitor PFdl PMPSpvcRoot PPort Spvc SmdsAlienGroupAddress SmdsAlienIndividualAddress SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress SmdsSwitchGroupAddress SvcConfig SvcNodePrefix SvcUserPart
PPort	Aps Channel Circuit LPort PerformanceMonitor PFdl (8-port ATM T1 card only)
Channel (valid for channelized DS3 card only)	LPort

Table 3-3. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
LPort	AssignedSvcSecScn Circuit PMPCktRoot PMPSpvcRoot SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress Spvc SvcAddress SvcConfig SvcPrefix SvcUserPart SvcSecScnActParam
SmdsIndividualScreen	SmdsLocalIndividualAddress SmdsAlienIndividualAddress
SmdsGroupScreen	SmdsSwitchGroupAddress SmdsAlienGroupAddress
SmdsNetwideGroupAddress	SmdsSwitchGroupAddress SmdsLocalIndividualAddress
PMPCktRoot	PMPCktLeaf
PMPSpvcRoot	PMPSpvcLeaf

Prerequisites

You must establish a session with the Provisioning Server.

Create a class **CvClient::ObjectId** for the object identifier.

To specify which attributes you want retrieved for the object, you must build the argument list. To do so, create a class **CvClient::Args** for the argument list.

Post-processing

The application must return to the **select** loop. Invoke **CvClient::processEvents** after activity is found on the file descriptor.

This function makes multiple callbacks to the callback handler, each representing an object contained by the parent:

- In the **CvClient::Callback** function, the *ObjId* parameter represents the listed object. The *Args* parameter represents the object's attributes.
- The last call to the callback handler returns the status **cveListDone**. The call does not include a valid object.
- A call to the callback handler that returns a status other than **cveNoError** or **cveListDone** indicates that an error has occurred. No more responses will be made for this request.

When issuing multiple requests, use the returned **CvReqId** to verify or look up the original request. Compare this **CvReqId** to the **CvReqId** of the callback function to associate a response with its original request.

See Also

listContainedObjects (asynchronous; list all), **listContainedObjects** (synchronous; list all), and **listContainedObjects** (synchronous; list).

listContainedObjects (synchronous: list)

Purpose

Retrieves a list of objects of a specified object type that are contained by another object.

Syntax

CvListHandle CvClient::listContainedObjects(
const CvClient::ObjectId& *ObjId***,**
CvObjectType *ObjType***,**
const CvClient::Args &*Args***,**
CvClient::ApiStatus* *Status* **);**

Description

listContainedObjects is a synchronous function that retrieves a list of objects contained by another object. The returned objects are those that match a specified object type.

ObjId is the **CvClient::ObjectId** class that specifies the parent object. The parent object can be the immediate parent of the contained objects (such as a PPort that is a parent of multiple LPorts) or the parent object can be positioned higher in the containment hierarchy (such as a switch that is a parent of multiple LPorts).

ObjType is the enumerated value that specifies the type of the objects to be retrieved.

Status, an output parameter, is a pointer to the **CvClient::ApiStatus** class that contains the status returned by the function.

Args is the **CvClient::Args** class that specifies the attributes to be retrieved. The API does not modify or delete this argument. If you want all attributes to be retrieved, pass in a null pointer for this argument.

Notes

You can issue the function on either:

Network level — The function retrieves a list of objects on a network, including all subnets. To issue the function on a network level, specify an IP address, such as 128.100.0.0.

Subnet level — The function retrieves a list of objects on a particular subnet. To issue the function on a subnet level, specify an IP address with a subnet number, such as 128.100.111.0.

Table 3-4 lists the valid parent and child object types you can specify with this function.

Table 3-4. Valid Parent and Child Object Types

Parent Object Type	Child Object Types
Network	Circuit Customer NetCac PMPCktLeaf PMPCktRoot PMPSpvcRoot SmdsAddressPrefix SmdsCountryCode SmdsLocalIndividualAddress SmdsNetwideGroupAddress SvcCUG SvcCUGMbrRule SvcSecScn Switch TrafficDesc ServiceName VPN
Customer	LPort Circuit
VPN	LPort Circuit
SvcCUGMbrRule	SvcCUGMbr

Table 3-4. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
SvcCUG	SvcCUGMbr
Switch	Aps Card Channel Circuit LPort PerformanceMonitor PFdl PMPSpvcRoot PPort Spvc SmdsAddressPrefix SmdsAlienGroupAddress SmdsAlienIndividualAddress SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress SmdsSwitchGroupAddress SvcAddress SvcConfig SvcNodePrefix SvcPrefix SvcUserPart

Table 3-4. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
Card	Aps Channel Circuit LPort PerformanceMonitor PFdl PMPSpvcRoot PPort Spvc SmdsAlienGroupAddress SmdsAlienIndividualAddress SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress SmdsSwitchGroupAddress SvcConfig SvcNodePrefix SvcUserPart
PPort	Aps Channel Circuit LPort PerformanceMonitor PFdl (8-port ATM T1 card only)
Channel (valid for channelized DS3 card only)	LPort

Table 3-4. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
LPort	AssignedSvcSecScn Circuit PMPCktRoot PMPSpvcRoot SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress Spvc SvcAddress SvcConfig SvcPrefix SvcUserPart SvcSecScnActParam
SmdsIndividualScreen	SmdsLocalIndividualAddress SmdsAlienIndividualAddress
SmdsGroupScreen	SmdsSwitchGroupAddress SmdsAlienGroupAddress
SmdsNetwideGroupAddress	SmdsSwitchGroupAddress SmdsLocalIndividualAddress
PMPCktRoot	PMPCktLeaf
PMPSpvcRoot	PMPSpvcLeaf

Prerequisites

You must establish a session with the Provisioning Server.

Create a class **CvClient::ObjectId** for the object identifier.

To specify which attributes you want retrieved for the object, you must build the argument list. To do so, create a class **CvClient::Args** for the argument list.

Post-processing

Once you receive a CvListHandle, issue **CvClient::nextObject** to retrieve the next object. Pass in the list handle value returned by the previous **CvClient::listContainedObjects**:

- When a call to **CvClient::nextObject** returns the status **cveNoError**, continue issuing **CvClient::nextObject**.
- When a call to **CvClient::nextObject** returns the status **cveListDone**, stop issuing **CvClient::nextObject**. This last call does not include a valid object.
- When a call to **CvClient::nextObject** returns a status other than **cveNoError** or **cveListDone**, an error has occurred.

Keep in mind that the CvListHandle has a limited lifetime. If more than 10 minutes elapse between requests with the same handle, the next request may return with an error status.

See Also

listContainedObjects (asynchronous; list all), **listContainedObjects** (asynchronous; lst), and **listContainedObjects** (synchronous; list all).

modifyObject (asynchronous)

Purpose

Modifies specific attributes of an object in the database and (optionally) in the switch.

Syntax

CvReqId CvClient::modifyObject(
const CvClient::ObjectId& *ObjId***,**
CvClient::Callback *Callback***,**
const CvClient::Args& *Args* **);**

Description

modifyObject is an asynchronous function that modifies specific attributes of an object.

ObjId is the **CvClient::ObjectId** class that specifies the object to be modified.

Callback is a pointer to the callback function of the class **CvClient::Callback**.

Args is the **CvClient::Args** class that specifies the attributes to be modified. The API does not modify or delete this argument. Specify only those attributes appropriate for the object type (refer to the *NavisXtend Provisioning Server Object Attribute Definitions*). You can modify any attributes except those with either the Read-Only or Create-Only access restriction.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

Use this function to create a card or PPort. The CascadeView database automatically populates each switch with cards of type "empty". Use this function to modify the card's type from "empty" to a specified type. Likewise, once a card has been configured, CascadeView automatically populates the card with all

necessary physical ports. Use this function to change the PPort specifications. In the case of the channelized DS3 card, once the card has been configured, CascadeView automatically populates the card with all necessary channels. Use this function to change the channel specifications.

Use this function to remove a card. To do so, modify the card type to “empty”.

Prerequisites

You must establish a session with the Provisioning Server.

Create each of the following:

- A class **CvClient::ObjectId** for the object to be modified.
- A class **CvClient::Args** for the list of attributes to be modified for the object.

Post-processing

The application must return to the **select** loop. Invoke **CvClient::processEvents** after activity is found on the file descriptor.

When issuing multiple requests, use the returned **CvReqId** to verify or look up the original request. Compare this **CvReqId** to the **CvReqId** of the callback function to associate a response with its original request.

See Also

modifyObject (synchronous).

modifyObject (synchronous)

Purpose

Modifies specific attributes of an object in the database and (optionally) in the switch.

Syntax

void CvClient::modifyObject(
const CvClient::ObjectId& *ObjId***,**
const CvClient::Args & *Args***,**
const CvClient::ApiStatus* *Status* **);**

Description

modifyObject is a synchronous function that modifies specific attributes of an object.

ObjId is the **CvClient::ObjectId** class that specifies the object to be modified.

Status, an output parameter, is a pointer to the **CvClient::ApiStatus** class that contains the status returned by the function.

Args is the **CvClient::Args** class that specifies the attributes for the object to be added. Specify only those attributes appropriate for the object type (refer to the *NavisXtend Provisioning Server Object Attribute Definitions*). You can modify any attributes except those with either the Read-Only or Create-Only access restriction.

Notes

For a list of object types that you can specify with this function, refer to "Valid Object Types for Operational Functions" in Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

Use this function to create a card or PPort. The CascadeView database automatically populates each switch with cards of type "empty". Use this function to modify the card's type from "empty" to a specified type. Likewise, once a card has been configured, CascadeView automatically populates the card with all

necessary physical ports. Use this function to change the PPort specifications. In the case of the channelized DS3 card, once the card has been configured, CascadeView automatically populates the card with all necessary channels. Use this function to change the channel specifications.

Use this function to remove a card. To do so, modify the card type to “empty”.

Prerequisites

You must establish a session with the Provisioning Server.

Create a class **CvClient::ObjectId** for the object to be modified.

Create a class **CvClient::Args** for the list of attributes to be modified for the object.

See Also

modifyObject (asynchronous).

nextObject

Purpose

Retrieves the next object in a list.

Syntax

CvClient::Args CvClient::nextObject(
CvListHandle *ListHandle***,**
CvClient::ApiStatus* *Status* **);**

Description

nextObject is a synchronous function that retrieves the next object in a list. When you request a list of objects contained by another object, use this function to retrieve the next object in the list.

ListHandle is the CvListHandle variable returned by a synchronous List Contained function (**listContainedObjects**, synchronous; list all or list version).

Status, an output parameter, is a pointer to the **CvClient::ApiStatus** class that contains the status returned by the function.

Prerequisites

You must establish a session with the Provisioning Server.

Use a List Contained function (**listContainedObjects**, synchronous; list all or list version) to obtain a CvListHandle.

Post-processing

Once you receive a CvListHandle, issue **CvClient::nextObject** to retrieve the next object:

- When a call to **CvClient::nextObject** returns the status **cveNoError**, continue issuing **CvClient::nextObject**.
- When a call to **CvClient::nextObject** returns the status **cveListDone**, stop issuing **CvClient::nextObject**. This last call does not include a valid object.
- When a call to **CvClient::nextObject** returns a status other than **cveNoError** or **cveListDone**, an error has occurred.

Keep in mind that the CvListHandle has a limited lifetime. If more than 10 minutes elapse between requests with the same handle, the next request may return with an error status.

See Also

listContainedObjects (synchronous list all) and **listContainedObjects** (synchronous list).

Select Loop Processing

CvClient::Callback (Client Callback Handler class) 3-61
getSelectInfo 3-63
processEvents 3-65
timeout 3-67

General Notes

To support asynchronous functions, callbacks are made to a virtual member function of the class **CvClient::Callback**. You must derive from this class to receive callbacks.

When you issue an asynchronous request, pass in a pointer to a class object derived from **CvClient::Callback**, which implements the virtual function **response**.

The other **select** loop processing functions are member functions of **CvClient**.

Purpose

Is the callback handler class that delivers a response to an asynchronous request. You should derive a class that implements the virtual member function:

Syntax

```
class MyHandler::public CvClient::Callback {
void response (
        const CvClient::ObjectId &ObjId,
        CvClient::ApiStatus Status,
        CvReqId RequestId,
        const CvClient::Args *Args );
        }
```

Description

CvClient::Callback is the callback handler class that delivers a response to an asynchronous request. The API calls the virtual member function **response** that delivers a response to an asynchronous request.

ObjId is the **CvClient::ObjectId** class that specifies the object.

Status is the **CvClient::ApiStatus** class that contains the status returned by the function.

RequestId is the request ID associated with the asynchronous request.

Args is the **CvClient::Args** class that contains the attributes for the response. The client must not modify or delete this argument.

Notes

When you issue an asynchronous request, pass in a pointer to the class object derived from **CvClient::Callback**. The API calls the virtual member function **response**.

Prerequisites

None.

Post-processing

Upon return, the function returns back into the API library and then into the application's **select** loop.

The application must return to the **select** loop to receive more callbacks.

Purpose

Obtains information needed to pass to a **select** system call after the completion of an asynchronous function.

Syntax

void CvClient::getSelectInfo(
int *WaitTimeMs*,
**int ***Numfds*,
struct fd_set **Fdset*,
struct timeval **Timeout* **);**

Description

getSelectInfo obtains information needed to pass to a **select** system call after the completion of an asynchronous function. You should issue this call before each call to **select**. The function provides information that **select** uses to notify your application of pending messages from the Provisioning Server.

WaitTimeMs is the maximum amount of time (in milliseconds) that **select** should wait for an event. Use this timeout argument as follows:

0 – For applications designed to perform polling in the **select** loop.

> 0 – For applications designed to time out. This value is useful for complex applications that use other kinds of timeouts. This timeout represents the time to wait for an event, not the time to wait for a specific response.

Numfd and *Fdset* indicate the file descriptors used by the application. *Numfd* is the number of file descriptors indicated by the fd_set structure. *Fdset* is a pointer to the fd_set structure that contains the set of file descriptors to be passed to **select**. Either initialize this argument as empty or set to a value to indicate the application's own file descriptors.

Timeout, an output parameter, is a pointer to the timeval structure. When the function returns, timeval is set to either of the following values, whichever is smaller:

- *WaitTimeMs*
- Preferred timeout of the API

Notes

This function augments the *Numfd* and *Fdset* arguments with its own file descriptor information and sets the timeval structure to the smallest requested timeout. Your application should pass this information to the **select** call.

Prerequisites

None.

Post-processing

Use this function in conjunction with a **select** loop:

1. Issue this function to obtain information for the **select** call.
2. If **select** returns 0, issue **CvClient::timeout** to determine whether a timeout occurred from the server.
3. When **select** returns a non-zero value, issue **CvClient::processEvents** to receive and process a message from the server.

See Also

processEvents and **timeout**.

Purpose

Processes activity on file descriptors in order to receive responses to an asynchronous function.

Syntax

int CvClient::processEvents(
long *TimeOutInMs*,
int *ReturnOnError* **);**

Description

processEvents processes activity on file descriptors in order to receive responses to an asynchronous function. The function returns the value from the last **select** call.

TimeOutInMs is the maximum amount of time (in milliseconds) that is used to control how **CvClient::processEvents** times out. Use this timeout argument as follows:

< 0 (Blocking) — Signals **processEvents** to never time out. Set this value if you want **processEvents** to block until a message arrives.

0 (Polling) — Signals **processEvents** to process a single message with no blocking. The function returns immediately if no events are pending. Set this value if you want **processEvents** to poll for an event. This value is useful for applications that use their own **select** loop and already know that an event is waiting.

> 0 (Blocking with timeout) — Signals **processEvents** to time out. Set this value if you want **processEvents** to return when either a single event occurs or the timeout expires. This value is useful for complex applications that use other kinds of timeouts. You can set this value higher or lower than the timeout that the client stack uses to wait for responses from the server.

ReturnOnError signals **processEvents** to return when an error return occurs in the **select** loop (such as when a signal is caught). Use this argument when *TimeOutInMs* is a non-zero value. Set to 0 if you do not want to be notified when an error occurs. Set to 1 to be notified when an error occurs.

Notes

The return value from this function is the same as that from the last **select** call that it makes. The value **errno** is untouched for examination by the caller.

Prerequisites

Use this function in conjunction with a **select** loop:

1. Before you issue this function, issue **CvClient::getSelectInfo** to obtain information needed to pass to a **select** system call.
2. Issue the **select** call.
3. If **select** returns 0, issue **CvClient::timeout** to determine whether a timeout occurred from the server.
4. When **select** returns a non-zero value, issue this function to receive and process a message from the server.

Post-processing

None.

See Also

getSelectInfo and **timeout**.

Purpose Determines if a timeout occurred for an outstanding asynchronous request.

Syntax **void CvClient::timeout( )**

Description **timeout** determines if a timeout occurred for an outstanding asynchronous request. The function calls the client's callback handler appropriately if any timeouts occurred.

Notes Issue this call when **select** returns 0.

Prerequisites None.

Post-processing None.

See Also **processEvents**.

CvClient::ObjectId

CvClient::ObjectId 3-69
getCard 3-71
getChannel 3-72
getDlci 3-73
getE164Address 3-74
getLPort 3-76
getName 3-77
getName2 3-78
getNetwork 3-79
getPMPSpvcLeafNo 3-80
getPPort 3-81
getStartVpi 3-82
getSVCAddress 3-83
getSwitch 3-85
getTrafficShaperId 3-86
getVci 3-87
getVpi 3-88
set 3-89
setAtmInfo 3-92
setCard 3-94
setChannel 3-96
setDlci 3-98
setE164Address 3-100
setLPort 3-102
setName 3-104
setName2 3-106
setNameAndType 3-108
setNetwork 3-110
setPMPSpvcLeafNo 3-112
setPPort 3-114
setSVCAddressAndType 3-116
setSwitch 3-118
setTrafficShaper 3-120
setType 3-122
setVirtualUniLPort 3-123
type 3-125

Purpose

Used to identify objects. **CvClient::ObjectId** has member functions to set and read specific type-value pairs to and from the object ID. Fields in the object ID have specific members to **set**/**get** their values. To designate IP addresses, E.164 addresses, SVC addresses, and ATM endpoints for circuits, you use other specific member functions, such as **setE164Address**, **setVci**, and so on.

The API provides several constructors for creating a **CvClient::ObjectId** class object:

Syntax

CvClient::ObjectId ( )

This null constructor declares an empty object ID by setting its type to CVT_Unknown.

Syntax

CvClient::ObjectId (const ObjectId& that)

This copy constructor makes a copy of an object ID.

Syntax

CvClient::ObjectId (const ObjectId& *Parent***,**
CvObjectType *ObjType***,**
int *ObjId* **);**

This constructor builds an object ID based on a known parent object in the containment hierarchy.

Parent is the **CvClient::ObjectId** class that specifies the parent object (the object that contains the next highest level type-value pair).

ObjType is the object type of the lowest level type-value pair.

ObjId is the value of the lowest level type-value pair.

Syntax

CvClient::ObjectId& operator = (const ObjectId&)

This is the assignment operator.

Syntax

int operator == (const ObjectId&) const

This is the comparison operator.

Syntax

operator CvIpAddr ( ) const

This operator returns an IP address (switch ID and network number).

Syntax

friend ostream &operator << (ostream&, const ObjectId &);

This operator allows you to write the text descriptions of an object ID to an ostream (such as a file or a character stream).

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you create a **CvClient::ObjectId** class that specifies the object's values.

Purpose Reads the card slot number out of an object ID.

Syntax **unsigned long CvClient::ObjectId::getCard( ) const**

Description **getCard** reads the card slot number out of an object ID. This function is part of the **get***ObjectType* family of functions used to read specified values out of the object ID.

Notes If the type of object specified by **CvClient::ObjectId** is not either CVT_Card or the child of a card, the function returns 0.

Prerequisites Make sure that the object ID is valid.

Post-processing None.

See Also Other **get***ObjectType* functions.

getChannel

Purpose Reads the channel number out of an object ID.

Syntax **unsigned long CvClient::ObjectId::getChannel( ) const**

Description **getChannel** reads the channel number (range 1 - 28) out of an object ID. This function is part of the **get***ObjectType* family of functions used to read specified values out of the object ID.

Notes If the type of object specified by **CvClient::ObjectId** is not CVT_Channel or the child of a channel, the function returns 0.

Prerequisites Make sure that the object ID is valid.

Post-processing None.

See Also Other **get***ObjectType* functions.

Purpose Reads the circuit DLCI number out of an object ID.

Syntax **unsigned long CvClient::ObjectId::getDlci( ) const**

Description **getDlci** reads the circuit DLCI number out of an object ID. This function is part of the **get***ObjectType* family of functions used to read specified values out of the object ID.

Notes If the type of object specified by **CvClient::ObjectId** is not CVT_Circuit or the child of a circuit, the function returns 0.

Prerequisites Make sure that the object ID is valid.

Post-processing None.

See Also Other **get***ObjectType* functions.

getE164Address

Purpose

Reads the value of an E.164 address out of an object ID.

Syntax

const CvE164Address&
CvClient::ObjectId::getE164Address( ) const

Description

getE164Address reads the 10-character E.164 address string out of an object ID. This function is part of the **get***ObjectType* family of functions used to read specified values out of the object ID.

The class **CvE164Addr** is used to represent E.164 addresses. The class enforces the basic syntax rules of E.164 addresses. The E.164 address can be:

- Full address without country code, such as 9789521111. The server will prepend the default country code to the address. To specify the default country code, use the environment variable CV_DFLT_SMDS_CC.
- Full address with country code, such as 1-9789521111. The server interprets characters preceding a dash (-) as the country code part of the address. Note that the dash is required between the country code and the remainder of the address.
- Address prefix only, such as 978952. To specify the length of the prefix, use the environment variable CV_SMDS_MASK_SIZE.
- Country code only, such as 1. A maximum of 3 characters is allowed.

CvE164Addr has a constructor that converts an E.164 address to or from a given string. You do not have to declare objects of this type. To set an E.164 address to a value or an object ID, cast a **char *** to a **CvE164Addr** address. To read an E.164 address, cast a **CvE164Addr** address to a **char ***. Keep in mind that if you violate the syntax rules of E.164 addresses, casting a **CvE164Addr** address to a **char *** produces a null pointer.

If the type of object specified by **CvClient::ObjectId** does not use an E.164 address, the function returns a null address with length 0.

Prerequisites Make sure that the object ID is valid.

Post-processing None.

See Also Other **get***ObjectType* functions.

getLPort

Purpose Reads the LPort number out of an object ID.

Syntax **unsigned long CvClient::ObjectId::getLPort( ) const**

Description **getLPort** reads the LPort number out of an object ID. This function is part of the **get***ObjectType* family of functions used to read specified values out of the object ID.

Notes If the type of object specified by **CvClient::ObjectId** is not CVT_LPort or the child of an LPort, the function returns 0.

Prerequisites Make sure that the object ID is valid.

Post-processing None.

See Also Other **get***ObjectType* functions.

Purpose

Reads the name out of an object ID.

Syntax

const char *CvClient::ObjectId::getName( ) const

Description

getName reads the name out of an object ID. This function is part of the **get***ObjectType* family of functions used to read specified values out of the object ID.

Notes

If the type of object specified by **CvClient::ObjectId** is not one of the following valid types, the function returns a null name with length 0:

CVT_AssignedSvcSecScn
CVT_Customer
CVT_ServiceName
CVT_SvcCUG
CVT_SvcCUGMbr
CVT_SvcCUGMbrRule
CVT_SvcSecScn
CVT_TrafficDesc
CVT_Vpn

For an SvcCUGMbr object, this function reads the CUG name string. To read the member name string of an SvcCUGMbr object, use **getName2**.

Prerequisites

Make sure that the object ID is valid.

Post-processing

None.

See Also

Other **get***ObjectType* functions.

getName2

Purpose

Reads the member name string of an SvcCUGMbr object out of an object ID.

Syntax

const char *CvClient::ObjectId::getName2( ) const

Description

getName2 reads the member name out of an SvcCUGMbr object ID. This function is part of the **get***ObjectType* family of functions used to read specified values out of the object ID.

Notes

If the type of object specified by **CvClient::ObjectId** is not CVT_SvcCUGMbr, the function returns a null name with length 0.

To read the CUG name string out of an SvcCUGMbr object, use **getName**. To read the name string of other types of objects, use **getName**.

Prerequisites

Make sure that the object ID is valid.

Post-processing

None.

See Also

Other **get***ObjectType* functions.

Purpose

Reads the network number part of the IP address out of an object ID.

Syntax

unsigned long CvClient::ObjectId::getNetwork( ) const

Description

getNetwork reads the network number part of the IP address out of an object ID; the lower order bits are 0. This function is part of the **get***ObjectType* family of functions used to read specified values out of the object ID.

Prerequisites

Make sure that the object ID is valid.

Post-processing

None.

See Also

Other **get***ObjectType* functions.

getPMPSpvcLeafNo

Purpose Reads the PMP SpvcLeaf number out of an object ID.

Syntax **unsigned long CvClient::ObjectId::getPMPSpvcLeafNo( ) const**

Description **getPMPSpvcLeafNo** reads the SpvcLeaf number out of an object ID. This function is part of the **get***ObjectType* family of functions used to read specified values out of the object ID.

Notes If the type of object specified by **CvClient::ObjectId** is not CVT_PMPSpvcLeaf, the function returns 0.

Prerequisites Make sure that the object ID is valid.

Post-processing None.

See Also Other **get***ObjectType* functions.

Purpose Reads the PPort number out of an object ID.

Syntax **unsigned long CvClient::ObjectId::getPPort( ) const**

Description **getPPort** reads the PPort number out of an object ID. This function is part of the **get***ObjectType* family of functions used to read specified values out of the object ID.

Notes If the type of object specified by **CvClient::ObjectId** is not CVT_PPort or the child of a PPort, the function returns 0.

Prerequisites Make sure that the object ID is valid.

Post-processing None.

See Also Other **get***ObjectType* functions.

getStartVpi

Purpose

Reads the number that represents the lower end of the range of VPI values out of an object ID for a virtual UNI LPort.

Syntax

unsigned long CvClient::ObjectId::getStartVpi( ) const

Description

getStartVpi reads the lower end of the range of VPI values out of an object ID for a virtual UNI LPort. This function is part of the **get***ObjectType* family of functions used to read specified values out of the object ID.

Prerequisites

Make sure that the object ID is valid.

Post-processing

None.

See Also

Other **get***ObjectType* functions.

Purpose

Reads the value of an SVC address out of an object ID.

Syntax

const SVCAddress& CvClient::ObjectId::getSVCAddress( ) const

Description

getSVCAddress reads the SVC address string out of an object ID. This function is part of the **get***ObjectType* family of functions used to read specified values out of the object ID.

Notes

The class **CvSVCAddress** is used to represent SVC addresses. The class enforces the basic syntax rules of of the seven different SVC address formats:

E.164native
DCCAESA
ICDAESA
E.164AESA
CustomAESA
DefaultRoute
UserPart

CvSVCAddress has a constructor that converts an SVC address to or from a given string. You do not have to declare objects of this type. To set an SVC address to a value or an object ID, cast a **char *** to a **CvSVCAddress** address. To read an SVC address, cast a **CvSVCAddress** address to a **char ***. Keep in mind that if you violate the syntax rules of SVC addresses, casting a **CvSVCAddress** address to a **char *** produces a null pointer. For details, refer to “CvSVCAddress”.

Prerequisites

Make sure that the object ID is valid.

Post-processing

None.

See Also Other **get***ObjectType* functions.

Purpose Reads the IP address out of an object ID.

Syntax **unsigned long CvClient::ObjectId::getSwitch( ) const**

Description **getSwitch** reads the IP address out of an object ID. This function is part of the **get***ObjectType* family of functions used to read specified values out of the object ID.

Prerequisites Make sure that the object ID is valid.

Post-processing None.

See Also Other **get***ObjectType* functions

getTrafficShaperId

Purpose Reads the Traffic Shaper identifier value out of an object ID.

Syntax **unsigned long CvClient::ObjectId::getTrafficShaperId( ) const**

Description **getTrafficShaperId** reads the Traffic Shaper identifier value out of an object ID. This function is part of the **get***ObjectType* family of functions used to read specified values out of the object ID.

Notes If the type of object specified by **CvClient::ObjectId** is not CVT_TrafficShaper, the function returns 0.

Prerequisites Make sure that the object ID is valid.

Post-processing None.

See Also Other **get***ObjectType* functions.

Purpose Reads the circuit VCI number out of an object ID.

Syntax **unsigned long CvClient::ObjectId::getVci( ) const**

Description **getVci** reads the VCI number of an endpoint of the ATM circuit out of an object ID. This function is part of the **get***ObjectType* family of functions used to read specified values out of the object ID.

Notes The function can return 0 if:

- The type of object specified by **CvClient::ObjectId** is not CVT_Circuit, CVT_Spvc, CVT_PMPCkt, CVT_PMPCktRoot, or CVT_PMPSpvcRoot.
- The type of object specified by **CvClient::ObjectId** is a Frame Relay circuit.

Prerequisites Make sure that the object ID is valid.

Post-processing None.

See Also Other **get***ObjectType* functions.

getVpi

Purpose

Reads the circuit VPI number out of an object ID.

Syntax

unsigned long CvClient::ObjectId::getVpi( ) const

Description

getVpi reads the VPI number of an endpoint of the ATM circuit out of an object ID. This function is part of the **get***ObjectType* family of functions used to read specified values out of the object ID.

Notes

The function can return 0 if:

- The type of object specified by **CvClient::ObjectId** is not CVT_Circuit, CVT_Spvc, CVT_PMPCkt, CVT_PMPCktRoot, or CVT_PMPSpvcRoot.
- The type of object specified by **CvClient::ObjectId** is a Frame Relay circuit.

Prerequisites

Make sure that the object ID is valid.

Post-processing

None.

See Also

Other **get***ObjectType* functions.

Purpose

Sets an SMDS address field of an object ID with an E.164 address.

Syntax

int CvClient::ObjectId::set(
CvObjectType *type***;**
CvE164Address *addr***;**
int *changeType* **);**

Description

set sets a specified field of an object ID with an E.164 address. This function is part of the **set***ObjectType* family of functions used to create an object ID based on the containment hierarchy. Use this function for all SMDS objects except screens.

type identifies the type of the object.

addr is the E.164 address string of the SMDS object.

The class **CvE164Addr** is used to represent E.164 addresses. The class enforces the basic syntax rules of E.164 addresses. The E.164 address can be:

- Full address without country code, such as 9789521111. The server will prepend the default country code to the address. To specify the default country code, use the environment variable CV_DFLT_SMDS_CC.
- Full address with country code, such as 1-9789521111. The server interprets characters preceding a dash (-) as the country code part of the address. Note that the dash is required between the country code and the remainder of the address.
- Address prefix only, such as 978952. To specify the length of the prefix, use the environment variable CV_SMDS_MASK_SIZE.
- Country code only, such as 1. A maximum of 3 characters is allowed.

CvE164Addr has a constructor that converts an E.164 address to or from a given string. You do not have to declare objects of this type. To set an E.164 address to a value or an object ID, cast a **char *** to a **CvE164Addr** address. To read an E.164 address, cast a **CvE164Addr** address to a **char ***. Keep in mind that if you violate the syntax rules of E.164 addresses, casting a **CvE164Addr** address to a **char *** produces a null pointer.

changeType indicates whether you want to change the object type to the specific SMDS object type. Use this argument as follows:

- Zero leaves the object type as is.
- Non-zero changes the object type to the specific SMDS object type.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build an object ID with the object's values. Use this function in conjunction with other **set***ObjectType* member functions to build the object ID based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

The function returns 0 if the parent object's type does not support the specified *type*. However, this is only a warning; the address is set nonetheless.

The E.164 address will be set to a null string if the address specified by *addr* is not syntatically correct.

Prerequisites

None.

Post-processing Once you fill in the complete object ID based on the containment hierarchy, issue **CvClient::ObjectType::setType** to specify the object type.

See Also Other **set***ObjectType* functions.

setAtmInfo

Purpose

Sets the circuit field of an object ID with a value for an object's ATM circuit ID.

Syntax

int CvClient::ObjectId::setAtmInfo(
unsigned long *vpi***;**
unsigned long *vci* **);**

Description

setAtmInfo fills in an object ID with the values for an object's ATM circuit ID. This function is also used to build an atmXportForFrNni LPort. This function is part of the **set***ObjectType* family of functions used to create an object ID based on the containment hierarchy.

vpi is the VPI number of an endpoint of the circuit. Valid values for the VPI endpoint are 0-15.

vci is the VCI number of an endpoint of the circuit.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build an object ID with the object's values. Use this function in conjunction with other **set***ObjectType* member functions to build the object ID based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

The function returns 0 if the parent object does not support the specified field. However, this is only a warning; the circuit ID is set nonetheless.

Prerequisites

None.

Post-processing Once you have filled in the complete object ID based on the containment hierarchy, you must issue **CvClient::ObjectType::setType** to specify the object type.

See Also Other **set***ObjectType* functions.

setCard

Purpose

Sets the card field of an object ID with a value for an object's card ID.

Syntax

int CvClient::ObjectId::setCard(
unsigned long *instance***;**
int *changeType* **);**

Description

setCard fills in an object ID with a value for an object's card ID. This function is part of the **set***ObjectType* family of functions used to create an object ID based on the containment hierarchy.

instance is the slot number of the card.

changeType, an optional parameter, indicates whether you want to change the object type to CVT_Card. Use this argument to convert a parent object into a child object or a child into a parent. Valid values are as follows:

- Zero leaves the object type as is (default).
- Non-zero changes the object type to CVT_Card.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build an object ID with the object's values. Use this function in conjunction with other **set***ObjectType* member functions to build the object ID based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

The function returns 0 if either:

- The parent object does not support the specified field.

- The original type of the object ID cannot be converted to the new type.

However, this is only a warning; the card ID is set nonetheless.

Prerequisites None.

Post-processing If you did not use *changeType* to change the object type, issue **CvClient::ObjectType::setType** to specify the object type once you have filled in the complete object ID based on the containment hierarchy.

See Also Other **set***ObjectType* functions.

setChannel

Purpose

Sets the channel field of an object ID with a value for an object's LPort ID.

Syntax

int CvClient::ObjectId::setChannel(
unsigned long *instance***;**
int *changeType* **);**

Description

setChannel fills in an object ID with a value for an object's channel ID. This function is part of the **set***ObjectType* family of functions used to create an object ID based on the containment hierarchy.

instance is the number of the channel.

changeType, an optional parameter, indicates whether you want to change the object type to CVT_Channel. Use this argument to convert a parent object into a child object or a child into a parent. Valid values are as follows:

- Zero leaves the object type as is (default).
- Non-zero changes the object type to CVT_Channel.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build an object ID with the object's values. Use this function in conjunction with other **set***ObjectType* member functions to build the object ID based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

The function returns 0 if either:

- The parent object does not support the specified field.

- The original type of the object ID cannot be converted to the new type.

However, this is only a warning; the channel ID is set nonetheless.

Prerequisites

None.

Post-processing

If you did not use *changeType* to change the object type, issue **CvClient::ObjectType::setType** to specify the object type once you have filled in the complete object ID based on the containment hierarchy.

See Also

Other **set***ObjectType* functions.

setDlci

Purpose

Sets the circuit field of an object ID with a value for an object's circuit ID.

Syntax

int CvClient::ObjectId::setDlci(
unsigned long *instance***;**
int *changeType* **);**

Description

setDlci fills in an object ID with a value for an object's Frame Relay circuit ID. This function is part of the **set***ObjectType* family of functions used to create an object ID based on the containment hierarchy.

instance is the DLCI number of an endpoint of the circuit.

changeType, an optional parameter, indicates whether you want to change the object type to CVT_Dlci. Use this argument to convert a parent object into a child object or a child into a parent. Valid values are as follows:

- Zero leaves the object type as is (default).
- Non-zero changes the object type to CVT_Dlci.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build an object ID with the object's values. Use this function in conjunction with other **set***ObjectType* member functions to build the object ID based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

The function returns 0 if either:

- The parent object does not support the specified field.

- The original type of the object ID cannot be converted to the new type.

However, this is only a warning; the circuit ID is set nonetheless.

Prerequisites None.

Post-processing If you did not use *changeType* to change the object type, issue **CvClient::ObjectType::setType** to specify the object type once you have filled in the complete object ID based on the containment hierarchy.

See Also Other **set***ObjectType* functions.

setE164Address

Purpose

Sets an object ID with a value for an object's E.164 address.

Syntax

int CvClient::ObjectId::setE164Address(
const CvE164Address& *addr* **);**

Description

setE164Address fills in an object ID with a value for an object's E.164 address. This function is part of the **set***ObjectType* family of functions used to create an object ID based on the containment hierarchy.

addr is the E.164 address.

The class **CvE164Addr** is used to represent E.164 addresses. The class enforces the basic syntax rules of E.164 addresses. The E.164 address can be:

- Full address without country code, such as 9789521111. The server will prepend the default country code to the address. To specify the default country code, use the environment variable CV_DFLT_SMDS_CC.
- Full address with country code, such as 1-9789521111. The server interprets characters preceding a dash (-) as the country code part of the address. Note that the dash is required between the country code and the remainder of the address.
- Address prefix only, such as 978952. To specify the length of the prefix, use the environment variable CV_SMDS_MASK_SIZE.
- Country code only, such as 1. A maximum of 3 characters is allowed.

CvE164Addr has a constructor that converts an E.164 address to or from a given string. You do not have to declare objects of this type. To set an E.164 address to a value or an object ID, cast a **char *** to a **CvE164Addr** address. To read an E.164 address, cast a **CvE164Addr** address to a **char ***. Keep in mind that if you violate the syntax rules of E.164 addresses, casting a **CvE164Addr** address to a **char *** produces a null pointer.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build an object ID with the object's values. Use this function in conjunction with other **set***ObjectType* member functions to build the object ID based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

The function returns 0 if the parent object does not support the specified field. However, this is only a warning; the address is set nonetheless.

The E.164 address will be set to a null string if the address specified by *addr* is not syntatically correct.

Prerequisites

None.

Post-processing

Once you have filled in the complete object ID based on the containment hierarchy, you must issue **CvClient::ObjectType::setType** to specify the object type.

See Also

Other **set***ObjectType* functions.

setLPort

Purpose

Sets the LPort field of an object ID with a value for an object's LPort ID.

Syntax

int CvClient::ObjectId::setLPort(
unsigned long *instance***;**
int *changeType* **);**

Description

setLPort fills in an object ID with a value for an object's Frame Relay LPort ID. This function is part of the **set***ObjectType* family of functions used to create an object ID based on the containment hierarchy.

instance is the port number of the LPort.

changeType, an optional parameter, indicates whether you want to change the object type to CVT_LPort. Use this argument to convert a parent object into a child object or a child into a parent. Valid values are as follows:

- Zero leaves the object type as is (default).
- Non-zero changes the object type to CVT_LPort.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build an object ID with the object's values. Use this function in conjunction with other **set***ObjectType* member functions to build the object ID based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

The function returns 0 if either:

- The parent object does not support the specified field.

- The original type of the object ID cannot be converted to the new type.

However, this is only a warning; the LPort ID is set nonetheless.

Prerequisites None.

Post-processing If you did not use *changeType* to change the object type, issue **CvClient::ObjectType::setType** to specify the object type once you have filled in the complete object ID based on the containment hierarchy.

See Also Other **set***ObjectType* functions.

setName

Purpose

Sets the name field of an object ID with an object's name.

Syntax

int CvClient::ObjectId::setName(
const char* *name* **);**

Description

setName fills in an object ID with an object's name. This function is part of the **set***ObjectType* family of functions used to create an object ID based on the containment hierarchy.

name is the object's name string.

Notes

Valid object types for this function are as follows:

CVT_AssignedSvcSecScn
CVT_Customer
CVT_ServiceName
CVT_SvcCUG
CVT_SvcCUGMbr
CVT_SvcCUGMbrRule
CVT_SvcSecScn
CVT_TrafficDesc
CVT_Vpn

For an SvcCUGMbr object, this function sets the CUG name string. To set the member name string of an SvcCUGMbr object, use **setName2**.

To set both the object name and type, use **setNameAndType**.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build an object ID with the object's values. Use this function in conjunction with other **set***ObjectType* member functions to build the object ID based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

The function returns 0 if the parent object does not support the specified field.

However, this is only a warning; the name is set nonetheless.

Prerequisites None.

Post-processing None.

See Also Other **set***ObjectType* functions.

setName2

Purpose

Sets the SvcCUGMbr field of an object ID with an object's SvcCUGMbr member name.

Syntax

int CvClient::ObjectId::setName2(
const char* *name* **);**

Description

setName2 fills in an object ID with an SvcCUGMbr object's member name. This function is part of the **set***ObjectType* family of functions used to create an object ID based on the containment hierarchy.

name is the member name string.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build an object ID with the object's values. Use this function in conjunction with other **set***ObjectType* member functions to build the object ID based on the containment hierarchy.

To set the CUG name string of an SvcCUGMbr object, use **setName**. To read the name string of other types of objects, use **getName**. To set both the object name and type, use **setNameAndType**.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

The function returns 0 if the parent object does not support the specified field. However, this is only a warning; the name is set nonetheless.

Prerequisites

None.

Post-processing None.

See Also Other **set***ObjectType* functions.

setNameAndType

Purpose

Sets the name and type fields of an object ID with values for an object's name and type.

Syntax

int CvClient::ObjectId::setNameAndType(
CvObjectType *type***;**
const char* *name*
const char* *name2*=**NULL);**

Description

setNameAndType fills in an object ID with a value for an object's name and type. This function is part of the **set***ObjectType* family of functions used to create an object ID based on the containment hierarchy.

type identifies the type of the object.

name is the object name string. Valid objects are as follows:

CVT_AssignedSvcSecScn
CVT_Customer
CVT_ServiceName
CVT_SvcCUG
CVT_SvcCUGMbr
CVT_SvcCUGMbrRule
CVT_SvcSecScn
CVT_TrafficDesc
CVT_Vpn

For an SvcCUGMbr object, this parameter sets the CUG name string. To set the member name string of an SvcCUGMbr object, use the *name2* parameter.

name2 is the member name string of an SvcCUGMbr object.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build an object ID with the object's values. Use this function in conjunction with other **set***ObjectType* member functions to build the object ID based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

The function returns 0 if the parent object does not support the specified field. However, this is only a warning; the name and type is set nonetheless.

Prerequisites

None.

Post-processing

None.

See Also

Other **set***ObjectType* functions.

setNetwork

Purpose

Sets the network field of an object ID with a value for an object's network number.

Syntax

int CvClient::ObjectId::setNetwork(
const CvClient::IpAddr& *addr***;**
int *changeType* **);**

Description

setNetwork fills in an object ID with a value for the network number and zeros out the last octets of the IP address that specify a node. This function is part of the **set***ObjectType* family of functions used to create an object ID based on the containment hierarchy.

addr is an IP address.

changeType, an optional parameter, indicates whether you want to change the object type to CVT_Network. Use this argument to convert a parent object into a child object or a child into a parent. Valid values are as follows:

- Zero leaves the object type as is (default).
- Non-zero changes the object type to CVT_Network.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build an object ID with the object's values. Use this function in conjunction with other **set***ObjectType* member functions to build the object ID based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

The function returns 0 if either:

- The parent object does not support the specified field.
- The original type of the object ID cannot be converted to the new type.

However, this is only a warning; the network number is set nonetheless.

Prerequisites None.

Post-processing If you did not use *changeType* to change the object type, issue **CvClient::ObjectType::setType** to specify the object type once you have filled in the complete object ID based on the containment hierarchy.

See Also Other **set***ObjectType* functions.

setPMPSpvcLeafNo

Purpose

Sets the PMP SpvcLeaf field of an object ID with a value for an object's SpvcLeaf ID.

Syntax

int CvClient::ObjectId::setPMPSpvcLeafNo(
unsigned long *instance***;**
int *changeType* **);**

Description

setPMPSpvcLeafNo fills in an object ID with a value for an object's SpvcLeaf ID. This function is part of the **set***ObjectType* family of functions used to create an object ID based on the containment hierarchy.

instance is the number of the SpvcLeaf.

changeType, an optional parameter, indicates whether you want to change the object type to CVT_PMPSpvcLeaf. Use this argument to convert a parent object into a child object or a child into a parent. Valid values are as follows:

- Zero leaves the object type as is (default).
- Non-zero changes the object type to CVT_PMPSpvcLeaf.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build an object ID with the object's values. Use this function in conjunction with other **set***ObjectType* member functions to build the object ID based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

The function returns 0 if either:

- The parent object does not support the specified field.

- The original type of the object ID cannot be converted to the new type.

However, this is only a warning; the SpvcLeaf ID is set nonetheless.

Prerequisites

None.

Post-processing

If you did not use *changeType* to change the object type, issue **CvClient::ObjectType::setType** to specify the object type once you have filled in the complete object ID based on the containment hierarchy.

See Also

Other **set***ObjectType* functions.

setPPort

Purpose

Sets the PPort field of an object ID with a value for an object's PPort ID.

Syntax

int CvClient::ObjectId::setPPort(
unsigned long *instance***;**
int *changeType* **);**

Description

setPPort fills in an object ID with a value for an object's PPort ID. This function is part of the **set***ObjectType* family of functions used to create an object ID based on the containment hierarchy.

instance is the port number of the PPort.

changeType, an optional parameter, indicates whether you want to change the object type to CVT_PPort. Use this argument to convert a parent object into a child object or a child into a parent. Valid values are as follows:

- Zero leaves the object type as is (default).
- Non-zero changes the object type to CVT_PPort.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build an object ID with the object's values. Use this function in conjunction with other **set***ObjectType* member functions to build the object ID based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

The function returns 0 if either:

- The parent object does not support the specified field.

- The original type of the object ID cannot be converted to the new type.

However, this is only a warning; the PPort ID is set nonetheless.

Prerequisites

None.

Post-processing

If you did not use *changeType* to change the object type, issue **CvClient::ObjectType::setType** to specify the object type once you have filled in the complete object ID based on the containment hierarchy.

See Also

Other **set***ObjectType* functions.

setSVCAddressAndType

Purpose

Sets an object ID with a value for an object's SVC address and type.

Syntax

void CvClient::ObjectId::setSVCAddressAndType(
const CvSVCAddress& *addr***;**
CvObjectType *type* **);**

Description

setSVCAddressAndType fills in an object ID with a value for an object's SVC address. This function is part of the **set***ObjectType* family of functions used to create an object ID based on the containment hierarchy.

addr is the SVC address.

The class **CvSVCAddress** is used to represent SVC addresses. The class enforces the basic syntax rules of the seven different SVC address formats:

E.164native
DCCAESA
ICDAESA
E.164AESA
CustomAESA
DefaultRoute
UserPart

CvSVCAddress has a constructor that converts an SVC address to or from a given string. You do not have to declare objects of this type. To set an SVC address to a value or an object ID, cast a **char *** to a **CvSVCAddress** address. To read an SVC address, cast a **CvSVCAddress** address to a **char ***. Keep in mind that if you violate the syntax rules of SVC addresses, casting a **CvSVCAddress** address to a **char *** produces a null pointer. For details, refer to "CvSVCAddress".

type identifies the type of the object.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build an object ID with the object's values. Use this function in conjunction with other **set***ObjectType* member functions to build the object ID based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

The function returns 0 if the parent object does not support the specified field. However, this is only a warning; the address is set nonetheless.

The SVC address will be set to a null string if the address specified by *addr* is not syntatically correct.

Prerequisites

None.

Post-processing

None.

See Also

Other **set***ObjectType* functions.

setSwitch

Purpose

Sets the switch field of an object ID with a value for an object's IP address.

Syntax

int CvClient::ObjectId::setSwitch(
const CvClient::IpAddr& *addr***;**
int *changeType* **);**

Description

setSwitch fills in an object ID with a value for the object's IP address. This function is part of the **set***ObjectType* family of functions used to create an object ID based on the containment hierarchy.

addr is the object's IP address.

changeType, an optional parameter, indicates whether you want to change the object type to CVT_Switch. Use this argument to convert a parent object into a child object or a child into a parent. Valid values are as follows:

- Zero leaves the object type as is (default).
- Non-zero changes the object type to CVT_Switch.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build an object ID with the object's values. Use this function in conjunction with other **set***ObjectType* member functions to build the object ID based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

The function returns 0 if either:

- The parent object does not support the specified field.

- The original type of the object ID cannot be converted to the new type.

However, this is only a warning; the address is set nonetheless.

Prerequisites None.

Post-processing If you did not use *changeType* to change the object type, issue **CvClient::ObjectType::setType** to specify the object type once you have filled in the complete object ID based on the containment hierarchy.

See Also Other **set***ObjectType* functions.

setTrafficShaper

Purpose

Sets the Traffic Shaper field of an object ID with a value for an object's Traffic Shaper ID.

Syntax

int CvClient::ObjectId::setTrafficShaper(
unsigned long *instance* **;**
int *changeType* **);**

Description

setTrafficShaper fills in an object ID with a value for an object's Traffic Shaper ID. This function is part of the **set***ObjectType* family of functions used to create an object ID based on the containment hierarchy.

instance is the one-based identifier for the traffic shaper attributes (range 1 - 15).

changeType, an optional parameter, indicates whether you want to change the object type to CVT_TrafficShaper. Use this argument to convert a parent object into a child object or a child into a parent. Valid values are as follows:

- Zero leaves the object type as is (default).
- Non-zero changes the object type to CVT_TrafficShaper.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build an object ID with the object's values. Use this function in conjunction with other **set***ObjectType* member functions to build the object ID based on the containment hierarchy.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

The function returns 0 if either:

- The parent object does not support the specified field.
- The original type of the object ID cannot be converted to the new type.

However, this is only a warning; the Traffic Shaper ID is set nonetheless.

Prerequisites None.

Post-processing If you did not use *changeType* to change the object type, issue **CvClient::ObjectType::setType** to specify the object type once you have filled in the complete object ID based on the containment hierarchy.

See Also Other **set***ObjectType* functions.

setType

Purpose

Sets the type of an object ID.

Syntax

void CvClient::ObjectId::setType(
CvObjectType *newtype* **);**

Description

setType fills in an object ID with a value for an object's type.

newtype identifies the type of the object.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build an object ID with the object's values. Use this function in conjunction with **set***ObjectType* member functions to build the object ID based on the containment hierarchy. Once you fill in the complete object ID based on the containment hierarchy, issue **setType** to specify the object type.

Prerequisites

Issue **set***ObjectType* member functions to build the object ID based on the containment hierarchy.

Post-processing

None.

See Also

Other **set***ObjectType* functions.

Purpose

Sets the LPort field of an object ID with a value for an object's ATM Virtual UNI LPort start VPI value.

Syntax

int CvClient::ObjectId::setVirtualUniLPort(
unsigned long *instance* **);**

Description

setVirtualUniLPort fills in an object ID with a value for an object's ATM Virtual UNI LPort start VPI value. This function is part of the **set***ObjectType* family of functions used to create an object ID based on the containment hierarchy.

instance is the number that represents the lower end of the range of VPI values.

Notes

The operational functions require you to specify an object ID to be acted on. To do so, you build an object ID with the object's values. Use this function in conjunction with other **set***ObjectType* member functions to build the object ID based on the containment hierarchy.

This function sets the LPort field of the Object ID to 0 and sets the object type to CVT_LPort.

Before the client issues this function to the Provisioning Server, it checks the function for correct syntax. The server checks the input parameters for validity and reports errors back to the client.

The function returns 0 if the parent object does not support the specified field.

Prerequisites

None.

Post-processing

None.

See Also Other **set***ObjectType* functions.

Purpose Reads the type out of an object ID.

Syntax **CvObjectType CvClient::ObjectId::type( ) const**

Description **type** reads the object type out of an object ID.

Prerequisites Make sure that the object ID is valid.

Post-processing None.

See Also **get***ObjectType* functions.

CvClient::Args

CvClient::Args 3-127
count 3-130
errorIndex 3-131
exists 3-132
getStatus 3-133
getType 3-134
idAt 3-135
setNull 3-136
statusAt 3-137
valueAt 3-138

Purpose

Used to build an argument list from attribute-value pairs. **CvClient::Args** has member functions to set and read specific attribute-value pairs to and from the argument list.

The API treats instances of **CvClient::Args** as single values. You can make copies of **CvClient::Args** objects, create individual **CvClient::Args** objects on a stack, and assign the objects with the assignment operator. You can also create temporary **CvClient::Args** objects and pass them as arguments.

To support multiple data types, **CvClient::Args** uses the helper class **CvClient::Value**. This class has constructors for all the appropriate object types, including integer, printable string, object ID, and IP address (**CvClient::IpAddr**). You do not need to explicitly create a **CvClient::Value** class to add a value to an argument list.

The API provides the following constructor for creating a **CvClient::Args** class object:

Syntax

CvClient::Args ( )

This null constructor declares an empty argument list.

Notes

Most of the API functions that perform provisioning operations on network components take one or more attributes. The attributes are specified in an argument list. To create an argument list, you create a **CvClient::Args** object that specifies the attribute-value pairs.

The C++ API provides two options for specifying an argument list. To set values into an argument list, you can either:

- Construct an argument list by specifying a variable number of arguments. For example:

  ```
  CvClient::Args myArgs( argId1, “String Value”, argId2, IntValue );
  ```

- Set arguments individually using the operator **[ ]**. For example:

```
CvClient::Args myArgs( )

myArgs[CVA_argId1] = “String Value”;

myArgs[CVA_argId2] = IntValue;
```

The operator **[ ]** also allows you to read values out of an argument list using an argument ID symbol as an index into the list.

The operator **[ ]** is available in both **const** and non-**const** versions. However, unless the **CvClient::Args** object is a **const**, there is no guarantee that the **const** version will be invoked. If the non-**const** version is invoked and a specified argument is not in the **CvClient::Args** object, the argument is added to the argument list as a null value. Thus, do not use the operator **[ ]** to check for the existence of an argument for a non-**const CvClient::Args** object.

Instead, use the member function **exists**.

You can operate on pairs of **CvClient::Args** objects with the following operators:

== Compares two argument lists for equality. Argument lists are equal if their argument numbers, IDs, values, and status are equal. If the argument lists are equal, the operator returns a non-zero value.

!= Compares two argument lists for inequality. Argument lists are unequal if their argument numbers, IDs, values, or status differ. If the argument lists are unequal, the operator returns a non-zero value.

+ Adds two argument lists together. The result is a new argument list that combines both sets of arguments. When an argument exists in both argument lists, the value from the second list is used.

- Subtracts two argument lists. The result is a new argument list that contains the arguments from the first set that are not contained in the second set.

***** Returns the intersection of two argument lists. The result is a new argument list containing arguments that exist in both sets of arguments. Values from the first list are used.

These operators are order-independent; the result does not depend on the order of the parameters contained within either argument list.

The operator << allows you to write the text argument descriptions of an argument list to an ostream. For example, an ostream could be a file or a character stream into which you are encoding arguments.

count

Purpose Retrieves the number of arguments in an argument list.

Syntax **int CvClient::Args::count( )**

Description **count** retrieves the number of arguments in an argument list.

Prerequisites Obtain the argument list by appropriate means, such as by creating a **CvClient::Args** class or by issuing an operational function (for example **CvClient::getObject**, which returns an Args).

Post-processing None.

See Also **idAt** and **valueAt**.

Purpose Determines if any argument in an argument list has an error status.

Syntax **int CvClient::Args::errorIndex( )**

Description **errorIndex** determines if any argument in an argument list has an error status. If an error status exists, the function returns the one-based index to the first argument with an error.

Notes The function's returned value is one-based. The value 0 indicates a non-error status. If you use the returned value in another function (such as **idAt**), subtract 1 from the value.

Prerequisites None.

Post-processing To return the error status for the argument, issue **CvClient::Args::statusAt**.

See Also **getResponseErrorIndex** and **statusAt**.

exists

Purpose

Determines if a specified argument exists in an argument list.

Syntax

int CvClient::Args::exists(
CvArgId *Id***);**

Description

exists determines if a specified argument exists in an argument list. Use this function to test if a given argument has been set in an argument list.

Id is the argument ID symbol for the argument in the argument list. The argument ID symbol is defined in the file CvArgId.H.

Notes

Valid return values for this function are as follows:

- Zero if the argument specified by *Id* is in the argument list.
- Non-zero if the argument specified by *Id* is not in the argument list.

Prerequisites

None.

Post-processing

None.

See Also

getType.

Purpose

Retrieves the error status code of an argument in an argument list.

Syntax

CvStatus CvClient::Args::getStatus(
CvArgId *Id* **);**

Description

getStatus retrieves the error status code of an argument in an argument list.

Id is the argument ID symbol for the argument in the argument list. The argument ID symbol is defined in the file CvArgId.H. If *Id* is not in the argument list, the status -1 is returned. Status is defined in the file CvErrors.H.

Prerequisites

Use this function in conjunction with other utility functions that traverse the argument list:

1. Obtain the argument list by appropriate means, such as by creating a **CvClient::Args** class or by issuing an operational function (for example **CvClient::getObject**).
2. If you want to determine if any argument in the argument list has an error status, issue **CvClient::Args::errorIndex**.
3. Issue this function to retrieve the error status code of a specific argument.

Post-processing

None.

See Also

statusAt.

getType

Purpose

Retrieves an integer value that represents the data type of an argument in an argument list.

Syntax

int CvClient::Args::getType( )

Description

getType retrieves an integer value that represents the data type of an argument in an argument list.

Notes

Use this function with the **CvClient::Args** operator **[ ]** to specify an argument ID symbol as an index into the list. For example:

```
myArgs[CVA_argId1].getType( )
```

Valid return values for this function are as follows:

INTEGER
STRING
OBJID
IP_ADDR
NULLOBJ

The following values are valid return values for this function, but are currently unused by the Provisioning Server:

GAUGE
COUNTER
TIMETICKS

Prerequisites

None.

Post-processing

None.

See Also

exists.

Purpose

Retrieves the argument ID at a specified location in an argument list.

Syntax

CvArgId CvClient::Args::idAt(
int *Index* **);**

Description

idAt retrieves the argument ID at a specified location in an argument list. If the index is invalid, the function returns 0.

Index is the zero-based index to the argument list.

Notes

Use this function to read argument IDs sequentially from a **CvClient::Args** object.

Prerequisites

Obtain the argument list by appropriate means, such as by creating a **CvClient::Args** class or by issuing an operational function (for example **CvClient::getObject**).

To count the number of arguments in an argument list, issue **CvClient::Args::count**.

Post-processing

None.

See Also

count and **valueAt**.

setNull

Purpose

Sets a null value for a particular attribute in an argument list.

Syntax

void CvClient::Value::setNull( )

Description

setNull sets a null value for a particular attribute in an argument list. Use this function to add an attribute with no value to the argument list in preparation for a query.

Notes

Use this function with the **CvClient::Args** operator **[ ]** to specify an argument ID symbol as an index into the list. For example:

```
myArgs[CVA_argId1].setNull( )
```

Prerequisites

Make sure that the argument list is valid.

Post-processing

Use this function in conjunction with an operational function that queries an object's attributes:

1. Issue this function to fill in the argument list.
2. Issue the operational function (such as **CvClient::getObject**).

Purpose

Retrieves the error status code of an argument at a specified location in an argument list.

Syntax

CvStatus CvClient::Args::statusAt(
int *N* **);**

Description

statusAt retrieves the error status code of an argument at a specified location in an argument list.

N is the zero-based index of the argument in the argument list. If *N* is out of range of the argument list, the status -1 is returned.

Prerequisites

Use this function in conjunction with other utility functions that traverse the argument list:

1. Obtain the argument list by appropriate means, such as by creating a **CvClient::Args** class or by issuing an operational function (for example **CvClient::getObject**).
2. Issue **CvClient::Args::errorIndex** to determine if any argument in the argument list has an error status. If you use the returned value in another function, subtract 1 from the value.
3. Issue this function to retrieve the error status code of the argument.

Post-processing

None.

See Also

getStatus.

valueAt

Purpose

Retrieves the attribute value at a specified location in an argument list.

Syntax

const CvClient::Value& CvClient::Args::valueAt(
int *Index* **);**

Description

valueAt retrieves the attribute value at a specified location in an argument list. If the index is invalid, the function returns 0.

Index is the zero-based index to the argument list.

Notes

Use this function to read values sequentially from a **CvClient::Args** object.

Prerequisites

Obtain the argument list by appropriate means, such as by creating a **CvClient::Args** class or by issuing an operational function (for example **CvClient::getObject**).

To count the number of arguments in an argument list, issue **CvClient::Args::count**.

Post-processing

None.

See Also

idAt and **count**.

CvClient::IpAddr

CvClient::IpAddr 3-140

Purpose

Used to create IP addresses using either string or integer representations. The class **CvClient::IpAddr** is used in each of the following:

- Member functions of **CvClient::ObjectId**
- One of the types for **CvClient::Value**
- The host identifier in the **CvClient** constructor

The API provides several constructors for creating a **CvClient::IpAddr** class object:

Syntax

CvClient::IpAddr ( )

This null constructor declares an empty IP address.

Syntax

CvClient::IpAddr (const IpAddr&)

This copy constructor makes a copy of an IP address.

Syntax

CvClient::IpAddr (const char* *String*, **int len=-1);**

This constructor makes an IP address from a given string. This constructor invokes **gethostbyname** to obtain the integer IP address value. If **gethostbyname** fails, the address is set to 0.

String is the string that represents the IP address. If **len==-1**, use **strlen** to obtain the length of the address.

Syntax

CvClient::IpAddr (CvIpAddr)

This constructor makes an IP address from a CvIpAddr (unsigned 32-bit) variable.

Syntax

CvClient::IpAddr& operator = (const IpAddr&)

This is the assignment operator.

Syntax

operator CvIpAddr ( ) const

This operator returns an integer representation of an IP address (or 0 if no valid address has been set).

Syntax

operator const char* ( ) const

This operator returns a string representation of an IP address. Each field is printed in integer format, regardless of how the IP address was constructed.

Notes

The API requires you to specify an IP address in each of the following situations:

- When you specify an object ID to be acted on that contains an IP address (using member functions of **CvClient::ObjectId**).
- When you specify an IP address as a value (one of the types for **CvClient::Value**).
- When you specify the host identifier in the **CvClient** constructor.

CvSVCAddress

CvSVCAddress 3-143

Purpose

Used to create SVC addresses. SVC addresses are represented as strings that conform to the notation specified in SVC addressing. The class **CvSVCAddress** enforces the basic syntax rules of the seven different SVC address formats:

E.164native
DCCAESA
ICDAESA
E.164AESA
CustomAESA
DefaultRoute
UserPart

For information on the syntax rules of SVC addresses, refer to Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

The class is used in member functions of **CvClient::ObjectId**. For example, you specify an SVC address when you specify an object ID to be acted on that contains an SVC address (using member functions of **CvClient::ObjectId**).

The API provides several constructors for creating an **SVCAddress** class object. To read an SVC address, cast **CvSVCAddress** to a **char ***. The class has a **!** operator to verify if the **CvSVCAddress** represents a valid SVC address or not.

Syntax

CvSVCAddress (const char* *String* **);**

This constructor makes an SVC address from a given string. The string must conform to the convention used to specify SVC addresses.

String is the string that represents the SVC address. The string must conform to the convention used to specify SVC addresses.

Syntax

CvSVCAddress (int *addrFmt,* **const char*** *afiStr,* **const char*** *addrStr,* **int** *nBits* **);**

This constructor makes an SVC address from a series of input parameters.

addrFmt is the number that represents the format type of the SVC address. Valid values are as follows:

1 — E.164native
100 — DCCAESA
101 — ICDAESA
102 — E.164AESA
103 — CustomAESA
104 — DefaultRoute
105 — UserPart

afiStr is the AFI value (the first 2 characters of the SVC address prefix).

Standard AFI values are:

39 — DCCAESA
45 — E.164AESA
47 — ICDAESA

addrStr is the remainder of the SVC address prefix. For AESA addresses, if the address prefix is an odd number of characters, the Provisioning Server stuffs the last 4 bits of the last octet with zeros, thereby appending a zero to the address prefix.

nBits is the number of bits. This field is optional for all address formats except for DefaultRoute. For DefaultRoute, you must specify the number of bits as zero.

For other address formats, if you omit this field, the Provisioning Server calculates the value and appends it to the address prefix.

The algorithms used to calculate nBits values are presented in Table 3-5.

Table 3-5. Calculated nBits Values

Address Format Type	*addrFmt* Value	Calculated nBits Value
E.164native	1	string length of address prefix * 8
DCCAESA	2	(integral-part-of((string length of address prefix + 1)/2)*8)
ICDAESA	3	(integral-part-of((string length of address prefix + 1)/2)*8)
E.164AESA	4	(integral-part-of((string length of address prefix + 1)/2)*8)
CustomAESA	5	(integral-part-of((string length of address prefix + 1)/2)*8)
UserPart	7	56

Notes

In the following cases, the Provisioning Server performs a conversion of the address string:

- When you omit the nBits field, the Provisioning Server calculates and appends an nBits value to the prefix address.
- When an address prefix of an AESA address is an odd number of characters, the Provisioning Server appends a zero to the prefix address.

A converted string is equivalent to the original string. Either address string can be used in an operation.

A

Sample Code

This appendix presents commented source listing of a sample NavisXtend Provisioning client. Use it as guide for writing your own Provisioning client.

C Sample Code

```
/* This program demonstrates the use of the Provisioning Server C
API. */

#include "sampdflt.h"

#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#include "CvParamValues.H"
#include "ProvClient.H"

#define ASYNC 1
#define SYNC 0
#define MAX_ASYNC 10
#define LIST 1
```

```
#define NOTLIST 0
#define BAD_REQID 0xffffffff

static CvServerHandle serverHdl;

static char* switchIPA = SWITCH;
static int cardNumber = CARD;
static int pport = PPORT;
static int lport = LPORT;
static char countryPrefix[7] = COUNTRY_PREFIX;
static char addressPrefix[7] = ADDR_PREFIX;
static char netwideGroupAddress[11] = NETWIDE_GROUP_ADDR;
static int individualAddress1 = INDIVIDUAL_ADDR_1;
static int individualAddress2 = INDIVIDUAL_ADDR_2;

/*****************************************************************
*
*
* Subroutines to process responses for asynchronous requests
*
*****************************************************************
/

/* An array of this struct keeps track of outstanding requests; if a
 * request is async and a list, we want to mark that so that we can
 * collect additional responses.
 */
typedef struct
{
  CvReqId reqID;
  int isList;
} asyncRequest;
static asyncRequest requests[MAX_ASYNC];

/* Initializes the above array */
void asyncInit()
{
  int idx;
  for (idx=0; idx < MAX_ASYNC; idx++)
    requests[idx].reqID = BAD_REQID;
}
```

```
/* Add an asynch request to the array */
void addRequest (const CvReqId req, const int list) {
  int idx;
  for (idx=0; idx < MAX_ASYNC; idx++)
    if (requests[idx].reqID == BAD_REQID)
      break;
  if (idx == MAX_ASYNC) {
    printf ("Too many async requests\n");
    exit (1);
  }
  requests[idx].reqID = req;
  requests[idx].isList = list;
}

/* Remove an asynch request */
void removeRequest (const int i) {
  int idx;
  for (idx=i; idx < (MAX_ASYNC - 2); idx++) {
    requests[idx].reqID = requests[idx + 1].reqID;
    requests[idx].isList = requests[idx + 1].isList;
  };
  requests[MAX_ASYNC-1].reqID = BAD_REQID;
}


/*
 * If async requests are pending, call asyncWait to get the
 * responses
 */
void asyncWait ()
{
  while (requests[0].reqID != BAD_REQID)
    CvProcessEvents(-1);
}

/*
 * Callback function for async requests
 */
void response(CvObjectId obj_id, CvStatus stat,
              CvReqId RequestId, void* context, CvConstArgs args) {
  int idx;
  char *argsStr;
```

```
  printf("Response received to request %lu:\n", RequestId);
  printf("   Status: %i\n", (int) stat);

  for (idx=0; idx < MAX_ASYNC; idx++)
    if (requests[idx].reqID == RequestId)
      break;
  if (idx == MAX_ASYNC) {
    printf ("Unknown request id: %lu\n", RequestId);
    CvObjectIdPrint(stdout, obj_id);
    printf(" ");
    CvArgsPrint(stdout, args);
    printf("\n");
    return;
  }
  CvArgsPrint(stdout, args);
  printf("\n");

  switch (stat) {
  case cveNoError:
    if (requests[idx].isList)   /* More responses pending? */
      return;                   /* yes, keep this one in the list */
    break;
  case cveListDone:
    printf("List Done\n");
    break;
  default:
    printf("Error from server: %s\n", CvGetErrorMsg(serverHdl));
  }
  removeRequest(idx);
}

static CvStatus status;
static CvReqId reqID;
static CvArgs emptyArgs;

static CVCBH callBackFn = response;


/*****************************************************************
*
*
```

```
* Function to create various example SMDS addresses
*
*****************************************************************
/

const char* makeAddrString(CvObjectType type, int num)
{
  static char string[20];

  switch (type) {

  case CVT_SmdsLocalIndividualAddress: {
    static int count = 1;
    if ((num < 0) || (num > 99))
      num = count++;
    sprintf(string,"%s%s%02i%02i", countryPrefix, addressPrefix,
cardNumber,
num);
    return string;
  }

  case CVT_SmdsAddressPrefix:
    return addressPrefix;

  case CVT_SmdsNetwideGroupAddress: {
    sprintf(string,"%s%s", countryPrefix, netwideGroupAddress);
    return string;
  }
  default:
    return "";
  }
}

/*****************************************************************
*
*
* Function to create various example object id's
*
*****************************************************************
/

CvObjectId makeId(CvObjectType objType, int num)
```

```
{
  CvObjectId objid;

  switch (objType) {

  case CVT_SmdsNetwideGroupAddress:
    {
      objid = CvCreateNetworkIdFromString(switchIPA);
      objid =
CvCreateSmdsNetwideGroupAddrId(objid,makeAddrString(CVT_SmdsNetwideG
roupAddress,
0));
      break;
    }

  case CVT_SmdsLocalIndividualAddress:
    {
      objid = makeId(CVT_LPort,0);
      objid =
CvCreateSmdsLocalIndivAddrId(objid,makeAddrString(CVT_SmdsLocalIndiv
idualAddress
,num));
      break;
    }


  case CVT_SmdsAddressPrefix:
    {
      objid = makeId(CVT_Switch,0);
      objid =
CvCreateSmdsPrefixId(objid,makeAddrString(CVT_SmdsAddressPrefix,0));
      break;
    }

  case CVT_LPort:
    objid = CvCreateSwitchIdFromString(switchIPA);
    objid = CvCreateCardId(objid, cardNumber);
    objid = CvCreatePPortId(objid, pport);
    objid = CvCreateLPortId(objid, lport);
    break;

  case CVT_PPort:
```

```
    objid = CvCreateSwitchIdFromString(switchIPA);
    objid = CvCreateCardId(objid, cardNumber);
    objid = CvCreatePPortId(objid, pport);
    break;

  case CVT_Card:
    objid = CvCreateSwitchIdFromString(switchIPA);
    objid = CvCreateCardId(objid, cardNumber);
    break;

  case CVT_Switch:
    objid = CvCreateSwitchIdFromString(switchIPA);
    break;
  }

  return objid;
}

/*****************************************************************
*
*
*     Function to check the status of API requests.  For async
*     requests, also saves the request context to be matched later
*     with the response.
*
*****************************************************************
/

int checkStatus(const int async, const CvReqId reqid,
  const int list)
{
  switch (status) {
  case cveNoError:
    if (async)
      addRequest(reqid, list);
    return 1;
  case cveListDone:
    printf("List done.\n");
    break;
  default:
    printf("Error from server: %s\n", CvGetErrorMsg(serverHdl));
    break;
```

```
  }
  if (!async) {
    CvArgsPrint(stdout, CvGetResponseArgs(serverHdl));
    printf("\n");
  }
  status = cveNoError;
  return 0;
}

/*
 * listAndPrint -- performs a synchronous List Contained and prints
 * the results
 */

static void listAndPrint(CvServerHandle serverHdl,
                         const CvObjectId *objectId,
                         CvObjectType type,
                         CvConstArgs reqArgs)
{
  CvListHandle listHdl;
  CvArgs rspArgs;
  if (type == CVT_Unknown)
    listHdl = CvsListAllContainedObjects(serverHdl, objectId,
&status);
  else
    listHdl = CvsListContainedObjects(serverHdl, objectId, type,
&status,
reqArgs);
  if (checkStatus(SYNC,0,LIST)) {
    while (1) {
      CvStatus objStatus;
      char* argsStr;
      rspArgs = CvsNextObject(serverHdl, listHdl, &status);
      if (!checkStatus(SYNC,0,LIST)) {
        CvArgsFree(rspArgs);
        break;
      }
      printf( "list item: \n");
      CvObjectIdPrint(stdout, CvArgsGetObjectId(rspArgs,
CVA_ServerObject,
&objStatus));
      printf(" ");
```

```
      CvArgsPrint(stdout, rspArgs);
      printf("\n");
      CvArgsFree(rspArgs);
    }
  }
}

/*********************************************************************
*
*
* Function to add network objects
*
*********************************************************************
/

void doAdds()
{
  CvObjectId cardId;
  CvArgs args;
  CvArgs rspArgs;

  printf("# Doing Adds\n");

  /*  Use Modify Object to add a card (synchronous, va) */

  /* NOTE: Objects representing all possible card slots are
     created when the switch is defined, and are initialized to a
     card type of "empty".  Adding a card therefore takes the form
     of "modifying" its type from "empty" to the actual card type.
     Deleting a card likewise takes the form of "modifying" its
     type to "empty".
  */

  printf( "cvmodify \n");

  cardId = makeId(CVT_Card,0);
  args = CvArgsMakeVals(CVA_CardDefinedType, cardType2PortHssi,
                        CVA_CardRedundSlotId, (unsigned long) 0,
                        CVA_CardAdminStatus, cardAdminUp,
                        ENDARG);

  CvsModifyObject(serverHdl, &cardId, &status, args);
```

```
checkStatus(SYNC,0,NOTLIST);
CvArgsFree(args);

/*  Use Get Object to read back the card arguments (synchronous,
    arglist) */

args = CvArgsMake();
CvArgsSetNull(args, CVA_CardDefinedType);
CvArgsSetNull(args, CVA_CardRedundSlotId);
CvArgsSetNull(args, CVA_CardAdminStatus);

printf( "cvget \n");
CvArgsPrint(stdout, args);
printf("\n");

rspArgs = CvsGetObject(serverHdl, &cardId, &status, args);
CvArgsFree(args);
checkStatus(SYNC,0,NOTLIST);
printf("cvget results:\n");
CvArgsPrint(stdout, rspArgs);
printf("\n");
CvArgsFree(rspArgs);

/*  Add a Netwide Group Address (synchronous) */
{
  CvObjectId ngaId;
  ngaId = makeId(CVT_SmdsNetwideGroupAddress,0);

  printf( "cvadd NGA\n");
  CvsAddObject(serverHdl, &ngaId, &status, emptyArgs);
  checkStatus(SYNC,0,NOTLIST);
}

/*  Add a physical port (asynchronous, arglist) */
{
  CvObjectId pportId;
  pportId = makeId(CVT_PPort,0);

  args = CvArgsMake();
  CvArgsSetInt(args, CVA_PPortClockSpeed, 9474000);
  CvArgsSetInt(args, CVA_PPortAdminStatus, pPortAdminUp);
  CvArgsSetInt(args, CVA_PPortClockSource, clockSrcDce);
```

```
    printf( "cvadd pport\n");
    reqID = CvaAddObject(serverHdl, &pportId, callBackFn, (void*)
0, args);
    checkStatus(ASYNC,reqID,0);
    /* wait for completion before adding logical port */
    asyncWait();
    CvArgsFree(args);
  }

  /*  Add a logical port (synchronous, va) */
  {
    CvObjectId lportId;
    char lportName[50];
    lportId  = makeId(CVT_LPort,0);
    sprintf(lportName, "Smds %i-1-1", cardNumber);

    args = CvArgsMakeVals(CVA_LPortName, lportName,
                          CVA_LPortSmdsType,  (unsigned long)
                          smdsDxiSniDce,
                          CVA_LPortServiceType,  (unsigned long)
                          serviceSmds,
                          CVA_LPortBandwidth, 256000,
                          CVA_LPortBitStuffing,  (unsigned long)
                          bitStuffInvalid,
                          CVA_LPortFractionalDs0s,(unsigned long) 0,
                          CVA_LPortCongestionThreshold, 11,
                          CVA_LPortErrorPerMinThreshold, 20,
                          CVA_LPortAdminStatus, lPortAdminUp,
                          CVA_LPortIsBackup, (unsigned long) 0,
                          CVA_LPortBilling, 1,
                          CVA_LPortCrcChecking, (unsigned long) 0,
                          ENDARG);

    CvsAddObject(serverHdl, &lportId, &status, args);
    CvArgsFree(args);
    printf( "cvadd lport\n");
    checkStatus(SYNC, 0, NOTLIST);
  }

    {
      /* Add SMDS Address Prefix */
```

```
        CvObjectId prefixId = makeId(CVT_SmdsAddressPrefix, 0);
        printf( "cvadd \n" );
        CvsAddObject(serverHdl, &prefixId, &status, emptyArgs);
        checkStatus(SYNC, 0, NOTLIST);
    }

    {
      /* Add two Local Individual Addresses */
        CvObjectId iaId = makeId(CVT_SmdsLocalIndividualAddress,
                                 individualAddress1);
        printf("cvadd \n");
     CvsAddObject(serverHdl, &iaId, &status, emptyArgs );
        checkStatus(SYNC, 0, NOTLIST);

        iaId = makeId(CVT_SmdsLocalIndividualAddress,
                      individualAddress2);
        printf("cvadd \n");
     CvsAddObject(serverHdl, &iaId, &status, emptyArgs );
        checkStatus(SYNC, 0, NOTLIST);
    }


  /* List everything defined on the card (synchronous) */
  {
    CvArgs reqArgs;
    reqArgs = CvArgsMake();
    listAndPrint(serverHdl, &cardId, CVT_Unknown, reqArgs);
    CvArgsFree(reqArgs);
  }
}

/*****************************************************************
*
*
* Subroutine to add address members
*
*****************************************************************
/

void doAddMember()
{
  /* Add members to netwide group address (asynchronous) */
```

```
  CvObjectId addrId;
  CvObjectId memberId;

  printf("# Adding Members to NGA\n");

  addrId = makeId(CVT_SmdsNetwideGroupAddress, 0);
  memberId = makeId(CVT_SmdsLocalIndividualAddress,
                    individualAddress1);

  printf("cvaddmember \n");
  reqID = CvaAddMember(serverHdl, &addrId, &memberId, callBackFn,
(void*) 0);
  checkStatus(ASYNC,reqID,0);

  memberId = makeId(CVT_SmdsLocalIndividualAddress,
                    individualAddress2);
  printf("cvaddmember \n");
  reqID = CvaAddMember(serverHdl, &addrId, &memberId, callBackFn,
(void*) 0);
  checkStatus(ASYNC,reqID,0);

 /* wait for adds to complete */
  asyncWait();

  /* List contained addresses (asynchronous) */

  reqID = CvaListContainedObjects(serverHdl, &addrId,
                                  CVT_SmdsLocalIndividualAddress,
                                  callBackFn,
                                  (void*) 0, emptyArgs);
  checkStatus(ASYNC, reqID, LIST);

  /* NOTE: we don't bother to wait for completion here */

}

/*****************************************************************
*
*
* Subroutine to delete address members
*
```

```
*****************************************************************
/

void doDeleteMember()
{

  /* Delete members from netwide group address (synchronous) */

  CvObjectId ngaId;
  CvObjectId memberId;

  ngaId = makeId(CVT_SmdsNetwideGroupAddress, 0);
  memberId = makeId(CVT_SmdsLocalIndividualAddress,
  individualAddress1);
  printf("# Deleting Members\n");

  printf("cvdeletemember \n");

  CvsDeleteMember(serverHdl, &ngaId, &memberId, &status);
  checkStatus(SYNC, 0, NOTLIST);

  memberId = makeId(CVT_SmdsLocalIndividualAddress,
  individualAddress2);
  printf("cvdeletemember \n");

  CvsDeleteMember(serverHdl, &ngaId, &memberId, &status);
  checkStatus(SYNC, 0, NOTLIST);

  /* List contained addresses (synchronous) */

  {
    CvArgs reqArgs;

    reqArgs = CvArgsMake();
    printf("# Listing Members: list should be empty.\n");
    listAndPrint(serverHdl, &ngaId, CVT_SmdsLocalIndividualAddress,
                 reqArgs);
    CvArgsFree(reqArgs);
  }
}

/*****************************************************************
```

```
*
*
* Subroutine to delete network objects
*
*****************************************************************
/

void doDeletes()
{
  CvObjectId cardId;
  CvObjectId pportId;
  CvObjectId lportId;

  printf("# Doing SMDS Deletes\n");

  cardId = makeId(CVT_Card, 0);
  pportId = makeId(CVT_PPort, 0);
  lportId  = makeId(CVT_LPort, 0);

  /* Delete the local addresses */

    {
      /* delete the Local Individual Addresses */
        CvObjectId iaId = makeId(CVT_SmdsLocalIndividualAddress,
                                 individualAddress1);
        printf("cvdelete\n", &iaId, &status);
        CvsDeleteObject(serverHdl, &iaId, &status);
        checkStatus(SYNC, 0, NOTLIST);

        iaId = makeId(CVT_SmdsLocalIndividualAddress,
                      individualAddress2);
        printf("cvdelete \n", &iaId, &status);
        CvsDeleteObject(serverHdl, &iaId, &status);
        checkStatus(SYNC, 0, NOTLIST);
    }


    {
      /* delete SMDS Address Prefix */
        CvObjectId prefixId = makeId(CVT_SmdsAddressPrefix, 0);
        printf( "cvdelete \n" );
```

```
        CvsDeleteObject(serverHdl, &prefixId, &status);
        checkStatus(SYNC, 0, NOTLIST);
    }



  /* Delete LPort (asynchronous) */
  printf("cvdelete \n");
  reqID = CvaDeleteObject(serverHdl, &lportId, callBackFn, (void*)
0);
  checkStatus(ASYNC,reqID,0);

  /* wait for LPort deletion to complete */
  asyncWait();

  /* Note:  We can't delete PPorts. */

  /* List all objects within card (should be empty) (asynchronous)
  */
  printf("list: should be empty\n");
  reqID = CvaListAllContainedObjects(serverHdl, &cardId,
  callBackFn, (void*) 0);
  checkStatus(ASYNC, reqID, LIST);

  /* Response will be handled later */

  /* Delete Card (asynchronous) */
  {
    CvArgs args;

    args = CvArgsMake();
    CvArgsSetInt(args, CVA_CardDefinedType, cardTypeEmpty);

    printf("cvmodify \n");

    reqID = CvaModifyObject(serverHdl, &cardId, callBackFn,
                            (void*) 0, args);
    checkStatus(ASYNC, reqID, 0);
    CvArgsFree(args);
  }

  /* Get card status (asynchronous) */
```

```
  printf("cvget for deleted card\n");
  reqID = CvaGetObject(serverHdl, &cardId, callBackFn, (void*)
0,emptyArgs);
  checkStatus(ASYNC,reqID, 0);
  /* Response will be handled later */

  /* Delete the netwide group address (asynchronous) */

  {
    CvObjectId ngaId;

    ngaId = makeId(CVT_SmdsNetwideGroupAddress, 0);
    printf( "cvdelete NGA\n");
    reqID = CvaDeleteObject(serverHdl, &ngaId, callBackFn, (void*)
0);
    checkStatus(ASYNC, reqID, 0);
  }

  /* wait for list, get and NGA delete to complete */
  asyncWait();

}

/*********************************************************************
*
*
* Main program
*
*********************************************************************
/

int main(int argc, char* argv[])
{
  const char* hostname = HOSTNAME;
  CvIpPort rport = HOSTPORT;
  const char* username = USERNAME;
  const char* password = PASSWORD;
  int modifyType = MODTYPE;

  const char* tmpname;
  const char* modTypeStr;
```

```
int ap;

/* Process environment variables */

tmpname = getenv("CVTEST_HOST");
if (tmpname)
  hostname = tmpname;
if (getenv("CV_PSRV_USE_LOGINS")) {
  username = getenv("CV_PSRV_USERNAME");
  password = getenv("CV_PSRV_PASSWORD");
}
modTypeStr = getenv("CVENV_MOD_TYPE");
if (modTypeStr)
  modifyType = atoi(modTypeStr);

/* Process command line arguments */

for (ap = 1; ap < argc; ++ap) {
  if (strcmp(argv[ap], "-switch")==0) {
    strcpy(switchIPA, argv[++ap]);
  }
  else if (strcmp(argv[ap], "-card")==0) {
    cardNumber = atoi(argv[++ap]);
  }
  else if (strcmp(argv[ap], "-pport")==0) {
    pport = atoi(argv[++ap]);
  }
  else if (strcmp(argv[ap], "-lport")==0) {
    lport = atoi(argv[++ap]);
  }
  else if (strcmp(argv[ap], "-country")==0) {
    strcpy(countryPrefix, argv[++ap]);
  }
  else if (strcmp(argv[ap], "-nga")==0) {
    strcpy(netwideGroupAddress, argv[++ap]);
  }
  else if (strcmp(argv[ap], "-addrprefix")==0) {
    strcpy(addressPrefix, argv[++ap]);
  }
  else if (strcmp(argv[ap], "-ia1")==0) {
    individualAddress1 = atoi(argv[++ap]);
  }
```

```
  else if (strcmp(argv[ap], "-ia2")==0) {
    individualAddress2 = atoi(argv[++ap]);
  }
  else if (strcmp(argv[ap], "-host")==0) {
    hostname = argv[++ap];
  }
  else if (strcmp(argv[ap], "-rport")==0) {
    rport = atoi(argv[++ap]);
  }
  else if (strcmp(argv[ap], "-user")==0) {
    username = argv[++ap];
  }
  else if (strcmp(argv[ap], "-password")==0) {
    password = argv[++ap];
  }
  else if (strcmp(argv[ap], "-modtype")==0) {
    modifyType = atoi(argv[++ap]);
  }
  else {
    printf("Undefined option %s\n", argv[ap]);
    exit(1);
  }
}

  /* Initialize the session with the Provisioning Server */

  emptyArgs = CvArgsMake(); /* initialize the empty Args
                               structure */

  asyncInit(); /* Initialize the async request array */
  serverHdl = CvConnect(hostname, rport, username, password);

  CvSetModifyType(serverHdl, (CvModifyType) modifyType);

  /* Execute examples */

  doAdds();
  doAddMember();
  doDeleteMember();
  doDeletes();

  /* Close session and exit */
```

```
    CvClose(serverHdl); /* close session */
    CvArgsFree(emptyArgs); /* delete the empty Args structure */
    printf("Done\n");
}
```

C++ Sample Code

The C++ sample code is contained in two files:

- ProvSample.C contains sample code for SMDS objects.
- ProvSample2.C contains sample code for most other objects.

The following sections present the C++ sample code.

Sample Code for SMDS Objects

```
// This program demonstrates the use of the Provisioning Server C++
// API.

#include “sampdflt.h”

#include <iostream.h>
#include <strstream.h>
#include <stdlib.h>
#include <string.h>

#include “ProvClient.H”
#include “CvParamValues.H”

#define CERR(TEXT)   do {cerr << TEXT << endl;} while (0)

#define ASYNC 1
#define MAX_ASYNC 10
#define LIST 1
#define BAD_REQID 0xffffffff

static char switchIPA[20] = SWITCH;
static int cardNumber = CARD;
```

```
static int pport = PPORT;
static int lport = LPORT;
static char countryPrefix[7] = COUNTRY_PREFIX;
static char addressPrefix[7] = ADDR_PREFIX;
static char netwideGroupAddress[11] = NETWIDE_GROUP_ADDR;
static int individualAddress1 = INDIVIDUAL_ADDR_1;
static int individualAddress2 = INDIVIDUAL_ADDR_2;

static const CvClient::Args emptyArgs;

//
//    SampleApp contains all the code needed for the sample
//    application, including async callbacks.  Because it's derived
//    from CvClient, its member functions don't need to use the
//    "CvClient::" prefix for any of the CvClient functions or
//    classes.
//

class SampleApp: public CvClient, public CvClient::Callback {
public:
    SampleApp (IpAddr  host,
               CvIpPort  remoteport,
               const char* userName = 0,
               const char* password = 0) ;// constructor

    void asyncWait ();                    // waits for async
                                          // responses to arrive

    void doAdds();                        // add some objects
    void doAddMember();                   // add some members to
                                          // Netwide Group Addresses
    void doDeleteMember();                // Delete these members
    void doDeletes();                     // delete all created
                                          // objects

private:

    typedef struct
    {
        CvReqId reqID;
        int isList;
    } asyncRequest;
```

```
    asyncRequest requests[MAX_ASYNC];

    void addRequest (const CvReqId req, const int list = 0);
    void removeRequest (const int i);
    int checkStatus(const int async = 0, const CvReqId reqid = 0,
  const int list = 0);

    void response(const CvObjectId &obj_id, ApiStatus stat,
// handles async responses
                  CvReqId RequestId, const Args *args);

    ApiStatus status_;           // Holds the status of each request
    int numReqs_;                // Number of outstanding requests
};



//
// SampleApp::SampleApp
//
SampleApp::SampleApp (IpAddr  host,
                      CvIpPort  remoteport,
                      const char* userName,
                      const char* password)
    : CvClient(host, remoteport, userName, password), numReqs_(0)
{
    // Initialize
    for (int idx=0; idx < MAX_ASYNC; idx++)
        requests[idx].reqID = BAD_REQID;
}




//
// SampleApp::addRequest()
//

void SampleApp::addRequest (const CvReqId req, const int list)
{
    //  Find a free slot
```

```
    for (int idx=0; idx < MAX_ASYNC; idx++)
        if (requests[idx].reqID == BAD_REQID)
            break;

    if (idx == MAX_ASYNC) {
        CERR ("Too many async requests");
        exit (1);
    }

    requests[idx].reqID = req;
    requests[idx].isList = list;
    numReqs_++;
}


//
// SampleApp::removeRequest()
//
void SampleApp::removeRequest (const int i)
{
    CvReqId req = requests[i].reqID;

    // Move later records down to fill in the array
    for (int idx=i; idx < (MAX_ASYNC - 2); idx++) {
        requests[idx] = requests[idx + 1];
    };

    requests[MAX_ASYNC-1].reqID = BAD_REQID; // last one in the
array is now empty
    numReqs_--;
}


//
// SampleApp::checkStatus() -- checks the status of API requests.
// Also updates the SampleApp's record of outstanding asynchronous
// requests.
//

int SampleApp::checkStatus(const int async, const CvReqId reqid,
const int list)
{
```

```
    switch (status_) {
    case cveNoError:
        if (async) {
            addRequest(reqid, list);
// Remember this request for when response arrives.
            CERR("Async request ID is " << reqid);
        }
        else
            CERR("OK");
        return 1;
    default:
        CERR("Error on request: " << reqid << ".  Status: " <<
status_ << " (as int:)" << (int) status_);
        if (!async)
            CERR(" Response arguments from server: " <<
getResponseArgs());
        break;
    case cveListDone:
        CERR("End of list.");
        break;
    }
    status_ = cveNoError;
    return 0;
}


//
// SampleApp::asyncWait
//
void SampleApp::asyncWait ()
{
   while (requests[0].reqID != BAD_REQID) // while any
                                          // outstanding requests
                                          // remain
        processEvents();                  // process the
                                          // responses (or
                                          // timeouts) one at a time
}


//
// SampleApp::response
```

```
//

void SampleApp::response(const CvObjectId &obj_id, ApiStatus stat,
                         CvReqId RequestId, const Args *args)
{
    CERR("Response received to request " << RequestId <<
         ": " << endl << "   Status: " << (int) stat << endl <<
         "   Object: " << obj_id << endl << *args);

    // Find the record of this request.
    for (int idx=0; idx < MAX_ASYNC; idx++)
        if (requests[idx].reqID == RequestId)
            break;

    if (idx == MAX_ASYNC) {
        CERR ("*** Unknown request id: " << RequestId);
        return;
    }

    // check the status from the request

    switch (stat) {
    case cveNoError:
        if (requests[idx].isList)        // for List requests
            return;                      // wait for more responses
                                         // to arrive
        break;
    case cveListDone:
        CERR("End of list.");
        break;
    default:
        if (getErrorMsg())               // Print error message
                                         // from this request
            CERR("   Error: " << getErrorMsg());
    }

    // delete the record of this request
    removeRequest(idx);
}


//
```

```
// Subroutine to create various example SMDS addresses.
//

const char* makeAddrString(CvObjectType
type=CVT_SmdsLocalIndividualAddress, int num = -1)
{
    static char string[20];
    ostrstream addrStream;

    switch (type) {
    case CVT_SmdsLocalIndividualAddress: {
        static int count = 1;
        if ((num < 0) || (num > 99))
            num = count++;

        addrStream << countryPrefix << addressPrefix << (cardNumber
< 10 ? "0" : "") << cardNumber <<
            (num < 10 ? "0" : "") << num;
        strcpy(string, addrStream.str());
        delete addrStream.str();
        return string;
    }

    case CVT_SmdsAddressPrefix:
        return addressPrefix;

    case CVT_SmdsNetwideGroupAddress: {
        addrStream << countryPrefix << netwideGroupAddress;
        strcpy(string, addrStream.str());
        delete addrStream.str();
        return string;
    }
    default:
        return "";
    }
}

//
// Subroutine to create various example object id's
//

CvClient::ObjectId makeId(CvObjectType objType, int num=-1)
```

```
{
    CvClient::ObjectId objid;

    switch (objType) {

    case CVT_SmdsNetwideGroupAddress:
     {

objid.setE164Address(makeAddrString(CVT_SmdsNetwideGroupAddress));
// makes a CvE164Address implicitly
            objid.setNetNumber(switchIPA);
            break;
        }

    case CVT_SmdsLocalIndividualAddress:
     {
            objid = makeId(CVT_LPort);
            CvE164Address addr(makeAddrString(objType, num));
// here, we make the CvE164Address explicitly
            objid.setE164Address(addr);
            break;
        }

    case CVT_SmdsAddressPrefix:
     {
            objid = makeId(CVT_Switch);
            CvE164Address
addr(makeAddrString(CVT_SmdsAddressPrefix));
            objid.setE164Address(addr);
            break;
        }

    case CVT_LPort:
        objid.set(CVT_LPort, lport);
        // Fall through

    case CVT_PPort:
        objid.set(CVT_PPort, pport);
        // Fall through

    case CVT_Card:
        objid.set(CVT_Card, cardNumber);
```

```
        // Fall through

    case CVT_Switch:
        objid.setIpAddr(switchIPA);
        break;
    }

    objid.setType(objType);
    return objid;
}


//
// SampleApp::doAdds
//

void SampleApp::doAdds()
{
    CERR("# Doing SMDS Adds");
    ObjectId cardId = makeId(CVT_Card);

    //  Use Modify Object to add a card (synchronous)

    // NOTE: Objects representing all possible card slots are
    // created when the switch is defined, and are initialized to a
    // card type of "empty".  Adding a card therefore takes the form
    // of "modifying" its type from "empty" to the actual card
    // type.  Deleting a card likewise takes the form of "modifying"
    // its type to "empty".

    Args args;
    args[CVA_CardDefinedType] =  cardType2PortHssi;
    args[CVA_CardRedundSlotId] =  (unsigned long) 0;
    args[CVA_CardAdminStatus] = cardAdminUp;

    CERR(endl << "cvmodify " << cardId << ' ' << args);

    modifyObject(cardId, args, &status_);
    checkStatus();

    //  Use Get Object to read back the card arguments (synchronous)
    args.set(CVA_CardDefinedType);
```

```
args.set(CVA_CardRedundSlotId);
args.set(CVA_CardAdminStatus);

CERR(endl << “cvget “ << cardId << ‘ ‘ << args);

Args rspArgs = getObject(cardId, args, &status_);
if (checkStatus())
 CERR(endl << “cvget results: “ << cardId << ‘ ‘ << rspArgs);

//  Add a Netwide Group Address (synchronous)
{
    ObjectId ngaId = makeId(CVT_SmdsNetwideGroupAddress);

    CERR(endl << “cvadd “ << ngaId);
    Args emptyArgs;
 addObject(ngaId, emptyArgs, &status_);
    checkStatus();
}

//  Add a physical port (asynchronous)
{
    ObjectId pportId = makeId(CVT_PPort);
    Args args;

    args[CVA_PPortClockSpeed] = 9474000;
    args[CVA_PPortAdminStatus] = pPortAdminUp;
    args[CVA_PPortClockSource] = clockSrcDce;

    CERR(endl << “cvadd “ << pportId << “ “ << args);
    CvReqId reqID = addObject(pportId, this, args);
    checkStatus(ASYNC,reqID);
    /* wait for completion before adding logical port */
    this->asyncWait();
}

//  Add a logical port (synchronous)
{
    ObjectId lportId  = makeId(CVT_LPort);
    ostrstream lportName;
    lportName << “Smds “ << cardNumber << “-1-1”;
    Args lportArgs(CVA_LPortName, lportName.str(),
                   CVA_LPortSmdsType,  (unsigned long)
```

```
                    smdsDxiSniDce,
                    CVA_LPortServiceType,  (unsigned long)
                    serviceSmds,
                    CVA_LPortBandwidth, 256000,
                    CVA_LPortBitStuffing,  (unsigned long)
                    bitStuffInvalid,
                    CVA_LPortFractionalDs0s ,  (unsigned long) 0,
                    CVA_LPortCongestionThreshold, 11,
                    CVA_LPortErrorPerMinThreshold, 20,
                    CVA_LPortAdminStatus, lPortAdminUp);
    lportArgs[CVA_LPortIsBackup] = (unsigned long) 0;
    lportArgs[CVA_LPortBilling] = cvBillingDisable;
    lportArgs[CVA_LPortCrcChecking] = cvCrc16;

    delete lportName.str();
    CERR(endl << "cvadd " << lportId << " " << lportArgs);
    addObject(lportId, lportArgs, &status_);
    checkStatus();
}

{
    // Add SMDS Address Prefix
    ObjectId prefixId = makeId(CVT_SmdsAddressPrefix);
    CERR( "cvadd " << prefixId);
    addObject(prefixId, emptyArgs, &status_);
    checkStatus();
}


{
    // Add two Local Individual Addresses
    ObjectId iaId = makeId(CVT_SmdsLocalIndividualAddress,
                           individualAddress1);
    CERR(endl << "cvadd " << iaId);
 addObject(iaId, emptyArgs, &status_);
    checkStatus();

    iaId = makeId(CVT_SmdsLocalIndividualAddress,
                  individualAddress2);
    CERR(endl << "cvadd " << iaId);
 addObject(iaId, emptyArgs, &status_);
    checkStatus();
```

```
    }


    // List everything defined on the card (synchronous)
    CvListHandle listPtr;
    Args listargs;

    CERR(endl << "cvlistallcontained " << cardId);
    listPtr = listContainedObjects(cardId, listargs, &status_);
    if (checkStatus()) {
        listargs = nextObject(listPtr, &status_);
        while (checkStatus()) {
            CERR( "list item: " << listargs);
            listargs = nextObject(listPtr, &status_);
        }
    }
}

//
// doAddMember -- add address members to a Netwide Group Address
//

void SampleApp::doAddMember()
{
    /* Add members to netwide group address (asynchronous) */

    CERR("# Adding Members to NGA");

    ObjectId addrId = makeId(CVT_SmdsNetwideGroupAddress);

    ObjectId memberId = makeId(CVT_SmdsLocalIndividualAddress,
                                          individualAddress1);

    CERR(endl << "cvaddmember " << addrId << ' ' << memberId);
    CvReqId reqID = addMember(addrId, memberId, this);
    checkStatus(ASYNC,reqID);

    memberId = makeId(CVT_SmdsLocalIndividualAddress,
                      individualAddress2);
    CERR(endl << "cvaddmember " << addrId << ' ' << memberId);
    reqID = addMember(addrId, memberId, this);
    checkStatus(ASYNC,reqID);
```

```
    // wait for adds to complete
    asyncWait();

    // List contained addresses (asynchronous)

    reqID = listContainedObjects(addrId,
CVT_SmdsLocalIndividualAddress,
                                          this, emptyArgs);
    checkStatus(ASYNC, reqID, LIST);

    // NOTE: we don't bother to wait for completion here.  Responses
    // will be picked up in doDeleteMember or in main()
}


//
// doDeleteMember -- delete addresses from a Netwide Group Address
//

void SampleApp::doDeleteMember()
{

    // Delete members from netwide group address (synchronous)

    ObjectId ngaId = makeId(CVT_SmdsNetwideGroupAddress);
    ObjectId memberId = makeId(CVT_SmdsLocalIndividualAddress,
                                        individualAddress1);

    CERR("# Deleting Members from " << ngaId);
    CERR(endl << "cvdeletemember " << ngaId << ' ' << memberId);
    deleteMember(ngaId, memberId, &status_);
    checkStatus();

    memberId = makeId(CVT_SmdsLocalIndividualAddress,
                      individualAddress2);
    CERR(endl << "cvdeletemember " << ngaId << ' ' << memberId);
    deleteMember(ngaId, memberId, &status_);
    checkStatus();

    // List contained addresses (synchronous)
```

```
    CvListHandle listPtr;
    Args listargs;

    CERR("# Listing Members of " << ngaId << ": list should be
         empty.");
    listPtr = listContainedObjects(ngaId,
CVT_SmdsLocalIndividualAddress,
                                                   listargs, &status_);
    if (checkStatus()) {
        listargs = nextObject(listPtr, &status_);
        while (checkStatus()) {
            CERR( "list item: " << listargs);
            listargs = nextObject(listPtr, &status_);
        }
    }
}

//
// doDeletes -- delete all objects created in doAdd
//

void SampleApp::doDeletes()
{
    CERR("# Doing SMDS Deletes");

    ObjectId cardId = makeId(CVT_Card);
    ObjectId pportId = makeId(CVT_PPort);
    ObjectId lportId  = makeId(CVT_LPort);

    // delete Local Addresses
    ObjectId memberId = makeId(CVT_SmdsLocalIndividualAddress,
                               individualAddress1);
    CERR(endl << "cvdelete " << memberId);
    deleteObject(memberId, &status_);
    checkStatus();

    memberId = makeId(CVT_SmdsLocalIndividualAddress,
                      individualAddress2);
    CERR(endl << "cvdelete " << memberId);
    deleteObject(memberId, &status_);
    checkStatus();
```

```
// delete Address Prefix
ObjectId prefixId = makeId(CVT_SmdsAddressPrefix);
CERR( “cvdelete “ << prefixId);
deleteObject(prefixId, &status_);
checkStatus();

// Delete LPort (asynchronous)
CERR(endl << “cvdelete “ << lportId);
CvReqId reqID = deleteObject(lportId, this);
checkStatus(ASYNC,reqID);

// wait for LPort deletion to complete
asyncWait();

// NOTE:  We can’t delete PPorts.  Skip to the card.

/// List all objects within card (should be empty)
// (asynchronous)
CERR(“list “ << cardId << “: should be empty”);
reqID = listContainedObjects(cardId, this, emptyArgs);
checkStatus(ASYNC, reqID, LIST);

// Delete Card (asynchronous)
Args args;
args[CVA_CardDefinedType] =  (unsigned long) cardTypeEmpty;

CERR(endl << “cvmodify “ << cardId << ‘ ‘ << args);

reqID = modifyObject(cardId, this, args);
checkStatus(ASYNC, reqID);

// Get card status (asynchronous)

CERR(endl << “cvget for deleted card”);
reqID = getObject(cardId, this, args);
checkStatus(ASYNC,reqID);


// Delete the netwide group address (asynchronous)

ObjectId ngaId = makeId(CVT_SmdsNetwideGroupAddress);
```

```
    CERR(endl << "cvdelete " << ngaId);
    reqID = deleteObject(ngaId, this);
    checkStatus(ASYNC,reqID);

    // wait for get and NGA delete to complete
    asyncWait();

}

//
// main
//

void main(int argc, char* argv[])
{
    const char* hostname = HOSTNAME;
    CvIpPort rport = HOSTPORT;
    const char* username = USERNAME;
    const char* password = PASSWORD;
    int modifyType = MODTYPE;
    int addFlag = 0, addMemberFlag = 0, deleteMemberFlag = 0,
    deleteFlag = 0;

    /* Process environment variables */

    const char* tmpname = getenv("CVTEST_HOST");
    if (tmpname)
        hostname = tmpname;
    if (getenv("CV_PSRV_USE_LOGINS")) {
        username = getenv("CV_PSRV_USERNAME");
        password = getenv("CV_PSRV_PASSWORD");
    }
    const char* modTypeStr = getenv("CVENV_MOD_TYPE");
    if (modTypeStr)
     modifyType = atoi(modTypeStr);

    /* Process command line arguments */

    for (int ap = 1; ap < argc; ++ap) {
        if (strcmp(argv[ap], "-switch")==0) {
            strcpy(switchIPA, argv[++ap]);
        }
```

```
    else if (strcmp(argv[ap], "-card")==0) {
cardNumber = atoi(argv[++ap]);
    }
    else if (strcmp(argv[ap], "-pport")==0) {
        pport = atoi(argv[++ap]);
    }
    else if (strcmp(argv[ap], "-lport")==0) {
        lport = atoi(argv[++ap]);
    }
    else if (strcmp(argv[ap], "-country")==0) {
        strcpy(countryPrefix, argv[++ap]);
    }
    else if (strcmp(argv[ap], "-nga")==0) {
        strcpy(netwideGroupAddress, argv[++ap]);
    }
    else if (strcmp(argv[ap], "-addrprefix")==0) {
        strcpy(addressPrefix, argv[++ap]);
    }
    else if (strcmp(argv[ap], "-ia1")==0) {
        individualAddress1 = atoi(argv[++ap]);
    }
    else if (strcmp(argv[ap], "-ia2")==0) {
        individualAddress2 = atoi(argv[++ap]);
    }
    else if (strcmp(argv[ap], "-host")==0) {
        hostname = argv[++ap];
    }
    else if (strcmp(argv[ap], "-rport")==0) {
rport = atoi(argv[++ap]);
    }
    else if (strcmp(argv[ap], "-user")==0) {
        username = argv[++ap];
    }
    else if (strcmp(argv[ap], "-password")==0) {
        password = argv[++ap];
    }
    else if (strcmp(argv[ap], "-modtype")==0) {
        modifyType = atoi(argv[++ap]);
    }
    else if (strcmp(argv[ap], "-add")==0) {
        addFlag = 1;
    }
```

```
    else if (strcmp(argv[ap], "-addmember")==0) {
        addMemberFlag = 1;
    }
    else if (strcmp(argv[ap], "-deletemember")==0) {
        deleteMemberFlag = 1;
    }
    else if (strcmp(argv[ap], "-delete")==0) {
        deleteFlag = 1;
    }
    else {
        CERR("Undefined option " << argv[ap]);
        exit(1);
    }
}

/* Initialize the session with the Provisioning Server */

SampleApp*  myApp = new SampleApp(hostname, rport, username,
                  password); // open session

myApp->setModifyType((CvModifyType) modifyType);

/* Execute examples */

if (addFlag)
    myApp->doAdds();
if (addMemberFlag)           // this leaves some outstanding
                             // requests, which may be
    myApp->doAddMember();      // picked up in doDeleteMember...
if (deleteMemberFlag)
    myApp->doDeleteMember();
else if (addMemberFlag)        // but if doDeleteMember isn't
                               // called, then
    myApp->asyncWait();        // catch up on responses from
                               // doAddMember()
if (deleteFlag)
    myApp->doDeletes();

/* Close session and exit */

delete myApp;
CERR("Done");
```

```
}
```

Sample Code for Other Objects

```
////////////////////////////////////////////////////////////////////
/
//
//    Ascend Communications, Inc.
//    Core Switching Division - NMS
//    Copyright (c) 1997, All Rights Reserved
//
//    DISCLAIMER:
//
//    The source code example presented in this file is for
//    illustration only. Although the customers may compile it and
//    execute it against the provisioning server, it is not
//    foolproof and Ascend Communications Inc. is not liable for any
//    problems/damages whatsoever.
//
////////////////////////////////////////////////////////////////////
/

////////////////////////////////////////////////////////////////////
/
//
//    ASSUMPTIONS:
//
//   The main() program has been written with the switch IP address,
//   slot numbers, pportIDs, lportIds, VPIs, VCIs and all
//   referential names hardcoded. It is expected the customers will
//   review the code and make the necessary changes to the input
//   before using the program.
//
////////////////////////////////////////////////////////////////////
/

#include <iostream.h>
#include <strstream.h>
#include “ProvClient.H”
#include “CvParamValues.H”
```

```
#define MAXLINE 255

extern CvClient Client;  // define Client as external
CvClient Client;         // declare a CvClient session

// Define a callback handler. A client making an asynchronous
// request has to provide such a class which is derived from a
// PS-API class called CvClient::Callback. This feature is
// illustrated in the addCircuit() function.

class CBsetHandler: public CvClient::Callback
{
private:

    char text[20];

public:

    void response( const CvClient::ObjectId&,
                   CvClient::ApiStatus,
                   CvReqId reqId,
                   const CvClient::Args* );

    void Msg(char* x){strcpy(text,x);}
};

// The call back function defined for the call back handler
void CBsetHandler::response( const CvClient::ObjectId& object,
                             CvClient::ApiStatus status,
                             CvReqId reqId, const CvClient::Args*
setArgs )
{
    if(status != cveNoError)
    {
       cout << endl << text << " " << object << " "  << *setArgs
            << " *** " << Client.getErrorMsg() << endl;
    }
    else
    {
       cout << endl << text << " " << object << " was added" <<
endl;
    }
```

```
}

////////////////////////////////////////////////////////////////////////
/
//
// FUNCTION
//
//    addLPort() - a function that adds an lport
//
// SYNOPSIS
//
//   addLPort(const char *lportId, const char *name)
//
//   “lportId” - is a string representation of the lport being added
//                containing its parent ids.
//
//    “name” - is the user chosen name being givent to the lport.
//
// DESCRIPTION
//
//    This function adds and lport, given the name of the lport and
//    hierarchical ID of its parent(s), provided as a string.
//    For example: the lport’s hierarchical identification would be:
//
//    switch.<IP>.card.<slotID>.pport.<pportID>.lport.<lportID>
//
//    The parent hierarchy is parsed by the API function
//    CvParseObjectId() in order to create an lport object which
//    would then be used by the rest of the API functions. Arguments
//    associated with the lport are also set based on the type of
//    lport being created.
//
//    The same function is used to add virtual UNI lports as well as
//    other lports
//
//    This function sends the add request to the server
//    synchronously.
//
// RETURNS
//
//    Integer.
//
```

```
// ERRORS
//
//    Prints a message to the stdout indicating success or failure.
In the
//    latter case, a message string identifying the error is also
//    printed.
//
// CAVEATS
//
//    Assumes that the virtual UNI lport names begin with the
//    character, ‘v’. Additional processing is performed based on
//    this checking.
//
// SEE ALSO
//
//    ProvClient.H and CvParamValues.H
//
////////////////////////////////////////////////////////////////////
/

int addLPort(const char *lportId, const char *name)
{
    // Argument list used to create the lports
    CvClient::Args setArgs;
    // Argument list that will store values after a get request
    CvClient::Args getArgs;
    CvClient::ObjectId lport;
    // initiate apiStatus to no error
    CvClient::ApiStatus apiStatus = cveNoError;

    lport = CvParseObjectId(lportId);
    if(lport.type() == CVT_Unknown)
    {
        cout << lportId << “ is not a valid object” << endl;
        return 0;
    }

    // creating the list of attribute/value pairs to used add the
    // lports
    setArgs[CVA_LPortName] = name;
    setArgs[CVA_LPortServiceType] = serviceAtm;
```

```
    if (strstr(name,"atmUniDce"))
    {
        setArgs[CVA_LPortAtmType] = atmUniDce;
    }
    else
    {
        setArgs[CVA_LPortAtmType] = atmUniDte;
    }

    setArgs[CVA_LPortAdminStatus] = lPortAdminUp;
    setArgs[CVA_LPortBandwidth] = 1536000;
    setArgs[CVA_LPortRmCellGen] = 1;
    if(strstr(name,"v"))
    {
        setArgs[CVA_LPortSvcVpiMin] = 2;
        setArgs[CVA_LPortSvcVpiMax] = 15;
        setArgs[CVA_LPortSvcVciMin] = 32;
        setArgs[CVA_LPortSvcVciMax] = 1023;
    }

    Client.addObject(lport,setArgs,&apiStatus);

    // if apiStatus is not 0, there was an error
    if(apiStatus != cveNoError)
    {
        // print out the error message
        cout << endl << "cvadd " << lport << " " << setArgs
             << " *** " << Client.getErrorMsg() << endl;
    }
    else
    {
        cout << endl << "cvadd  " << lport << " was added" << endl;
    }
}

////////////////////////////////////////////////////////////////////
/
//
// FUNCTION
//
//   seeLPorts() - a function to list the lports using listContained
//
```

```
// SYNOPSIS
//
//    seeLPorts( const char *parentId )
//
//    “parentId” - is the string representation of the parent,
//                 example: switchId
//
// DESCRIPTION
//
//   This function shows the list of lports that are present under
//   a parent using the listContained operation. The hierarchical
//   ID of its parent (example: switch) is provided as a string.
//   For example: the switch’s hierarchical identification would be:
//
//    switch.<IP>
//
//    The parent hierarchy is parsed by the API function
//    CvParseObjectId() in order to create a parent object which
//    would then be used by the rest of the API functions. Arguments
//    that have to be listed for the lport (child object) are also
//    specified.
//
//    Unlike other operations (add, modify etc), listContained is a
//    not a one shot operation. As such the client would have to
//    call the nextObject() function in a loop to list all the child
//    objects of a parent.
//
//    This function sends the request to the server synchronously.
//
// RETURNS
//
//    Integer.
//
// ERRORS
//
//    The results are displayed in the case of success only.
//
// CAVEATS
//
//    None.
//
////////////////////////////////////////////////////////////////////
```

```
/

int seeLPorts( const char *parentId )
{
   // Argument list that will store values after a get request
    CvClient::Args getArgs;
   // declare object to be operated on. In this case it is a switch.
    CvClient::ObjectId object;

   // Initiate apiStatus to no error
    CvClient::ApiStatus apiStatus = cveNoError;
    CvListHandle listHandle;

    cout << endl << "*** List of LPorts that were created" << endl
<< endl;

    object = CvParseObjectId ( parentId );
    if(object.type() == CVT_Unknown)
    {
        cout << parentId << " is not a valid parent object" << endl;
        return 0;
    }

    // Generating an argument list of lport attributes with their
    // values set to NULL
    getArgs.set(CVA_LPortName);
    getArgs.set(CVA_LPortServiceType);
    getArgs.set(CVA_LPortFrameRelayType);
    getArgs.set(CVA_LPortAdminStatus);
    getArgs.set(CVA_LPortDceErrorThreshold);
    getArgs.set(CVA_LPortDceEventCount);
    getArgs.set(CVA_LPortDceVerifTimer);
    getArgs.set(CVA_LPortCongestionThreshold);
    getArgs.set(CVA_LPortFractionalDs0s);
    getArgs.set(CVA_LPortLinkMgmtProtocol);
    getArgs.set(CVA_LPortLmiDelay);
    getArgs.set(CVA_LPortErrorPerMinThreshold);
    getArgs.set(CVA_LPortBitStuffing);


    // Get the list of lports defined on the switch with the values
    // for the argument list
```

```
    listHandle = Client.listContainedObjects(object, // switch
                                                      // (parent)
                                            CVT_LPort,// lport
                                                      // (child)
                                            getArgs,  // child's
                                                      // args
                                            &apiStatus);

   // set the argument list to the first lport/argument list in the
   // list of lports
    getArgs = Client.nextObject(listHandle,&apiStatus);

   // When we have reached the end of the list apiStatus will return
   // cveListDone

    while(apiStatus == cveNoError)
    {
        // print out the object (lport)  with the list of arguments
        cout << getArgs[CVA_ServerObject] << " " << getArgs << endl
<< endl;
        // set the argument list to the next lport/argument list in
        // the list of lports
        getArgs = Client.nextObject(listHandle,&apiStatus);
    }
}

////////////////////////////////////////////////////////////////////
/
//
// FUNCTION
//
//     addCircuit() - a function that adds a circuit given the
//     endpoints
//
// SYNOPSIS
//
//    addCircuit(const char *ep1, const char*ep2, int vpi, int vci,
//               conat char *cktName)
//
//    "ep1" -  identifies endpoint1
//
//    "ep2" - identifies endpoint2
```

```
//
//    “vpi” - identifies the vpi to be used for the first endpoint.
//     vpi for the second endpoint is just an increment by 1.
//
//    “vci” - identifies the vci to be used for the first endpoint.
//     vci for the second endpoint is just an increment by 1.
//
//    “cktName” - is the name of the circuit being created.
//
// DESCRIPTION
//
//    This function adds a circuit given the two endpoints, their
//    vpi/vci info and the circuit name. Endpoints could be regular
//    or ServiceName based.
//
//    The hierarchical representation of the endpoints is provided
//    as a string and CvParseObjectId() is used to convert them from
//    the string repersentation to the object representation.
//
//    The configuration arguments for the circuit are also set to
//    appropriate values. The request from the client to the server
//    is sent asynchronously. The callback class defined at the
//    beginning of this file is used for this purpose.
//
//    The circuit could be added using either of the two endpoints.
//    This example uses endpoint1.
//
// RETURNS
//
//    Integer.
//
// ERRORS
//
//    No error checking is explicitly done to verify the results.
//
// CAVEATS
//
//    None.
//
////////////////////////////////////////////////////////////////////
/
```

```
int addCircuit( const char* ep1, const char* ep2,int vpi,int vci,
                const char* name)
{
    // Argument list used to create the circuit
    CvClient::Args setArgs;
    // Argument list that will store values after a get request

    CvClient::Args getArgs;
    CvClient::ObjectId endpoint1;  // declare object endpoint1 of
                                   // the circuit
    CvClient::ObjectId endpoint2;  // declare object endpoint2 of
                                   // the circuit

    // initiate apiStatus to no error
    CvClient::ApiStatus apiStatus = cveNoError;
    CBsetHandler CBset;

    endpoint1 = CvParseObjectId(ep1);
    if(endpoint1.type() == CVT_Unknown)
    {
        cout << ep1 << " is not a valid circuit endpoint object" <<
endl;
        return 0;
    }


    endpoint1.setAtmInfo(vpi,vci);
    endpoint1.setType(CVT_Circuit);

    endpoint2 = CvParseObjectId(ep2);
    if(endpoint2.type() == CVT_Unknown)
    {
        cout << ep2 << " is not a valid circuit endpoint object" <<
endl;
        return 0;
    }

    endpoint2.setAtmInfo(vpi+1,vci+100);
    endpoint2.setType(CVT_Circuit);

    // creating the list of attribute/value pairs used to add the
    // circuit
```

```
    setArgs[CVA_CircuitName] = name;
    setArgs[CVA_CircuitFwdQOSClass] = 1;
    setArgs[CVA_CircuitRevQOSClass] = 1;
    setArgs[CVA_CircuitFwdTrafficDescType] = 1;
    setArgs[CVA_CircuitRevTrafficDescType] = 1;
    setArgs[CVA_CircuitFwdParam1] = 120;
    setArgs[CVA_CircuitRevParam1] = 120;
    setArgs[CVA_CircuitFwdParam2] = 120;
    setArgs[CVA_CircuitRevParam2] = 120;
    setArgs[CVA_CircuitAdminStatus] = circuitAdminUp;

    // adding the object endpoint2 to the argument list
    setArgs[CVA_CircuitEndpoint2] = endpoint2;

    // add the circuit using endpoint1

    //setting the string in CBset to Circuit
    CBset.Msg("cvadd");

    // adding object using the async call

    Client.addObject(endpoint1,&CBset,setArgs);
    // block until message arrives
    Client.processEvents(-1);
}

////////////////////////////////////////////////////////////////////
/
//
// FUNCTION
//
//     bindServiceName() - bind a ServiceName to a primary lport
//
// SYNOPSIS
//
//    bindServiceName( const char* lportId, const char* name)
//
//    "lportId" - is the string representation of the lport in its
//                hierarchical form.
//
//    "name" - is the ServiceName to which the lport will be bound.
//
```

```
// DESCRIPTION
//
//    This function binds a ServiceName to a primary lport.
//    CvParseObjectId() is used to convert from the hierarchical
//    representation to an object representation.
//
//    Necessary arguments for the binding are also set. The actual
//    binding is added using the addObject() API operation.
//
//    This function uses the synchronous interface to talk to the
//    server.
//
// RETURNS
//
//    Integer.
//
// ERRORS
//
//    Prints a message to the stdout indicating success or failure.
//    In the  latter case, a message string identifying the error is
//    also printed.
//
// CAVEATS
//
//    None.
//
////////////////////////////////////////////////////////////////////////
/
int bindServiceName( const char* lportId, const char* name)
{
    CvClient::ObjectId lport,serviceName;
    // Argument list used to create the circuit
    CvClient::Args setArgs;
    // initiate apiStatus to no error
    CvClient::ApiStatus apiStatus = cveNoError;

    char *tmp = NULL;

    lport = CvParseObjectId(lportId);
    if(lport.type() == CVT_Unknown)
    {
        cout << lportId << " is not a valid lport object" << endl;
```

```
        return 0;
    }

    ostrstream tmpStream;
    tmpStream << “network.193.1.0.0.servicename.” << name << ends;
    tmp = tmpStream.str();

    serviceName = CvParseObjectId(tmp);
    if(serviceName.type() ==  CVT_Unknown)
    {
        cout << tmp << “ is not a valid service name object” <<
endl;
        return 0;
    }

    setArgs[CVA_ServiceNamePrimaryLPort] = lport;

    delete ( tmp );

    Client.addObject(serviceName,setArgs,&apiStatus);
    if(apiStatus != cveNoError)
    {
        cout << endl << “cvadd “ << serviceName << “ “ << setArgs
             << “ *** “ << Client.getErrorMsg() << endl;
    }
    else
    {
        cout << endl << “cvadd “ << serviceName << “ was added” <<
endl;
    }
}

////////////////////////////////////////////////////////////////////
/
//
// FUNCTION
//
//    addSvcAddr() - function to add an SVC address object
//
// SYNOPSIS
//
//    addSvcAddr( const char* svcAddrId, CvObjectType type)
```

```
//
//    “svcAddrId” - is the string representation of the SVC Address
//     object being added
//
//    “type” - is the type of the SVC address being added. Various
//     types ofSVC address objects are listed in the file
//     CvObjectType.H
//
// DESCRIPTION
//
//    This function adds an SVC address object. Some types of SVC
//    address objects are Prefix, Address and the UserPart. The
//    SvcAddress itself is specified to this function in a
//    hierchical form.
//
//    Since the type of the SVC address is not represented in the
//    string form, the type is specified as the second argument.
//    CvParseObjectId() is used to convert from the string from to
//    the object representation.
//
//    This function sends the request to the server synchronously.
//
// RETURNS
//
//    Integer.
//
// ERRORS
//
//    Prints a message to the stdout indicating success or failure.
//    In the latter case, a message string identifying the error is
//    also printed.
//
// CAVEATS
//
//    None.
//
// SEE ALSO
//
//    CvObjectType.H
//
////////////////////////////////////////////////////////////////////
/
```

```
int addSvcAddr( const char* svcAddrId, CvObjectType type)
{
    CvClient::ObjectId svcAddr;
    CvClient::Args setArgs; // Argument list used to create the
                            // circuit
    // initiate apiStatus to no error
    CvClient::ApiStatus apiStatus = cveNoError;

    svcAddr = CvParseObjectId(svcAddrId);
    if(svcAddr.type() == CVT_Unknown)
    {
        cout << svcAddrId << " is not a valid svc address object"
<< endl;
        return 0;
    }

    svcAddr.setType(type);

    Client.addObject(svcAddr, setArgs, &apiStatus);
    if(apiStatus != cveNoError)
    {
        cout << endl << "cvadd " << svcAddr << " " << setArgs
             << " *** " << Client.getErrorMsg() << endl;
    }
    else
    {
        cout << endl << "cvadd " << svcAddr << " was added" << endl;
    }
}

////////////////////////////////////////////////////////////////////
/
//
// FUNCTION
//
//    addSecScreen() - function to add a SecurityScreen to an LPort.
//
// SYNOPSIS
//
//    addSecScreen(const char* screenId, const char* lportId)
//
```

```
//    "screenId" - string representation of the SecurityScreen
//     object.
//
//    "lportId" - string  representation of the lport object.
//
// DESCRIPTION
//
//   This function adds a SecurityScreen object (child) to an LPort,
//   synchronously. CvParseObjectId() is used to convert the
//   hierarchical representation of the objects into the object
//   form.
//
//   Since the SecurityScreen is a NetworkWide object, adding the
//   SecurityScreen and assigning it to an LPort are performed as
//   two independent operations.
//
//   The appropriate arguments are set for both the operations.
//
// RETURNS
//
//    Integer.
//
// ERRORS
//
//   Prints a message to the stdout indicating success or failure.
//   In the latter case, a message string identifying the error is
//   also printed.
//
// CAVEATS
//
//    None.
//
////////////////////////////////////////////////////////////////////
/

int addSecScreen(const char* screenId, const char* lportId)
{
    CvClient::ObjectId lport,screen;
    CvClient::Args setArgs,args; // Argument list used to create the
                                 // screen
    // initiate apiStatus to no error
    CvClient::ApiStatus apiStatus = cveNoError;
```

```
lport = CvParseObjectId(lportId);
if(lport.type() == CVT_Unknown)
{
    cout << lportId << " is not a valid lport object" << endl;
    return 0;
}

screen = CvParseObjectId(screenId);
if(screen.type() == CVT_Unknown)
{
    cout << screenId << " is not a valid security screen
                        object" << endl;
    return 0;
}

setArgs[CVA_SvcSecScnCallingAddressType] = scnAddrTypeAesa;
setArgs[CVA_SvcSecScnCalledAddressType] = scnAddrTypeE164;
setArgs[CVA_SvcSecScnCalledSubAddressType] = scnAddrTypeIgnored;
setArgs[CVA_SvcSecScnCallingSubAddressType] =
scnAddrTypeIgnored;
setArgs[CVA_SvcSecScnCallDirection] = scnCallDirIngress;
setArgs[CVA_SvcSecScnType] = scnTypeBlock;
setArgs[CVA_SvcSecScnCallingAddress] =
                  "ABCDEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF";
setArgs[CVA_SvcSecScnCalledAddress] = "111111111111111";


Client.addObject(screen,setArgs,&apiStatus);
if(apiStatus != cveNoError)
{
    cout << endl << "cvadd " << screen << " " << setArgs
         << " *** " << Client.getErrorMsg() << endl;
}
else
{
    cout << endl << "cvadd " << screen << " was added" << endl;
}

args[CVA_AssignedSvcSecScnSecurityAdminStatus] =
secScnAdminActive;
```

```
    Client.addObject(lport,args,&apiStatus);
    if(apiStatus != cveNoError)
    {
        cout << endl << "cvadd " << lport << " " << args
             << " *** " << Client.getErrorMsg() << endl;
    }
    else
    {
        cout << endl << "cvadd " << lport << " was added" << endl;
    }
}

////////////////////////////////////////////////////////////////////
/
//
// FUNCTION
//
//    modifySvcConfig() - a function to modify the SvcConfig.
//
// SYNOPSIS
//
//    modifySvcConfig(const char* lportId)
//
//    "lportId" - is the string representation of the SvcConfig
//     sub-object under the lport.
//
// DESCRIPTION
//
//    This function performs a modify operation on the SvcConfig
//    subobject under the lport object. This operation is performed
//    synchronously.
//
//    Once again the CvParseObjectId() is used to convert from the
//    string form to the object form.
//
//    The appropriate ArgIds are set for the configuration purpose.
//
// RETURNS
//
//    Integer.
//
// ERRORS
```

```
//
//    Prints a message to the stdout indicating success or failure.
//    In the latter case, a message string identifying the error is
//    also printed.
//
// CAVEATS
//
//    None.
//
////////////////////////////////////////////////////////////////////
/

int modifySvcConfig(const char* lportId)
{
    CvClient::ObjectId lport;
    CvClient::Args setArgs; // Argument list used to create the
                            // circuit
    // initiate apiStatus to no error
    CvClient::ApiStatus apiStatus = cveNoError;

    lport = CvParseObjectId(lportId);
    if(lport.type() == CVT_Unknown)
    {
        cout << lportId << " is not a valid lport object" << endl;
        return 0;
    }

    setArgs[CVA_SvcConfigInsertMode] = cfgCgPtyInsertModeDisabled;
    setArgs[CVA_SvcConfigEgAddrXlateMode] =
    cfgEgAddrXlateModeDisabled;
    setArgs[CVA_SvcConfigIgAddrXlateMode] =
    cfgIgAddrXlateModeDisabled;
    setArgs[CVA_SvcConfigPresentMode] = cfgCgPtyPresentModeUser;
    setArgs[CVA_SvcConfigSvcHDTimer] = 60;
    setArgs[CVA_SvcConfigSvcCdvTolerance] = 650;
    setArgs[CVA_SvcConfigSvcLoadBalance] = 3650;
    setArgs[CVA_SvcConfigSvcFailureTrapThresh] = 1;
    setArgs[CVA_SvcConfigSvcCugState] = cfgCugStateDisabled;
    setArgs[CVA_SvcConfigSvcFrameDiscard] = frameDiscardDisabled;

    Client.modifyObject(lport,setArgs,&apiStatus);
    if(apiStatus != cveNoError)
```

```
    {
        cout << endl << "cvmodify " << lport << " " << setArgs
             << " *** " << Client.getErrorMsg() << endl;
    }
    else
    {
        cout << endl << "cvmodify " << lport << " was was modified"
<< endl;
    }
}

////////////////////////////////////////////////////////////////////
/
//
// FUNCTION
//
//    addSPVC() - a function to add an SPVC
//
// SYNOPSIS
//
//    addSPVC(const char* endpointId, const char* address, const
//    char* name)
//
//    "endpointId" - is the string representation of the originating
//     endpoint.
//
//    "address" - identifies the SVC address of the terminating
//     endpoint.
//
//    "name" - is the name of the SPVC circuit.
//
// DESCRIPTION
//
//
//    This function adds a SPVC object synchronously. Both the
//    orginating and the terminating endpoints are specified as
//    strings, one of the them being an SVC address.
//
//    CvParseObjectId() is used to convert the string representation
//    of the originating endpoint to the object form. The
//    terminating endpoint is specified as a string and is set along
//    with other configuration ArgIds.
```

```
//
// RETURNS
//
//    Integer.
//
// ERRORS
//
//    Prints a message to the stdout indicating success or failure.
//    In the latter case, a message string identifying the error is
//    also printed.
//
// CAVEATS
//
//    None.
//
////////////////////////////////////////////////////////////////////
/

int addSPVC(const char* endpointId, const char* address, const
char* name)
{
    CvClient::ObjectId endpt;
    CvClient::Args setArgs; // Argument list used to create the
                            // circuit
    // initiate apiStatus to no error
    CvClient::ApiStatus apiStatus = cveNoError;

    endpt = CvParseObjectId(endpointId);
    if(endpt.type() == CVT_Unknown)
    {
        cout << endpointId << " is not a valid SPVC object" << endl;
        return 0;
    }

    setArgs[CVA_SpvcTerminatingEndpointAddr] = address;
    setArgs[CVA_SpvcName] = name;
    setArgs[CVA_SpvcAdminStatus] = circuitAdminUp;
    setArgs[CVA_SpvcTargetSelectType] = cvTargetSelectTypeAny;
    setArgs[CVA_SpvcRetryInterval] = (unsigned long)0;
    setArgs[CVA_SpvcRetryLimit] = (unsigned long)0;
    setArgs[CVA_SpvcFwdTrafficDesc] = "PMP Rev CBR";
    setArgs[CVA_SpvcRevTrafficDesc] = "PMP Rev CBR";
```

```
    Client.addObject(endpt,setArgs,&apiStatus);
    if(apiStatus != cveNoError)
    {
        cout << endl << "cvadd " << endpt << " " << setArgs
             << " *** " << Client.getErrorMsg() << endl;
    }
    else
    {
        cout << endl << "cvadd " << endpt << " was added" << endl;
    }
}

////////////////////////////////////////////////////////////////////
/
//
// FUNCTION
//
//    addPmpSPVC(const char* endpointId, const char* address,
//               const char* name) - a function to add a PMP SPVC
//               root.
//
// SYNOPSIS
//
//    addPmpSPVC(const char* endpointId, const char* address,
//               const char* name)
//
//    "endpointId" - is the string representation of the PMP SPVC
//     Root
//
//    "address" - is the string representation of the terminating
//     SVC endpoint.
//
//    "name" - is the name of the PMP Spvc root object.
//
// DESCRIPTION
//
//    This function adds a PMP SPVC root object synchronously.
//    Both the orginating and the terminating endpoints are
//    specified as strings, one of the them being an SVC address.
//
//    CvParseObjectId() is used to convert the string representation
```

```
//    of the originating endpoint to the object form. The
//    terminating endpoint is specified as a string and is set along
//    with other configuration ArgIds.
//
// RETURNS
//
//    Integer.
//
// ERRORS
//
//    Prints a message to the stdout indicating success or failure.
//    In the latter case, a message string identifying the error is
//    also printed.
//
// CAVEATS
//
//    None.
//
////////////////////////////////////////////////////////////////////////
/

int addPmpSPVC(const char* endpointId, const char* address,
               const char* name)
{
    CvClient::ObjectId endpt;
    CvClient::Args setArgs; // Argument list used to create the
                            // circuit
    // initiate apiStatus to no error
    CvClient::ApiStatus apiStatus = cveNoError;

    endpt = CvParseObjectId(endpointId);
    if(endpt.type() == CVT_Unknown)
    {
        cout << endpointId << " is not a valid PMP SPVC object" <<
endl;
        return 0;
    }

    setArgs[CVA_PMPSpvcRootTerminatingEndpointAddr] = address;
    setArgs[CVA_PMPSpvcRootName] = name;
    setArgs[CVA_PMPSpvcRootAdminStatus] = circuitAdminUp;
    setArgs[CVA_PMPSpvcRootTargetSelectType] =
```

```
cvTargetSelectTypeAny;
    setArgs[CVA_PMPSpvcRootRetryInterval] = (unsigned long)0;
    setArgs[CVA_PMPSpvcRootRetryLimit] = (unsigned long)0;
    setArgs[CVA_PMPSpvcRootFwdTrafficDesc] = “PMP Rev CBR”;

    Client.addObject(endpt,setArgs,&apiStatus);
    if(apiStatus != cveNoError)
    {
       cout << endl << “cvadd “ << endpt << “ “ << setArgs << “ ***
“
            << Client.getErrorMsg() << endl;
    }
    else
    {
        cout << endl << “cvadd “ << endpt << “ was added” << endl;
    }
}

////////////////////////////////////////////////////////////////////
/
//
// FUNCTION
//
//    addPmpRoot() - a function to add a regular PMP Ckt Root.
//
// SYNOPSIS
//
//    addPmpRoot( const char* endpointId, const char* name)
//
//    “endpointId” - is the string representation of the regular PMP
//                   Ckt root.
//
//    “name” - is the name of the regular PMP Ckt root object.
//
// DESCRIPTION
//
//    This function adds a PMP circuit Root object synchronously.
//    The input specified is the hierarchical representation of the
//    root. CvParseObjectId() is used to convert from the string
//    form to the object form.
//
//    Root name is specified is specified among several configurable
```

```
//    ArgIds.
//
// RETURNS
//
//    Integer.
//
// ERRORS
//
//    Prints a message to the stdout indicating success or failure.
//    In the latter case, a message string identifying the error is
//    also printed.
//
// CAVEATS
//
//    None.
//
////////////////////////////////////////////////////////////////////
/

int addPmpRoot( const char* endpointId, const char* name)
{
    CvClient::ObjectId endpt;
    CvClient::Args setArgs,getArgs; // Argument list used to create
                                    // the circuit
    // initiate apiStatus to no error
    CvClient::ApiStatus apiStatus = cveNoError;

    endpt = CvParseObjectId(endpointId);
    if(endpt.type() == CVT_Unknown)
    {
        cout << endpointId << " is not a valid Pmp Root object" <<
endl;
        return 0;
    }

    setArgs[CVA_PMPCktRootName] = name;
    setArgs[CVA_PMPCktRootTrafficDescType] =
    cvAtmTfD_PcrClp0PcrClp01;
    setArgs[CVA_PMPCktRootNdcEnable] = cktNdcEnableOff;
    setArgs[CVA_PMPCktRootQOSClass] = cvCOSCBR;
    setArgs[CVA_PMPCktRootparam1] = 120;
    setArgs[CVA_PMPCktRootparam2] = 120;
```

```
    setArgs[CVA_PMPCktRootCdvTolerance] = 600;
    setArgs[CVA_PMPCktRootPrivateNetOverflow] =
    privateNetOverflowPublic;
    setArgs[CVA_PMPCktRootAcctChrgPartyId] = “11”;
    setArgs[CVA_PMPCktRootAcctUsageMeasure] =
    cvCktAtmBillUsageEnabled;
    setArgs[CVA_PMPCktRootAcctPvcControl] =
    cvCktAtmBillControlEnabled;

    Client.addObject(endpt,setArgs,&apiStatus);
    getArgs = Client.getResponseArgs();
    if(apiStatus != cveNoError)
    {
       cout << endl << “cvadd “ << endpt << “ “ << getArgs << “ ***
“
            << Client.getErrorMsg() << endl;
    }
    else
    {
        cout << endl << “cvadd “ << endpt << “ was added” << endl;
    }
}

////////////////////////////////////////////////////////////////////
/
//
// FUNCTION
//
//    addPmpLeaf() - a function to add a regular PMP Ckt Leaf.
//
// SYNOPSIS
//
//    addPmpLeaf(const char* rootId, const char* leafId)
//
//    “rootId” - is the string representation of the PMP ckt root
//
//    “leafId” - is the string representation of the PMP ckt leaf.
//
// DESCRIPTION
//
//    This function adds a PMP ckt leaf to a PMP ckt root, given the
//    string representations of both the root and the leaf. This
```

```
//    operation is performed synchronously.
//
//    CvParseObjectId() is used to convert the string
//    representations into the object representations.
//
//    The Root object is specified as one of the ArgIds while adding
//    the leaf.
//
// RETURNS
//
//    Integer.
//
// ERRORS
//
//    Prints a message to the stdout indicating success or failure.
//    In the latter case, a message string identifying the error is
//    also printed.
//
// CAVEATS
//
//    None.
//
////////////////////////////////////////////////////////////////////////
/
int addPmpLeaf(const char* rootId, const char* leafId)
{
    CvClient::ObjectId root,leaf;
    CvClient::Args setArgs; // Argument list used to create the
                            // circuit
    // initiate apiStatus to no error
    CvClient::ApiStatus apiStatus = cveNoError;

    root = CvParseObjectId(rootId);
    if(root.type() == CVT_Unknown)
    {
        cout << rootId << " is not a valid Pmp Root object" << endl;
        return 0;
    }

    leaf = CvParseObjectId(leafId);
    if(leaf.type() == CVT_Unknown)
    {
```

```
        cout << leafId << " is not a valid Pmp Leaf object" << endl;
        return 0;
    }


    setArgs[CVA_PMPCktRoot] = root;
    setArgs[CVA_PMPCktAdminStatus] = cktLeafAdmUp;

    Client.addObject(leaf,setArgs,&apiStatus);
    if(apiStatus != cveNoError)
    {
        cout << endl << "cvadd " << leaf << " " << setArgs << " ***
"
             << Client.getErrorMsg() << endl;
    }
    else
    {
        cout << endl << "cvadd " << leaf << " was added" << endl;
    }
}

////////////////////////////////////////////////////////////////////
/
//
// FUNCTION
//
//    rmObject() - a function to delete an object.
//
// SYNOPSIS
//
//    rmObject( const char* object)
//
//    "object" - identifies the string representation of the object.
//
// DESCRIPTION
//
//    This function deletes the specified object syncshronously.
//    CvParseObjectId() used to convert the string representation
//    into the object form.
//
// RETURNS
//
```

```
//    Integer.
//
// ERRORS
//
//    Prints a message to the stdout indicating success or failure.
//    In the latter case, a message string identifying the error is
//    also printed.
//
// CAVEATS
//
//    None.
//
////////////////////////////////////////////////////////////////////
/

int rmObject( const char* object)
{
    CvClient::ObjectId obj;
    // initiate apiStatus to no error
    CvClient::ApiStatus apiStatus = cveNoError;

    obj = CvParseObjectId(object);
    if(obj.type() == CVT_Unknown)
    {
        cout << object << " is not a valid object" << endl;
        return 0;
    }

    Client.deleteObject(obj,&apiStatus);
    if(apiStatus != cveNoError)
    {
        cout << endl << "cvdelete " << obj << " *** "
             << Client.getErrorMsg() << endl;
    }
    else
    {
        cout << endl << "cvdelete " << obj << " was deleted" <<
endl;
    }
}

////////////////////////////////////////////////////////////////////
```

```
/
//
// FUNCTION
//
//    rmObjects() - a wrapper function which uses rmObject()
//
// SYNOPSIS
//
//    rmObjects( void)
//
// DESCRIPTION
//
//    A wrapper function which invokes rmObject() defined above to
//    delete an object.
//
// RETURNS
//
//    None.
//
// ERRORS
//
//    None.
//
// CAVEATS
//
//    None.
//
////////////////////////////////////////////////////////////////////
/

void rmObjects( void )
{

rmObject("switch.193.1.141.2.card.7.pport.1.startVpi.2.vpi.2.vci.100
");

rmObject("switch.193.1.141.2.card.7.pport.1.startVpi.2.vpi.2.vci.101
");

rmObject("switch.193.1.141.2.card.7.pport.1.startVpi.2.vpi.2.vci.102
");
```

```
rmObject("switch.193.1.141.2.card.7.pport.1.lport.1.vpi.1.vci.100");

rmObject("switch.193.1.141.2.card.12.pport.1.lport.1.vpi.1.vci.101")
;

rmObject("switch.193.1.141.2.card.12.pport.1.lport.1.vpi.1.vci.102")
;


rmObject("switch.193.1.141.2.card.12.pport.1.lport.1.vpi.1.vci.300")
;

rmObject("switch.193.1.141.2.card.12.pport.1.lport.1.vpi.1.vci.301")
;

rmObject("switch.193.1.141.2.card.12.pport.1.lport.1.vpi.1.vci.302")
;

    rmObject("network.193.1.0.0.ServiceName.vAtmUniDte");


rmObject("switch.193.1.141.2.card.12.pport.1.startVpi.2.svcprefix.1-
111111111111111-120");

rmObject("switch.193.1.141.2.card.12.pport.1.startVpi.2.svcaddress.1
-111111111111111-120");

rmObject("Switch.193.1.141.2.card.7.pport.1.startVpi.2.svcuserpart.7
-aaaaaaaaaaaaaa-56");


rmObject("Switch.193.1.141.2.card.7.pport.1.startVpi.2.assignedSvcSe
cScn.securityScreen");
    rmObject("Network.193.1.0.0.SvcSecScn.securityScreen");

    rmObject(

"Switch.193.1.141.2.card.12.pport.1.startVpi.2.Spvc.vpi.5.vci.55");
    rmObject(

"Switch.193.1.141.2.card.12.pport.1.startVpi.2.Spvc.vpi.5.vci.56");
    rmObject(
```

```
"Switch.193.1.141.2.card.12.pport.1.startVpi.2.Spvc.vpi.5.vci.57");

    rmObject(

"Switch.193.1.141.2.card.7.pport.1.startVpi.2.PMPSpvcRoot.vpi.6.vci.
66");

    rmObject(

"Switch.193.1.141.2.card.7.pport.1.startVpi.2.PMPCktLeaf.vpi.8.vci.8
8");
    rmObject(

"Switch.193.1.141.2.card.7.pport.1.lport.1.PMPCktLeaf.vpi.1.vci.99")
;
    rmObject(

"Switch.193.1.141.2.card.12.pport.1.startVpi.2.PMPCktRoot.vpi.7.vci.
77");

    rmObject("switch.193.1.141.2.card.7.pport.1.startVpi.2");
    rmObject("switch.193.1.141.2.card.12.pport.1.startVpi.2");

    rmObject("switch.193.1.141.2.card.7.pport.1.lport.1");
    rmObject("switch.193.1.141.2.card.12.pport.1.lport.1");
}

// Main Program invokes the functions defined above.

int main()
{
    char
lport1[MAXLINE],lport2[MAXLINE],lport3[MAXLINE],lport4[MAXLINE];

    strcpy(lport1,"switch.193.1.141.2.card.7.pport.1.lport.1");
    strcpy(lport2,"switch.193.1.141.2.card.7.pport.1.startVpi.2");
    strcpy(lport3,"switch.193.1.141.2.card.12.pport.1.lport.1");
    strcpy(lport4,"switch.193.1.141.2.card.12.pport.1.startVpi.2");

    // open a session between client and server
    Client.open("152.148.10.189",4001,"","");
```

```
    // set the modify type to configure the database only.
    Client.setModifyType(modifyDbOnly);

    // Remove any objects that have been created previously. Users
    // don't have to do this, it is only being called first for the
    // purpose of demonstrating the deletes.
    rmObjects();

    // adding lports to create a circuit on
    addLPort(lport1,"atmUniDce711");
    addLPort(lport2,"vAtmUniDte711");
    addLPort(lport3,"atmUniDte1211");
    addLPort(lport4,"vAtmUniDce1211");


    // List the lports created so far.
    seeLPorts ( "switch.193.1.141.2" );

    // Now adding the circuits on the created lports. These ckt
    // endpoints are non-ServiceName based endpoints, i.e regular
    // endpoints.

    addCircuit(lport2,lport4,2,100,"virtual711-1211-1");
    addCircuit(lport2,lport4,2,101,"virtual711-1211-2");
    addCircuit(lport2,lport4,2,102,"virtual711-1211-3");
    addCircuit(lport1,lport4,1,100,"virtual711-1211-4");
    addCircuit(lport3,lport2,1,101,"virtual711-1211-5");
    addCircuit(lport3,lport2,1,102,"virtual711-1211-6");

    // bind a service name to one of the lports that was created
    // above.
    bindServiceName(lport2,"vAtmUniDte");

    // Adding the circuits with one endpoint being a ServiceName.

    addCircuit(lport3,

"Network.193.1.0.0.ServiceName.vAtmUniDte",1,300,"snCkt1");
    addCircuit(lport3,

"Network.193.1.0.0.ServiceName.vAtmUniDte",1,301,"snCkt2");
    addCircuit(lport3,
```

```
"Network.193.1.0.0.ServiceName.vAtmUniDte",1,302,"snCkt3");


addSvcAddr("switch.193.1.141.2.card.12.pport.1.startVpi.2.svcprefix.
1-111111111111111-120",
                CVT_SvcPrefix);

addSvcAddr("switch.193.1.141.2.card.12.pport.1.startVpi.2.svcaddress
.1-111111111111111-120",
                CVT_SvcAddress);

addSvcAddr("Switch.193.1.141.2.card.7.pport.1.startVpi.2.svcuserpart
.7-aaaaaaaaaaaaaa-56",
                CVT_SvcUserPart);

    addSecScreen(" Network.193.1.0.0.SvcSecScn.securityScreen",

"Switch.193.1.141.2.card.7.pport.1.startVpi.2.assignedSvcSecScn.secu
rityScreen");


modifySvcConfig("Switch.193.1.141.2.card.7.pport.1.startVpi.2.svccon
fig");


addSPVC("Switch.193.1.141.2.card.12.pport.1.startVpi.2.Spvc.vpi.5.vc
i.55",
            "1-111111111111111-120",
            "spvc1211-1");

addSPVC("Switch.193.1.141.2.card.12.pport.1.startVpi.2.Spvc.vpi.5.vc
i.56",
            "1-111111111111112-120",
            "spvc1211-2");

addSPVC("Switch.193.1.141.2.card.12.pport.1.startVpi.2.Spvc.vpi.5.vc
i.57",
            "1-111111111111113-120",
            "spvc1211-3");

    addPmpSPVC(
```

```
“Switch.193.1.141.2.card.7.pport.1.startVpi.2.PMPSpvcRoot.vpi.6.vci.
66”,
     “1-555555555555555-120”,
     “PmpSpvc711-1”);

   addPmpRoot(

“Switch.193.1.141.2.card.12.pport.1.startVpi.2.PMPCktRoot.vpi.7.vci.
77”,
     “PmpRoot1211”);

   addPmpLeaf(

“Switch.193.1.141.2.card.12.pport.1.startVpi.2.PMPCktRoot.vpi.7.vci.
77”,

“Switch.193.1.141.2.card.7.pport.1.startVpi.2.PMPCktLeaf.vpi.8.vci.8
8”);

   addPmpLeaf(

“Switch.193.1.141.2.card.12.pport.1.startVpi.2.PMPCktRoot.vpi.7.vci.
77”,

“Switch.193.1.141.2.card.7.pport.1.lport.1.PMPCktLeaf.vpi.1.vci.99”)
;
}
```

B

Error Codes

This appendix lists the error codes that can be returned by the NavisXtend Provisioning Server API functions or the CLI commands. The error codes are listed in numerical order, followed by the message name. When writing source code, you should test for the message name rather than the error code number.

Success Condition

0 **cveNoError** — The function returned without error.

Error Codes

1 **cveAccessDenied** — Access denied.

2 **cveLockRequired** — A lock is required to perform this action.

3 **cveNoMemory** — Out of memory.

4 **cveInvalidSession** — Invalid session used.

5 **cveInvalidRequest** — Invalid request made.

6 **cveModReadOnly** — An attempt was made to modify a read-only field.

7 **cveDatabaseInvalid** — A database inconsistency has been found.

8 **cveGetUName** — Failed to get user name by ‘getpwent()’.

9 **cveUsersNotFound** — The built-in users have not been installed in the database.

10 **cveConflictingNetwideParams** — A conflicting network-wide parameters object was found.

11 **cveInvalidAddrSignif** — The address significance must be either ‘local’ or ‘global’.

12 **cveInvalidNetNumber** — The network number must represent a valid Class B network.

13 **cveNoMapApplInfo** — The CascadeView-specific information for this map could not be found.

14 **cveDbObjNotFound** — The requested database object could not be found.

15 **cveContactTooLong** — The contact name is too long.

16 **cvePhoneNoTooLong** — The phone number is too long, Max = 32 digits.

17 **cveLocationTooLong** — The location value is too long, Max = 160 chars.

18 **cveMaxConsolesTooBig** — The max. telnet consoles value is too large.

19 **cveConsoleTimeoutTooBig** — The console timeout value is too large.

20 **cveConflictingSwitches** — A conflicting switch object was found.

21 **cveInvalidIpAddr** — The IP address is invalid.

22 **cveNetwideInfoNotFound** — The network object was not found.

23 **cveMaxSwitchesExceeded** — The maximum number of switches (400) has been exceeded.

24 **cveNoSuchObject** — The object does not exist.

25 **cveLPortsExist** — The object has logical ports defined for it.

26 **cveSwitchNotManaged** — This switch will not be managed by CascadeView.

27 **cveInvalidSwitchName** — Invalid switch name.

28 **cveInvalidSwitchType** — Invalid switch type.

29 **cveOnlyOvDataFound** — The object data has been set up by OpenView but not by CascadeView.

30 **cveInvalidCommName** — Invalid community name.

31 **cveInvalidAccess** — Invalid IP Address.

32 **cveCommunityExists** — An object with the specified community name already exists.

33 **cveNmsEntryExists** — An object with the specified NMS address already exists.

35 **cveMaxNmsAccessExceeded** — The maximum number of NMS entries has been exceeded.

34 **cveSwitchNotFound** — The switch object was not found.

36 **cveInvalidGatewayAddr** — Invalid gateway address.

37 **cveNmsPathExists** — A path to the specified NMS address already exists.

38 **cveInvalidMgmtDlciName** — Invalid management DLCI name.

39 **cveNoSuchMgmtDlci** — The specified management connection does not exist.

40 **cveInvalidNmsPathType** — Invalid NMS path type */

41 **cveInvalidAdminStatus** — Invalid admin status value.

42 **cveSnmpTimedOut** — The SNMP request to the agent timed out.

43 **cvePduReturnedNull** — The PDU returned with the response is NULL.

44 **cveRequestNotFound** — The response PDU could not be matched with an original request.

45 **cvePduError** — Error processing request: PDU contains more information.

46 **cveNmsNotFound** — No NMS objects were found in the database.

47 **cveTestTimedOut** — The operation did not complete within the configured time period.

48 **cveSnmpSwitchNotManaged** — The switch is not currently managed, so no SNMP request was sent.

49 **cveNoMapOpen** — No map is currently open */

50 **cveOutOfSync** — + Out of Sync */

51 **cveTypeMismatch** — + Type Mismatch.

52 **cveStatusUp** — Status Up.

53 **cveStatusMarginal** — Status Marginal.

54 **cveStatusDown** — Status Down.

55 **cveStatusUnmanaged** — Status Unmanaged.

56 **cveStatusUnreachable** — Status Unreachable.

57 **cveStatusInvalid** — Status Invalid.

58 **cveStatusTesting** — Status Testing.

59 **cveStatusInitializing** — Status Initializing.

60 **cveStatusRedundBoot** — Status Redundant Boot.

61 **cveStatusUnknown** — Status Unknown.

62 **cveCardNotDefined** — Not Defined.

63 **cveCardNotPresent** — Not Present.

64 **cveConfigIsUnchanged** — The old and new configurations are identical.

65 **cveFileOpenFailure** — The file could not be opened.

66 **cveCardObjectNotFound** — The card object could not be found in the database.

67 **cveCardInUseAsRedund** — The slot specified is already defined as the redundant slot of another card.

68 **cveStdxNoCardAdminStatus** — Card admin. status is not supported for STDX 3000- and 6000-type switches.

69 **cveCardMaintenNoOtherChg** — If you are setting the card's admin. status to 'Maintenance', no other change can be made.

70 **cveCardMaintenNoChg** — No change can be made while the card's admin. status is 'Maintenance'.

71 **cveInvalidCardForSlot** — The specified card type is invalid for this slot.

72 **cveCardLPortsExist** — Logical ports are defined on the card.

73 **cveStdxNoCardRedundancy** — The STDX 3000 and 6000 do not support redundant cards.

74 **cveNoEmptyRedundCard** — An empty slot cannot be defined as redundant.

75 **cveRedundCardMustBeHigher** — The redundant card must be in the next higher slot from the primary card.

76 **cveRedundCardInUse** — The slot specified as redundant is already in use.

77 **cveCardExtlClockNotValid** — 'External clock' is not a valid parameter for the specified card type.

78 **cveModStandbyCard** — The standby card's configuration cannot be modified directly.

79 **cveUioPortUnusable** — This port cannot be used because a neighboring port has exceeded bandwidth limitations.

80 **cvePPortInvalidClockSrc** — Invalid clock source setting for this port type.

81 **cvePPortClockSpeedTooHigh** — Clock speed setting too high for this port type.

82 **cvePPortInvalidLineType** — Invalid line framing for this port type.

83 **cvePPortInvalidZeroCoding** — Invalid zero coding for this port type.

84 **cvePPortInvalidConnType** — Invalid connection type setting for this port type.

85 **cvePPortInvalidLineLength** — Invalid line length setting for this port and connection type.

86 **cvePPortInvalidLineBuildOut** — Invalid line build-out setting for this port and connection type.

87 **cvePPortInvalidChannels** — Some of the allocated channels are not valid for this port type.

88 **cvePPortObjectNotFound** — The card object could not be found in the database.

89 **cvePPortTypeChange** — The port and/or interface type cannot be changed on this port.

90 **cvePPortBandwidthTooLow** — The specified bandwidth is too low to support all the logical ports defined on this physical port.

91 **cvePPortChannelsDeallocated** — Some of the channels deallocated are in use by logical ports.

92 **cvePPortUioBandwidthTooHigh** — The specified bandwidth is too high because neighboring ports are already in use.

93 **cvePPortBitStuffNotInUse** — The zero coding cannot be set to 'jammed bit' because not all logical ports have bit stuffing enabled.

94 **cveStatusMaintenance** — Status Maintenance.

95 **cveLPortTypeInvalid** — Invalid logical port type.

96 **cveLPortTypeNotSupported** — Logical port type not supported.

97 **cveLPortServiceInvalid** — Invalid logical port service type.

98 **cveLPortServiceNotSupported** — Logical port service type not supported.

99 **cveDceVerifTimerInvalid** — Invalid DCE Poll Verify Timer value (5 - 30).

100 **cveDceErrThreshInvalid** — Invalid DCE Error Threshold value (1 - 10).

101 **cveDceEventCountInvalid** — Invalid DCE Event Count value (1 - 10).

102 **cveDteErrThreshInvalid** — Invalid DTE Error Threshold value (1 - 10).

103 **cveDteEventCountInvalid** — Invalid DTE Event Count value (1 - 10).

104 **cveDtePollTimerInvalid** — Invalid DTE Poll Interval value (5 - 30).

105 **cveDteFullPollInvalid** — Invalid DTE Full Status Poll Frequency value (1 - 255).

106 **cveDlcmiStdInvalid** — Invalid management standard value.

107 **cveDlciAddrFmtInvalid** — Invalid DLCI address format value.

108 **cveDlciAddrLenInvalid** — Invalid DLCI address length value.

109 **cveGlobalIdWithLocal** — Attempt to set global DLCI ID in local address mode.

110 **cveDteSetForDce** — The UNI DTE-specific fields of the port structure should not be set.

111 **cveDceSetForDte** — The UNI DCE-specific fields of the port should not be set.

112 **cveMaxFrameSizeInvalid** — Invalid value for max. frame size.

113 **cveCongThreshInvalid** — Invalid congestion threshold value.

114 **cveErrPMinThreshInvalid** — Error-per-minute threshold value is invalid.

115 **cveAdminStatusInvalid** — Invalid administrative status value.

116 **cveMaxLPortsExceeded** — Maximum number of logical ports exceeded.

117 **cveLPortNameInUse** — The specified logical port name is already in use on this switch.

118 **cveInvalidLPortId** — The specified logical port ID is not valid in this location.

119 **cveNoBigDlcis** — The current management standard setting does not allow DLCI IDs higher than 991.

120 **cveLPortIdInUse** — The specified logical port ID is already in use on this physical port.

121 **cveOptTrunkOnChan** — An OPTimum trunk cannot be defined on a channelized port.

122 **cveOptTrunkNoFeeder** — An OPTimum trunk cannot be defined unless a feeder-type logical port of the same service type has already been defined on the same physical port.

123 **cveLPortChannelsNotAvail** — Some of the DS0s requested are not available.

124 **cveLPortBandwidthTooHigh** — More bandwidth was requested than the physical port has available.

125 **cveBitStuffForUio** — Bit stuffing should not be set for this logical port type.

126 **cveBitStuffInvalid** — Invalid bit stuffing value.

127 **cveLPortChannelsForUio** — Fractional DS0s should not be set for this logical port type.

128 **cveLPortChannelsNotSet** — The fractional DS0s field is not set.

129 **cveLPortJammedBit** — The physical port's zero coding is set to 'Jammed Bit'; bit stuffing must be on.

130 **cveLPortObjectNotFound** — The logical port could not be found.

131 **cveLPortDelTrunkDefined** — The logical port cannot be deleted because it is in use as a trunk endpoint.

132 **cveLPortDelDlciDefined** — The logical port cannot be deleted because a circuit and/or management connection is defined on it.

133 **cveLPortDelFeeder** — A feeder port cannot be deleted as long as one or more OPTimum trunks of the same service type are defined on the same physical port.

134 **cveLPortTypeChange** — The service type and port type cannot be changed.

135 **cveLPortBandwidthInUse** — The specified bandwidth is too low for the circuits defined on the port.

136 **cveLPortModTrunkDefined** — The bandwidth of this port cannot be changed because it is defined as a trunk endpoint.

137 **cveTrunkNameInUse** — The specified trunk name is not unique.

138 **cveTrunkOnLPorts** — A trunk has already been configured on one or both of the specified logical ports.

139 **cveTrunkObjectNotFound** — The trunk does not exist.

140 **cveTrunkInPath** — The trunk is required for one or more defined circuit paths.

141 **cveCktInvalidPrio** — Invalid circuit priority value.

142 **cveCktCirTooHigh** — The CIR value must be less than the bandwidth configured for either logical port.

143 **cveCktCir0BcNon0** — When CIR is zero, BC must be zero.

144 **cveCktCir0BeTooHigh** — When CIR is zero, BE must be less than the bandwidth configured for either logical port.

145 **cveCktCirNon0BcInvalid** — When CIR is non-zero, BC must be non-zero.

146 **cveCktCirNon0BeInvalid** — When CIR is non-zero, BE must be a multiple of BC.

147 **cveCktCirNon0BeTooHigh** — The quantity (1 + BE/BC) * CIR must be less than the bandwidth configured for either logical port.

148 **cveCktSegSizeInvalid** — Invalid segment size.

149 **cveCktAdminStatusInvalid** — Invalid administrative status.

150 **cveCktNameInUse** — The specified circuit name is not unique.

151 **cveCktPortConflict** — The specified logical ports do not have valid types for creating a circuit.

152 **cveCktInvalidDlciId** — DLCI IDs must be between 16 and 991 (1007 if link mgmt. is LMI).

153 **cveDlciIdInUse** — The specified DLCI or VPI/VCI is not unique.

154 **cveMaxCktsExceeded** — The maximum number of circuits (including multicast and management connections) cannot exceed 938 for a card.

155 **cveCktObjectNotFound** — The circuit could not be found.

156 **cveMaxCktPathsExceeded** — The maximum number of circuit paths has been exceeded.

157 **cveCktPathInvalid** — Invalid circuit path.

158 **cveMgmtDlciNameInUse** — The specified management connection name is not unique.

159 **cveMgmtDlciBadLPort** — The specified logical port cannot have a management connection.

160 **cveMgmtDlciObjectNotFound** — The management connection could not be found.

161 **cveMgmtDlciDelInUse** — The management connection cannot be deleted because it is in use by one or more NMS paths.

162 **cveMcastMembersInvalid** — The group member list is invalid; each number must be the DLCI ID of a circuit endpoint located on the same logical port as the multicast DLCI group.

163 **cveMcastTooManyMembers** — A multicast DLCI can have at most 32 members.

164 **cveLockFailed** — Lock failed, the object is locked by others.

165 **cveUnLockFailed** — Unlock failed, the object is locked by others.

166 **cveCardExtlClockValue** — The external clock rate must be a multiple of 8Kps and cannot be higher than 1.544Mbps.

167 **cveObjNameTooLong** — The object name entered is too long (max. 32 chars).

168 **cveDiagAdminNotDown** — The test cannot be run because the admin. status for the target object(s) is not ‘down’.

169 **cveArgRange** — Arguments Required are out of range.

170 **cveUIRead** — Error in reading user input data.

171 **cveCfm_CommitPPort** — Changes in PPort Attributes must be committed before setting LPort attributes. Click OK to commit changes.

172 **cveCfm_NmsLostContact** — This operation will cause the NMS to lose connectivity to all switches.

173 **cveCfm_SaveChanges** — Do you want to save the changes?

174 **cveCfm_DisruptUserData** — This Operation may disrupt user data. Do you really want to continue?

175 **cveCfm_Logoff** — Do you really want to logoff?

176 **cveCapabilityMismatch** — + Capability mismatch.

177 **cveLPortDelMgmtAddrDefined** — Cannot delete Logical Port while there is SMDS Management Address defined.

178 **cveMgmtSsiAddrNotDefined** — No feeder address has been defined on the select SSI-DTE logical port yet.

179 **cveMgmtSsiAddrDifferent** — The Management Address must be the same as the feeder address defined on the selected SSI-DTE logical port.

180 **cveCfm_DownloadSW** — Do you really want to download the selected software to the switch?

181 **cveCpSlot2Error** — + Undefined Control/Switch Processor card detected in slot 2 .

182 **cveLPortDelDteAddrExist** — Cannot delete SSI logical port while there are Smds Addresses defined in the address pool.

183 **cveCfm_DownloadPramFile** — Do you really want to download the selected pram file to the switch?

184 **cveHssiPPortBwTooHigh** — Error: Total HSSI PPort bandwidth has exceeded the card bandwidth of 44212000 bps.

185 **cveCfm_RemoveIsdnAa** — Changing the B-Channel type from ISDN to Switched 56 will result in the loss of the list of selected Authentication Addresses. Do you really want to continue?

186 **cveCfm_ChgAtmProtocol** — Do you wish to reset all signalling tuning attributes to their ATM protocol specific default values?

187 **cveCfm_CommitChannel** — Changes in Channel Attributes must be committed before setting LPort attributes. Click OK to commit changes.

188 **cveCfm_ChgAtmUniVersion** — Do you wish to reset all signaling tuning attributes to their UNI version specific default values?

189 **cveInvalidADSvr1Addr** — The server 1's IP address is invalid.

190 **cveInvalidADSvr2Addr** — The server 2's IP address is invalid.

191 **cveInvalidADSvr3Addr** — The server 3's IP address is invalid.

192 **cveInvalidObjName** — Invalid character in the object name.

193 **cveCfm_CommitPPortChan** — Changes in PPort Attributes must be committed before setting Channel attributes. Click OK to commit changes.

194 **cveCfm_DelSecScn** — There exists LPorts to which this screen is assigned to. Click OK to confirm the delete.

195 **cveCfm_LockoutOfProtection** — Sending the Lockout of Protection command to the switch will prevent the working channel from switching to the protection line. Do you really want to continue?

196 **cveCfm_ForcedWorkToProtectSwitch** — Sending this Forced Switch command to the switch will cause the switch to change the specified working channel to the protection line regardless of any failures that may occur. Do you really want to continue?

197 **cveCfm_ForcedProtectToWorkSwitch** — Sending this Forced Switch command to the switch will cause the switch to change the working channel

back from the protection line to the working line regardless of any failures that may occur. Do you really want to continue?

198 **cveCfm_svcRelCall** — Are you sure you wish to release the selected SVC?

199 **cveInvalidDlciId** — The specified DLCI Number is not valid in this location.

200 **cveCfm_CommitAuthenDomainDown** — Warning: Setting the authentication domain admin status down will prevent any console login to the associate switch which has the console authentication enabled. Do you really want to proceed?

201 **cveCktFail_AdminDown** — Circuit admin status is down.

202 **cveCktFail_NoVcbuff** — Internal error: No VC buffer.

203 **cveCktFail_NoBandwidth** — Not enough bandwidth.

204 **cveCktFail_NoRoute** — Destination node is unreachable.

205 **cveCktFail_Timeout** — PVC segment call is timed out.

206 **cveCktFail_NoPduBuff** — Internal Error: No PVC PDU buffer.

207 **cveCktFail_NoDest** — Destination DLCI is not configured.

208 **cveCktFail_TrunkRnr** — Optimum path trunk data flow is blocked.

209 **cveCktFail_TrunkDown** — Trunk is down.

210 **veCktFail_BalanceReroute** — Load balance reroute.

211 **cveCktFail_Dead** — No response from the remote switch beyond.

212 **cveCktFail_DefPathReroute** — Attempting to reroute through the defined path.

213 **cveCktFail_N1Down** — UNI/NNI is down.

214 **cveCktFail_OtherPvcSecDown** — PVC segments are inactive beyond.

215 **cveCktFail_OtherPvcSegRnr** — PVC segments are not ready to receive data beyond.

216 **cveCktFail_UsingAltPathWarning** — Defined path is NOT available. The alternate path is being used.

217 **cveCktFail_IopDown** — IOP/IOM is down.

218 **cveCktFail_NoPduMsg** — No PVC Manager PDU msg buffer, call Cascade (1-800-DIALWAN).

219 **cveCktFail_NoPort** — Port is not Configured.

220 **cveCktFail_Misconfig** — Mis-configuration.

221 **cveCktFail_SVCSetupFail** — SVC setup failed.

222 **cveCktFail_SrcBackedup** — Source is in a ‘backup’ condition.

223 **cveCktFail_SrcUnknown** — Source is unknown.

224 **cveCktFail_DestUnknown** — Destination is unknown.

225 **cveCktFail_BuDLCICollision** — DLCI collision on backup.

226 **cveCktFail_OldVerInPath** — Node running incompatible version of switch software exists in circuit path.

227 **cveCktFail_SmdsMgmtTrunk** — SMDS management trunk.

228 **cveCktFail_NeverCalled** — Endpoint never called.

229 **cveCktFail_BothEndPtBackup** — Both Endpoints in ‘backup’.

230 **cveCktFail_PvcRouteMgtTrunk** — Attempting to route through management trunk.

231 **cveCktFail_NoMultiPointParent** — Multipoint parent not found.

232 **cveCktFail_PvcRouteFail** — Route changed during setup.

233 **cveCktFail_NoVpiVci** — No VPI and VCI available.

234 **cveCktFail_internalreset** — SVC cleared by user.

235 **cveCktFail_PathRegister** — Circuit path registration failed.

236 **cveCktFail_AtmChannel** — Selected channel cannot be allocated.

237 **cveCktFail_NoReverseBw** — No available bandwidth in reverse direction.

238 **cveCktFail_InternalReset** — PVC reset internally - reuse path.

239 **cveCktFail_PriRouteDisrupt** — Disrupted due to priority routing.

240 **cveCktFail_NoNegPriBw** — Couldn't allocate negative priority bandwidth.

241 **cveCktFail_NoDestVpiVci** — Destination VPI/VCI is not configured.

242 **cveCktFail_BkpVpiVciColl** — Backup call fails because VPI/VCI in use.

243 **cveCktFail_SmdsFrameOnly** — If SMDS VP connection try to go over Frame-Only card.

244 **cveCktFail_CacNotInit** — CAC not initialized.

245 **cveCktFail_CacSigError** — Signalling error: SCR maybe greater than the PCR or PCR equal to zero.

246 **cveCktFail_CacLargeMbs** — MBS is larger than the maximum configured MBS for the customized CAC.

247 **cveCktFail_CacCdvNotAvail** — Configured CDV objectives cannot be satisified for the CBR QoS Class.

248 **cveCktFail_QOSUnavail** — Requested QOS is unavailable.

251 **cveMaxOneConfigDlg** — Request Aborted: Please finish the current configuration task before starting another one.

252 **cveCktSegSizeNotSupported** — QuickPath is not currently supported; the segment size must be 0.

253 **cveCfgSyncDownloadFailed** — The download of the configuration-sync file failed.

254 **cveInvalidTrkUtilization** — Invalid Over Subscription Factor value.

255 **cveNullLPortName** — Logical port name cannot be empty.

256 **cveNullTrunkName** — Trunk name cannot be empty.

257 **cveNullCircuitName** — Circuit name cannot be empty.

258 **cveNullMgmtDlciName** — Mgmt DLCI name cannot be empty.

259 **cveLPortDelMulSsiExist** — The logical port cannot be deleted because it is in use by multiplexed DXI SNI lports.

260 **cveLPortDelIaDefined** — The logical port cannot be deleted because there is an SMDS Address defined in it.

261 **cveCardSmdsLPortsExist** — SMDS Logical ports are defined on the card.

262 **cveSmdsIaObjectNotFound** — The SMDS IA Object could not be found.

263 **cveInvalidSsiLPortName** — Invalid Ssi LPort name.

264 **cveMaxAddrNameSizeExceeded** — Address Name is too long.

265 **cveSmdsAaObjectNotFound** — The SMDS AA Object could not be found.

266 **cveSmdsAaLPortTypeInvalid** — Cannot define SMDS AA Object in the current logical port type.

267 **cveMaxSmdsAaExceeded** — Maximum number of SMDS AA has been exceeded.

268 **cveSmdsAaObjectInScreen** — Cannot delete the S AA Object which is associated with other screen objects.

269 **cveSmdsAddrPoolObjectNotFound** — The SMDS Address Pool Object cannot be found.

270 **cveSmdsAddrPoolLPortTypeInvalid** — Cannot define SMDS Address Pool Object in the current logical port type.

271 **cveSmdsDteAddrNotUnique** — Duplicate SMDS Dte Address.

272 **cveSmdsAddrSizeInvalid** — Invalid SMDS Address size, total # of digits including the address prefix and mask is: 10 for North American Address format and between 11 and 15 for International Address Format.

273 **cveSmdsAddrInvalid** — Invalid SMDS Address.

274 **cveSmdsDteAddrObjectNotFound** — The SMDS Dte Address object cannot be found.

275 **cveSmdsDteAddrObjectUsed** — Cannot delete an SMDS Dte Address while it is being used.

276 **cveSmdsGaObjectNotFound** — The SMDS GA Object cannot be found.

277 **cveMaxSmdsGaExceeded** — Maximum number of SMDS GA Object has been exceeded.

278 **cveSmdsGaObjectHasMember** — Cannot delete an SMDS GA Object while it contains members.

279 **cveSmdsGaInScreen** — Cannot delete an SMDS GA Object while it is associated with a screen.

280 **cveSmdsNetGaObjectNotFound** — The SMDS Net GA Object cannot be found.

281 **cveSmdsIaLPortTypeInvalid** — Cannot define SMDS IA Object in the current logical port type.

282 **cveSmdsIaNotUnique** — Duplicate SMDS IA Address.

283 **cveMaxSmdsIaExceeded** — Maximum number of SMDS IA Object has been exceeded.

284 **cveSmdsIaInGa** — Cannot delete an SMDS IA Object while it is associated with a SMDS Group Address.

285 **cveSmdsIaInScreen** — Cannot delete an SMDS IA Object while it is associated with a SMDS screen.

286 **cveSmdsMaskNotUnique** — The masked portion of the defined address is already used on another switch.

287 **cveSmdsMaskTooLong** — SMDS Address Prefix cannot be longer than 8 digits.

288 **cveSmdsMaskTooShort** — SMDS Address Prefix cannot be less than 3 digits.

289 **cveSmdsMaskObjectNotFound** — The SMDS Prefix Object cannot be found.

290 **cveSmdsMgmtAddrObjectNotFound** — The SMDS Mgmt Address Object cannot be found.

291 **cveSmdsMgmtAddrLPortTypeInvalid** — SMDS Mgmt Address Object cannot be defined in the current logical port type.

292 **cveSmdsNGaObjectHasMember** — Cannot delete an SMDS Net GA Object while it contains members.

293 **cveSmdsScnObjectNotFound** — The SMDS Screen Object cannot be found.

294 **cveSmdsScnLPortTypeInvalid** — Cannot define SMDS Screen Object in the current logical port type.

295 **cveSmdsScnExist** — An existing SMDS Screen Object of the same type has been defined in the current logical port.

296 **cveSmdsScnObjectHasMember** — Cannot delete an SMDS Screen Object while it contains members.

297 **cveIPAddrSizeInvalid** — Invalid IP Address size.

298 **cveSmdsNGaDefined** — An existing SMDS Net Group address has been defined in the network.

299 **cveSmdsNGaObjectNotFound** — The SMDS Net GA Object cannot be found.

300 **cveSmdsMgmtAddrExists** — The SMDS Mgmt Address Object has been defined previously.

301 **cveInvalidMgmtAddrName** — Invalid management Address name.

302 **cveAddSmdsLPortInFrameRelay** — Cannot add SMDS Logical ports to a card with Frame Relay Only capability.

303 **cveLPortDelScnMbrExist** — Cannot delete Logical ports while there are screen members defined.

304 **cveDelMgmtAddrWithNmsPath** — Cannot delete the Management Address entry while it is used in a NMS Path.

305 **cveDelLastSmdsGaMem** — Cannot delete the last member from the Group Address which is included as a screen member. Please remove the group screen membership first.

306 **cveNoSSIFeederAddr** — Cannot define Management Address before the Feeder Address is defined.

307 **cveSmdsAaNotUnique** — Duplicate SMDS Alien Address.

308 **cveDelSmdsAaWithScreen** — Cannot delete Smds AA while it is still associated with more than one screen.

309 **cveSmdsGaNotUnique** — Duplicate SMDS GA Address. The SMDS GA may have been defined as an Alien Group Address in the switch that contains the selected member.

311 **cveInitDlg** — Error initializing the dialog box.

353 **cveSmdsInvalidMaskSize** — Number of digits in the Smds Prefix is different than the Network wide Prefix Size set in cascadeview.cfg file.

354 **cveDeleteMaskWithSmdsIa** — Cannot delete the Smds Address Prefix while there is Smds IA defined on it.

355 **cveInvalidHBPInterval** — Heart Beat Poll Interval value must be between 1 and 40 seconds.

356 **cveInvalidHBPNAThreshold** — Heart Beat Poll NA Threshold value must be between 1 and 255.

357 **cveBillingInvalid** — Billing Enabled/Disabled is only supported for Smds Dxi logical lports.

358 **cveBillingAggPeriodInvalid** — Billing aggregation period must be greater than 10 minutes.

359 **cveBillingCollPeriodInvalid** — Billing collection period must be greater than 30 seconds.

360 **cveSmdsSSINoFdrAddr** — Error getting the Feeder address of the selected SMDS SSI DTE Logical Port.

361 **cveChangeSsiWithSmdsIa** — Cannot change the multiplexed SSI logical port while there are Smds IA's defined.

362 **cvePPortAtmUniMaxBufferSize** — The Max. Buffer Size must be between 48 and 8192 bytes.

363 **cvePPortAtmUniPeakCellRate** — The Peak Cell Rate must be between 256 and 96000 cells/sec.

364 **cvePPortPCQInUseTooLow** — One of the Peak Cell Rate values is not high enough to support all configured logical ports.

365 **cveRerouteCountInvalid** — Reroute Count value is invalid (1-64).

366 **cveRerouteDelayInvalid** — Reroute Delay value is invalid (1-32767).

367 **cvePVCLoopbacksInvalid** — Invalid PVC Loopback combination.

368 **cveInvalidMaxSegSize** — Maximum Segment Size must be a multiple of 32, up to 4144.

369 **cveDiffMaxSegSize** — WARNING: There has already been a map using the same network number but with different Maximum Segment Size; the new Maximum Segment Size will be ignored.

370 **cveModifySwAdvanAttr** — Modifying switch tuning attributes may cause serious traffic and disrupt user data. Do you really want to continue?

371 **cveBurstToleranceInvalid** — The Maximum Burst Size must be between 1 and 63.

372 **cvePcqIndexInvalid** — The Peak Cell Rate index must be between 0 and 7.

373 **cveScrInvalid** — The Sustainable Cell Rate must be less than PCR and greater than (PCR/63).

374 **cveLPortPPortConflict** — This logical port cannot be added to this physical port.

375 **cveLPortDelAtmXportFeeder** — A feeder port cannot be deleted while one or more ATM Transport for FR logical ports are defined on the same physical port or channel.

376 **cveCktScr0BtNon0** — When SCR is 0, the MBS must also be 0.

377 **veCktScrNon0BtInvalid** — When SCR is non-zero, the MBS must be a multiple of SCR.

378 **cveCktPcrTooHigh** — The value (PCR * 48 * 8) must be no higher than the logical port bandwidth.

379 **cveCktXlationInvalid** — Invalid value for translation type.

380 **cveSNameObjectNotFound** — The service name could not be found.

381 **cveSNameDelDlciDefined** — The service name cannot be deleted because it is in use as a circuit endpoint.

382 **cveSNameNameInUse** — The specified service name is not unique.

383 **cveNullSNameName** — Service Name name cannot be empty.

384 **cveLPortDelSNameDefined** — The logical port cannot be deleted because it is bound to a service name.

385 **cveSNameOnFROnly** — The logical port cannot be bound to a service name because it is not of type FR UNI or FR NNI.

386 **cveLPortModDlciDefined** — The logical port cannot be designated as a 'backup' because a circuit is defined on it.

387 **cveLPortModSNameDefined** — The 'backup' designation cannot be removed from the logical port because a service name is defined on it.

388 **cveLPortPrimaryForSName** — The logical port cannot be designated as a 'backup' because it is bound to a service name as a primary logical port.

389 **cvePvcAtmUniGracefulDiscard** — For a circuit with one ATM UNI endpoint on an ATM UNI DS3 card,'Graceful Discard' must be OFF.

390 **cvePvcAtmFrBurstTolTooHigh** — For a circuit with one ATM UNI endpoint on an ATM UNI DS3 card, (Bc + Be)/48/8 must be less than or equal to 2016.

391 **cveCktPcrInvalid** — PCR must be non-zero.

392 **cveCktScrInvalid** — SCR must be non-zero and less than or equal to PCR.

393 **cveCktBurstTolInvalid** — Maximum Burst Size cannot be less than 32 or greater than 2016.

394 **cveCktScrInvalidDs3** — For circuits on DS3 ATM cards, SCR cannot be less than PCR/64 or greater than PCR.

395 **cveCktBurstTolInvalidDs3** — For circuits on DS3 ATM cards, the MBS cannot be less than 1 or greater than 2016.

396 **cveCktAtmDs3PcrNotInQ** — The PCR value must be chosen from the values assigned to the physical ports' rate queues.

397 **cveCktSNamePathsNotAllowed** — Defined paths are not allowed for circuits with service name endpoints.

398 **cveLPortBuNotOkForPrimary** — The selected backup logical port cannot backup the defined primary logical port.

399 **cveLPortNotPrimary** — A backup logical port cannot be bound to a service name as a primary logical port.

400 **cveLPortNotBackup** — The logical port must be designated as a 'backup' before it can be bound to a service name as a backup logical port.

401 **cveCktLPortIsBackup** — A circuit cannot be defined on a logical port designated as a backup.

402 **cveMcastDlciLPortIsBackup** — A multicast DLCI cannot be defined on a logical port designated as a backup.

403 **cveMgmtDlciLPortIsBackup** — A management connection cannot be defined on a logical port designated as a backup.

404 **cveCktBothEndptsVariable** — A circuit cannot have both endpoints defined on service names.

405 **cveBuLPortOnFROnly** — The logical port cannot be designated as a 'backup' because it is not of type FR UNI or FR NNI.

406 **cveNoAtmUniDteLPort** — Cannot add the current LPort when there is no ATM feeder logical port defined.

407 **cveOneCktOnFradLport** — Cannot define more than one PVC on an Encapsulation FRAD logical port.

408 **cveE1CardExtlClockValue** — The external clock rate must be a multiple of 8Kps and cannot be higher than 2.048Mbps.

409 **cveInvalidLPortBw** — Invalid Logical Port Bandwidth.

410 **cveNetAddrPrefixTooLong** — Country Prefix cannot be longer than 6 digits.

411 **cveCktMbsMulltiplePcr** — The MBS must a non-zero multiple of the PCR.

412 **cveInvalidVpiId** — Virtual Path ID must be between 0 and 15.

413 **cveZeroVpiId** — Virtual Channel ID cannot be less than 32 if Virtual Path ID is 0.

414 **cveInvalidFrVciId** — Virtual Channel ID must be between 32 and 63 for frame-based ATM endpoints.

415 **cveInvalidAtmVciId** — Virtual Channel ID must be between 32 and 255.

416 **cveSynchCpFirst** — CP/SP is out of synchronization. Synchronize the CP/SP first, then synchronize this IOP/IOM.

417 **cveNoCpFound** — Error finding CP/SP from the database.

423 **cveLPortInvalidUniType** — The UNI type is invalid.

434 **cveLPortTooManyVpiVciBits** — The total number of valid VPI and VCI bits must be 14 or less.

435 **cveLPortTooFewVpiBits** — The number of valid VPI bits cannot be less than zero.

436 **cveLPAtmProtocolConnTypeInv** — Connection type must be \"network <-> network\" if the ATM protocol is IISP 3.1.

437 **cveLPortInvalidBwPolicing** — The bandwidth policing option is invalid.

438 **cveLPortConnTypeInv** — The connection type is invalid.

439 **cveLPortInvalidIlmiAdminStatus** — The ILMI admin status is invalid.

430 **cveLPortInvalidIlmiPollPeriod** — The ILMI poll period value is invalid.

431 **cveLPortInvalidIlmiLossThresh** — The ILMI loss threshold value is invalid.

432 **cveLPortInvalidSigAdminStatus** — The signaling admin status is invalid.

433 **cveLPortInvalidIlmiPfxScreen** — The ILMI DTE prefix screen mode is invalid.

434 **cveQoSCbrRMInvalid** — The CBR QoS Routing Metric is invalid.

435 **cveQoSVbrRTRMInvalid** — The VBR Real Time QoS Routing Metric is invalid.

436 **cveQoSVbrNRTRMInvalid** — The VBR Non-Real Time QoS Routing Metric is invalid.

437 **cveQoSUbrRMInvalid** — The UBR QoS Routing Metric is invalid.

438 **cveLPortInvalidOamAlmTimerThresh** — The OAM alarm timer threshold value is invalid.

439 **cveLPortInvalidSigTuningValue** — An invalid signaling tuning value has been entered.

440 **cveCktVciInvalidForCktType** — One or both VCI values are invalid for the selected circuit type.

441 **cveInvalidCellBasedAtmVccVpiId** — VPIs for a VCC must be greater than or equal to 0 and less than 2 to the power N where N is the number of VPI bits valid for the logical port endpoint.

442 **cveInvalidCellBasedAtmVciId** — VCIs for a VCC must be greater than 31 and less than 2 to the power N where N is the number of VCI bits valid for the logical port endpoint.

443 **cveCardInvalidSysClockSrc** — The card's system clock source must be a valid physical port ID.

444 **cveCardInvPortSysClockCombo** — An invalid card system clock source and port type combination has been selected.

445 **cvePPortNoOc3CbitParity** — C-Bit parity is not allowed on OC-3 cards.

446 **cveCardOpticalNotSupported** — The card does not support optical attributes.

447 **cvePmpCktRootWithLeafs** — Request failed: Point-to-Multiple-Point Circuit Root has one or more corresponding leafs defined.

448 **cvePmpCktRootNotFound** — Cannot find the corresponding Point-to-Multiple-Point Circuit Root record.

449 **cvePmpCktLeafNotFound** — Cannot find the corresponding Point-to-Multiple-Point Circuit Leaf record.

450 **cveIsdnNoAaIfBChanUndef** — B-Channel attributes must be defined before authentication addresses can be defined.

451 **cveNoIsdnPriSwType** — The ISDN PRI Switch Type is invalid for the ISDN PRI T1 Card.

452 **cveInvalidIsdnPriSwType** — The ISDN PRI Switch Type should not be set for non-ISDN PRI T1 Card.

453 **cveIsdnAddrDefined** — Duplicate ISDN E164 Address.

454 **cveIsdnInvalidAddr** — Invalid ISDN E164 Address.

455 **cveIsdnAddrNotFound** — Unable to find the given ISDN E164 address in the database.

456 **cveIsdnAddrInUse** — Unable to delete the given ISDN E164 address as it is referenced by others.

457 **cveIsdnNoAaForSw56** — Authentication addresses cannot be defined for Switched 56 logical ports.

458 **cvePPortInvalidIsdn** — ISDN PRI can only be enabled on ISDN PRI T1 physical ports.

459 **cvePPortNoIsdnPriChgLPDef** — Unable to change the ISDN PRI setting as there are logical ports defined on this physical port.

460 **cveLPortDChanIsdnPriDis** — The ISDN D-Channel logical port cannot be added because its physical port does not have ISDN PRI enabled.

461 **cveLPortDChanMustBeFirst** — The ISDN B-Channel logical port cannot be added because there is no ISDN D-Channel logical port defined on its physical port.

462 **cveLPortTooManyIsdnDs0s** — A single DS0 must be selected for an ISDN logical port.

463 **cveLPortInvalidDChanDs0** — An ISDN D-Channel logical port must be defined on DS0 24 only.

464 **cveLPortInvalidBChanDs0** — An ISDN B-Channel logical port must be defined on one of DS0 1 through 23 only.

465 **cveLPortDelDChan** — The ISDN D-Channel logical port cannot be deleted because at least one ISDN B-Channel logical port is still defined on its physical port.

466 **cveLPortNoDChanTypeChg** — The logical port type of an ISDN D-Channel logical port cannot be changed.

467 **cveIsdnBChanLPortNotFound** — The logical port of the ISDN B-Channel does not exist.

468 **cveIsdnBChanTypeInvalid** — A non-D-Channel ISDN logical port must be either of type ISDN B-Channel or Switched 56.

469 **cveIsdnBChanBaMissing** — An ISDN B-Channel address has not been selected.

470 **cveIsdnBChanSw56WithBaAa** — An ISDN Switched 56 logical port cannot have a B-Channel address or authentication addresses.

471 **cveIsdnBChanTooManyAaMbrs** — An ISDN B-Channel logical port can only have a maximum of 10 authentication addresses.

472 **cveNoIsdnNfasValue** — The selected number of D-Channels on the card is invalid.

473 **cveCardNfas1DChanWithPPPriOff** — All physical ports must have ISDN PRI on if the allowed number of ISDN D-Channels on the card is set to one.

474 **cveLPortDChanNotOnPPort1** — This card is configured to only support one ISDN D-Channel defined on physical port number one.

475 **cveRerouteAlgInvalid** — Load Balancing Algorithm value is invalid.

476 **cveLPortLmiDelayInvalid** — Lmi Update Delay value is invalid.

480 **cveModNmsEntryNeedPramSync** — NMS entries cannot be changed dynamically in the switch. They can only be maintained through PRAM synchronization or script download.

481 **cveZeroCirForFrAtmCkt** — CIR must be non-zero for FR <-> ATM circuits.

482 **cveInvalidSmdsPduViolThr** — SMDS Pdu Violation Threshold must be between 0 and 255.

483 **cveNoChannelSelected** — At least one channel must be selected.

484 **cveNonConsecutiveChs** — Non consecutive channels are selected.

485 **cveNoSrcLport** — A source Logical Port must be selected.

486 **cveNoDesLport** — A Destination Logical Port must be selected.

487 **cveSameSrcDesLports** — The Source and Destination Logical Ports must be different.

488 **cveNoMoveSrvCkt** — Cannot move circuit with one endpoint defined on service name.

489 **cveNoMovePathedCkt** — Cannot move circuit with circuit path defined in it.

490 **cvePramULNoSuchCard** — Unable to construct the object from PRAM because its card has not been defined.

491 **cveOfflinePramFileOFail** — Error opening offline pram input file.

492 **cveOfflineDirCreate** — Failed creating offline pram file directory.

493 **cveNoOfflinePramItem** — No input offline pram item.

494 **cveOfflineEmptySlot** — No offline pram file for empty slot and redundant slot.

495 **cveNULLPramFile** — Pram file cannot be empty.

496 **cveModifyLBLport** — Cannot modify logical port while the logical port is in loopback state.

497 **cveInvalidDirToken** — Invalid pram file directory token.

498 **cvePramMisMatchSwVer** — Pram file does not match switch software version.

499 **cveAutoDetectOnNni** — Cannot set Link Mgmt Protocol to autoDetect if the logical port type is NNI.

500 **cveCardOneSysClockPort** — Only one physical port per node can serve as a primary or secondary system clock source.

501 **cveInvalidCgPtyInsertAddrKey** — Invalid Calling Party Insertion Address database key.

502 **cveLPortInvalidIlmiVpi** — The ILMI VPI must be greater than or equal to 0 and less than 2 to the power N where N is the number of VPI bits valid for this logical port.

503 **cveLPortInvalidIlmiVci** — The ILMI VCI must be greater than or equal to 0 and less than 2 to the power N where N is the number of VCI bits valid for this logical port.

504 **cveCktCellBasedPcrTooHigh** — The value (PCR * 53 * 8) must be no higher than the physical port bandwidth.

505 **cveLPortInvalidUpc** — The UPC function option is invalid.

506 **cveInvalidAtmTrkVpiId** — VPI must be greater than 0 and less than 2 to the power of N, where N is the number of VPI bits for the ATM feeder Logical Port.

507 **cveInvalidAtmTrkVciId** — VCI must be greater than 31 and less than 65535.

508 **cveLPortInvalidAtmTrkVpiVci** — VPI must be greater than or equal to 0 and less than 2 to the power N where N is the number of VPI bits valid for the ATM feeder logical port, and VCI (if required) must be greater than 31 and less than 2 to the power M where M is the number of VCI bits valid for the ATM feeder logical port.

509 **cveLPortTooFewVciBits** — The number of valid VCI bits must be greater than 5.

510 **cvePPortLPBwTooHighForPlcp** — PLCP cannot be enabled on this physical port because it will reduce the physical port data rate below that which is required by the member logical ports.

511 **cveInvalidMgmtVpi** — Management VPIs must be greater than or equal to 0 and less than 2 to the power N where N is the number of VPI bits valid for the logical port.

512 **cveInvalidMgmtVci** — Management VCIs must be greater than 31 and less than 2 to the power N where N is the number of VCI bits valid for the logical port.

513 **cveInvalidMgmtVpiVciName** — Invalid management VPI/VCI name.

514 **cveLPortCdvTooSmall** — Logical port cell delay variation cannot be less than 0.

515 **cveLPortInvalidDs3TrkCdv** — Invalid DS-3 logical port direct trunk cell delay variation.

516 **cveLPortInvalidOc3TrkCdv** — Invalid OC-3 logical port direct trunk cell delay variation.

517 **cveCktMbsInvalid** — MBS must be non-zero and less than or equal to 65535.

518 **cveVpi1InUse** — The VPI entered for endpoint one is being used by another VPC or a VCC on the logical port.

519 **cveVpi1InUseByVpc** — The VPI entered for endpoint one is being used by a VPC on the logical port.

520 **cveVpi2InUse** — The VPI entered for endpoint two is being used by another VPC or a VCC on the logical port.

521 **cveVpi2InUseByVpc** — The VPI entered for endpoint two is being used by a VPC on the logical port.

522 **cveLPortPvcUsingVpiSpace** — The number of valid VPI bits cannot be reduced below the value needed to support the largest PVC VPI on the logical port.

523 **cveLPortPvcUsingVciSpace** — The number of valid VCI bits cannot be reduced below the value needed to support the largest PVC VCI on the logical port.

524 **cveLPortTrkUsingVpiSpace** — The number of valid VPI bits cannot be reduced below the value needed to support the largest OPTimum trunk VPI in use on this logical port.

525 **cveLPortTrkUsingVciSpace** — The number of valid VCI bits cannot be reduced below the value needed to support the largest OPTimum trunk VCI in use on this logical port.

526 **cveTrunkLPortInvalid** — The specified logical port types cannot support a trunk.

527 **cveTrunkLPortTypeDiff** — The specified logical ports are not of the same type.

528 **cveTrunkLPortBwDiff** — The specified logical ports have different configured bandwidths.

529 **cveInvalidCellBasedAtmVpcVpiId** — VPIs for a VPC must be greater than or equal to 0 and less than 256.

530 **cveMvCktInvalidSwitch** — Circuits cannot be moved either to or from logical port endpoints residing on CBX-500 switches.

531 **cveLeafVpiInUse** — The entered leaf VPI is being used by another VPC or a VCC on the logical port.

532 **cveLeafVpiInUseByVpc** — The entered leaf VPI is being used by a VPC on the logical port.

533 **cveLeafVpiVciInUse** — The entered leaf VPI/VCI is being used on the logical port.

534 **cveRootVpiInUse** — The entered root VPI is being used by another VPC or a VCC on the logical port.

535 **cveRootVpiInUseByVpc** — The entered root VPI is being used by a VPC on the logical port.

536 **cveRootVpiVciInUse** — The entered root VPI/VCI is being used on the logical port.

537 **cveVpiInUseByVpc** — The entered VPI is being used by a VPC on the logical port.

538 **cveVpiVciOfFeederLPortDup** — The entered VPI/VCI is being used by the feeder logical port in the physical port.

539 **cveVpiOfVpcFeederLPortDup** — The entered VPI is being used by a VPC on the feeder logical port in the physical port.

540 **cveVpiOfFeederLPortDup** — The entered VPI is being used by a VPC or a VCC on the feeder logical port in the physical port.

541 **cveCktInvalidCdvTol** — CDV tolerance must be greater than zero and less then 65536 microseconds.

542 **cveCktFwdRevTfDNull** — Zero-valued traffic descriptors are allowed in either the forward or the reverse direction but not in both directions.

543 **cveCktNdcEnableInvalid** — NDC Enable value must be either On or Off.

544 **cveLPortInvalidCongThresh** — Severe Congestion Thresholds specify invalid settings.

545 **cveCardInvBulkStatsSettings** — Invalid Bulk Statistics Settings.

546 **cveCardInvBulkStatsCollectPeriod** — Invalid Bulk Statistics Collection Period.

547 **cveCktNdcObjectNotFound** — The Circuit NDC entry could not be found.

548 **cveCktNdcCktObjectNotFound** — The Circuit for Ckt NDC entry could not be found.

549 **cveCktNdcMaxCardEnable1** — The Maximum Limit of 360 NDC-enable endpoints per card have been reached on Endpoint 1.

550 **cveCktNdcMaxCardEnable2** — The Maximum Limit of 360 NDC-enable endpoints per card have been reached on Endpoint 2.

551 **cveSwitchBulkAddressZero** — Bulk Statistics Collector Address is not set. Do you really want to continue?

552 **cveLPortTooManyVpiBits** — The number of valid VPI bits cannot be greater than 8.

553 **cveLPortTooManyVciBits** — The number of valid VCI bits cannot be greater than 14.

554 **cveInvalidAtmNniVpcVpiId** — VPIs for a VPC must be greater than or equal to 0 and less than 4096.

555 **cveLPortInvalidSvcRanges** — The SVC VPI/VCI Ranges entered are invalid.

556 **cveLPortSvcPvcRangeError** — The specified SVC VPI/VCI Ranges are not within the VCC VPI/VCI Ranges.

557 **cveLPortSvcRangeConflict1** — The SVC VPI/VCI Ranges entered are in conflict with those used by another logical port or trunk.

558 **cveLPortSvcRangeConflict2** — The VPI entered is in conflict with those used by another logical port or trunk.

559 **cveLPortDelVUNIFeeder** — A feeder port cannot be deleted as long as one or more Virtual UNI logical ports are defined on the same physical port.

560 **cveCktVpiVciConflictIlmi1** — Invalid VPI/VCI due to conflict with ILMI VPI/VCI Settings of Logical Port Endpoint 1.

561 **cveCktVpiVciConflictIlmi2** — Invalid VPI/VCI due to conflict with ILMI VPI/VCI Settings of Logical Port Endpoint 2.

562 **cveLPortVpiVciConflictIlmi** — Invalid ILMI VPI/VCI due to conflict with PVC VPI/VCI Setting.

563 **cveLPortSvcVpcUsingVpiSpace** — The number of valid VPI bits cannot be reduced below the value needed to support the largest SVC VPI/VCI Range VPC VPI on this physical port.

564 **cveLPortSvcVccUsingVpiSpace** — The number of valid VPI bits cannot be reduced below the value needed to support the largest SVC VPI/VCI Range VCC VPI on this physical port.

565 **cveLPortSvcVccUsingVciSpace** — The number of valid VCI bits cannot be reduced below the value needed to support the largest SVC VPI/VCI Range VCC VCI on this physical port.

566 **cveNetDistAllNodeFailure** — Network-wide distribution failure encountered at every node in the distribution list.

567 **cveSvcCUGNameInUse** — The specified CUG name is already being used.

568 **cveSvcCUGNotFound** — The CUG does not exist.

569 **cveSvcCUGIdsExhausted** — All CUG identifiers are currently in use.

570 **cveSvcCUGMbrRuleInUse** — The specified CUG member rule is already being used.

571 **cveSvcCUGMRIdsExhausted** — All CUG member rule identifiers are currently in use.

572 **cveSvcCUGMbrRuleNotFound** — The CUG member rule does not exist.

573 **cveSvcCUGMRInvNmbPlan** — The CUG member rule numbering plan is invalid.

574 **cveSvcCUGMRNull** — The CUG member rule regular expression cannot be empty.

575 **cveSvcCUGMRTooManyStars** — The CUG member rule regular expression can only contain one '*' character.

576 **cveSvcCUGMRBadAesaDigit** — The CUG member rule AESA regular expression can only contain hexidecimal digits ('0'..'9' and 'A'..'F') and wildcard characters ('?' to replace a single digit and '*' to replace zero or more digits).

577 **cveSvcCUGMbrExists** — This CUG and member rule have already been associated.

578 **cveOverMaxSvcCUGPerNode** — This member rule cannot be associated with any additional CUGs because a node on which this member rule resides already has 1024 CUGs configured on it.

579 **cveOverMaxSvcCUGMRAssoc** — A member rule cannot be associated with more than sixteen CUGs.

580 **cveOverMaxSvcCUGAssoc** — A CUG cannot contain more than 128 member rules.

581 **cveSvcCUGMbrNotFound** — The CUG-to-member rule association does not exist.

582 **cveSvcCUGMRBadE164Digit** — The CUG member rule E.164 regular expression can only contain decimal digits and wildcard characters ('?' to replace a single digit and '*' to replace zero or more digits).

583 **cveSvcSecScnNameInUse** — The specified security screen name is already being used.

584 **cveSvcSecScnCallDirInv** — The value for security screen call direction is invalid.

585 **cveSvcSecScnTypeInv** — The value for security screen type is invalid.

586 **cveSvcSecScnIdsExhausted** — All security screen identifiers are currently in use.

587 **cveSvcSecScnNotFound** — The security screen does not exist.

588 **cveSvcSSAddrInvType** — The security screen address type is invalid.

589 **cveSvcSSAddrNull** — The security screen address regular expression cannot be empty.

590 **cveSvcSSAddrTooManyStars** — The security screen address regular expression can only contain one '*' character.

591 **cveSvcSSAddrBadAesaDigit** — The security screen address AESA regular expression can only contain hexidecimal digits ('0'..'9' and 'A'..'F') and wildcard characters ('?' to replace a single digit and '*' to replace zero or more digits).

592 **cveSvcSSAddrBadE164Digit** — The security screen address E.164 regular expression can only contain decimal digits and wildcard characters ('?' to replace a single digit and '*' to replace zero or more digits).

593 **cveSSScnSubAddrInvType** — The security screen sub-address type can only be 'AESA' or 'Disabled'.

594 **cveSvcPdSecScnNotFound** — The predefined security screen does not exist.

595 **cveSvcPdSecScnIgScnModeInv** — The predefined security screen ingress mode value is invalid.

596 **cveSvcPdSecScnIgDefScnInv** — The predefined security screen ingress default screen value is invalid.

597 **cveSvcPdSecScnEgScnModeInv** — The predefined security screen egress mode value is invalid.

598 **cveSvcPdSecScnEgDefScnInv** — The predefined security screen egress default screen value is invalid.

599 **cveSvcCUGMbrRuleIAInv** — The CUG member rule incoming access value is invalid.

600 **cveSvcCUGMbrRuleOAInv** — The CUG member rule outgoing access value is invalid.

601 **cveSvcCUGMbrICBInv** — The CUG-to-member rule association incoming calls barred value is invalid.

602 **cveSvcCUGMbrOCBInv** — The CUG-to-member rule association outgoing calls barred value is invalid.

603 **cveSvcSecScnBindingExists** — This security screen has already been applied to the logical port.

604 **cveOverMaxSvcSecScnPerLPort** — A logical port cannot have more than 16 security screens applied to it.

605 **cveSvcSecScnDupBinding** — A security screen with the same addresses has already been applied to the logical port.

606 **cveSvcSecScnBindingAdminInv** — The security screen binding admin status is invalid.

607 **cveSvcSecScnBindingNotFound** — The security screen binding does not exist.

608 **cveLPortSecScnsBound** — All SVC security screens applied to the logical port must be removed before the logical port can be deleted.

609 **cveSvcCUGMbrRuleNameInUse** — The specified CUG member rule name is already being used.

610 **cveSvcNPfxOverCUGMRPerNode** — This SVC node prefix cannot be configured on this node due to the fact that the addition of the SVC CUG member rules matching this node prefix would exceed the maximum number of SVC CUG member rules that can be supported by the node.

611 **cveSvcPfxOverCUGMRPerNode** — CUG termination cannot be enabled for this SVC prefix on this node due to the fact that the addition of the SVC CUG member rules matching this prefix would exceed the maximum number of SVC CUG member rules that can be supported by the node.

612 **cveSvcAddrOverCUGMRPerNode** — CUG termination cannot be enabled for this SVC address on this node due to the fact that the addition of the SVC CUG member rules matching this address would exceed the maximum number of SVC CUG member rules that can be supported by the node.

613 **cveSvcUPOverCUGMRPerNode** — This SVC user part cannot be configured on this node due to the fact that the addition of the SVC CUG member rules matching this user part would exceed the maximum number of SVC CUG member rules that can be supported by the node.

614 **cveSvcNPfxOverCUGPerNode** — This SVC node prefix cannot be configured on this node due to the fact that the addition of the SVC CUGs associated with the SVC CUG member rules matching this node prefix would exceed the maximum number of SVC CUGs that can be supported by the node.

615 **cveSvcPfxOverCUGPerNode** — CUG termination cannot be enabled for this SVC prefix on this node due to the fact that the addition of the SVC CUGs associated with the SVC CUG member rules matching this prefix would exceed the maximum number of SVC CUGs that can be supported by the node.

616 **cveSvcAddrOverCUGPerNode** — CUG termination cannot be enabled for this SVC address on this node due to the fact that the addition of the SVC CUGs associated with the SVC CUG member rules matching this address would exceed the maximum number of SVC CUGs that can be supported by the node.

617 **cveSvcUPOverCUGPerNode** — This SVC user part cannot be configured on this node due to the fact that the addition of the SVC CUGs associated with the SVC CUG member rules matching this user part would exceed the maximum number of SVC CUGs that can be supported by the node.

618 **cveOverMaxSvcCUGMRPerNode** — This SVC CUG member rule cannot be added due to the fact that one of the nodes which must receive this member rule already has 1024 member rules configured on it.

619 **cveSvcNodePrefixAttrInvalid** — The SVC Node Prefix Attribute settings entered are invalid.

620 **cveSvcPrefixAttrInvalid** — The SVC Port Prefix Attribute settings entered are invalid.

621 **cveSvcAddressAttrInvalid** — The SVC Address Attribute settings entered are invalid.

622 **cveSvcAddressRegNotAllowed** — SVC Address Registration is not allowed for this prefix configuration.

623 **cveLPortSvcUsingVpcVpiSpace** — The number of valid VPI bits cannot be reduced below the value needed to support the largest specific PVP VPI connection ID in use on this physical port.

624 **cveLPortSvcUsingVccVpiSpace** — The number of valid VPI bits cannot be reduced below the value needed to support the largest specific PVC VPI connection ID in use on this physical port.

625 **cveLPortSvcUsingVccVciSpace** — The number of valid VCI bits cannot be reduced below the value needed to support the largest specific PVC VCI connection ID in use on this physical port.

626 **cveLPortInvalidPvcRanges** — The VUNI PVC Address Range entered are invalid.

627 **cveLPortVuniUsingVpiSpace** — The number of valid VPI bits cannot be reduced below the value needed to support the largest VP MUX PVC Range Stop value in use on this physical port.

628 **cveNullObjName** — The object's name cannot be empty.

629 **cveNoDeleteTfdUsed** — This traffic descriptor is being used and may not be deleted.

630 **cveLimitSpvcCard** — SPVC limit per card of 1024 has been reached. Addition may not be made.

631 **cveLimitTfdSwitch** — Traffic Descriptor limit per switch of 512 has been reached. Addition may not be made.

632 **cveNoTfdSelected** — Forward and Reverse Traffic Descriptors must be selected for this circuit.

633 **cveInvalidSvcFailTrapThresh** — The value for failure trap threshold cannot be less than zero or greater than 65535.

634 **cveInvalidSvcCdvT** — The value for CDV tolerance cannot be less than zero.

635 **cveLPortDefinedAtmVpi** — Invalid VPI due to the same VPI is used by another logical port.

636 **cveInvalidPartlyAtmTrkVpiId** — VPI must be greater than 0 and less than 15.

637 **cveNoMoveLoopbackCkt** — Cannot move circuit that is in pvc loopback.

642 **cvTfdQosRuleCBR** — Forward and Reverse Traffic Descriptors must comply with QoS Class Rules (CBR->,<-CBR).

643 **cvTfdQosRuleUnspecified** — Forward and Reverse Traffic Descriptors must comply with QoS Class Rules (Unspecified-XXX->,<-Unspecified-XXX).

644 **cvTfdQosRulePmpUBR** — Forward and Reverse Traffic Descriptors must comply with QoS Class Rules (VBR-RT,VBR-NRT,UBR->,<-UBR).

645 **cveCktInvalidMPVCvpi** — VPIs for a Management PVC logical port must be greater than or equal to 1 and less or equal to 15.

646 **cveCktInvalidMPVCvci** — VCIs for a Management PVC logical port must be greater than or equal to 32 and less or equal to 1023.

647 **cveSvcCUGMRE164BadLe**n — The CUG member rule E.164 regular expression cannot contain more than 15 characters.

648 **cveSvcCUGMRAesaBadLen** — The CUG member rule AESA regular expression cannot contain more than 40 characters.

649 **cveSvcSecScnE164BadLen** — The security screen address E.164 regular expression cannot contain more than 15 characters.

650 **cveSvcSecScnAesaBadLen** — The security screen address AESA regular expression cannot contain more than 40 characters.

651 **cveSvcSecScnUnknownBadLen** — The security screen address regular expression cannot contain more than 40 characters.

652 **cveSpvcScrInvalid** — SCR must be less than or equal to PCR.

653 **cveSpvcMbsInvalid** — MBS must be non-zero if PCR or SCR are non-zero.

654 **cveCktPcrTooHighForPPort** — The value (PCR * 48 * 8) must be no higher than the physical port bandwidth.

655 **cveModLPortNegativeBandwidth** — Changing the logical port bandwidth may cause a negative circuit bandwidth condition if the new value is not sufficient to support all of the previously provisioned PVCs on that port.

701 **cveMaxClockSourceExceeded** — The number of clock source entries for a switch is exceeded.

702 **cveInvalidPriority** — The range for the priority value is 1 to 20.

703 **cveClockSourcesExist** — Card is a possible source of a reference clock. Clock source entry must be deleted first.

704 **cvePPortInvalidCbrClockMode** — Invalid cbr clock mode setting for this port type.

705 **cveInvalidAAL1VpiVciId** — Invalid setting of VPI or VCI value for this port type.

706 **cvePPortInvalidLaserStatus** — Invalid laser status setting for this port type.

707 **cvePPortInvalidBipErrorsThresh** — Invalid setting of bit interleaved parity errors threshold for this port type.

708 **cvePPortAtmIwuOc3VpiBitsOutOfRange** — Invalid setting of VPI Length for this port type.

709 **cvePPortInvalidShaperPriority** — The range of values for shaper priority is 0 - 15.

710 **cvePPortShaperScrExceedClockSpeed** — The sustainable cell rate exceeds the port's data rate.

711 **cvePPortShaperPcrExceedClockSpeed** — The peak cell rate exceeds the port's data rate.

712 **cvePPortShaperScrHigherThanPcr** — The sustainable cell rate is higher than the corresponding peak cell rate.

713 **cveNewCardTypeNotSupported** — The specified card type is not supported by this switch version.

714 **cveOneCktOnAAL1Lport** — There is already a circuit defined using the selected logical ports. ATM AAL1 User Ports allow a single circuit only.

715 **cveCbrCktPortConflict** — One of the following conditions is violated: An AAL1 User Port can only be connected to:

- an ATM UNI port residing on an ATM IWU card.

- another AAL1 User Port if the connected cards are of the same type.

716 **cveLPortBwExceedPPortDataRateShaperChange** — As a consequence of the change of traffic shaper values, the total bandwidth of the logical ports on this pport would exceed the pport data rate.

717 **cveVpiBitsChangeWithLPortsDefined** — The VPI Length cannot be changed as long as there are logical ports configured for this physical port.

718 **cvePPortShaperScrTooLow** — The sustainable cell rate is too low.

719 **cvePPortInvalidShaperBurstTolerance** — The range of values for burst tolerance is 0 - 255.

720 **cveCktOspfCtdInvalid** — The range of values for cell transfer delay is 1 - 100 [milli seconds].

721 **cveCktOspfCdvInvalid** — The range of values for cell delay variation is 100 - 1000 [micro seconds].

722 **cveInvalidAtmIwuVciId** — Invalid Virtual Channel ID.

723 **cveInvalidAtmIwuVpiId** — Invalid Virtual Path ID.

724 **cveInvalidSvcRetryTimer** — Network Clearing Timer for SVC's must be between 0..255.

725 **cveInvSvcE164Len** — E.164 address must be 1..15 octet in length.

726 **cveInvSvcPrfxLen** — AESA Node/Port Prefixes must be 1..20 octet in length.

727 **cveInvSvcFmt** — Invalid Address Format.

728 **cveInvSvcDRouteLen** — Default Route must be 1 octet in length.

729 **cveInvSvcDRouteBLen** — Default Route must have zero number of bits.

730 **cveInvSvcAddrLen** — AESA Addresses must be 20 octet in length.

731 **cveInvSvcUPartLen** — User Part must be 7 octet in length.

732 **cveInvSvcOctetBit** — number of bits must be 8 times number of octets.

733 **cveInvSvcBitOctet** — number of bits must implies the number of octets.

734 **cveSvcCleared** — The selected SVC is already cleared.

750 **cveInvalidTrkAdminCost** — Trunk admin cost must be within the range of 1 to 65534.

751 **cveXfaceMismatch** — + Interface Mismatch.

760 **cveInvalidLPortOversub** — QoS oversubscription factors must be between 100% and 1,000%.

773 **cveMcastDlciObjectNotFound** — The multicast DLCI could not be found.

774 **cveMaxMcastGroupsExceeded** — The maximum number of multicast DLCI objects cannot exceed 32 per switch.

775 **cvePvcLoopOnQPathckt** — PVC loopback cannot be set on a circuit with quickpath option.

776 **cveBackupPortInUse** — The specified backup port is already in use.

777 **cveOneLportInDs0Lb** — Another logical port on the same physical port is already in DS0 loopback and only one logical port can be set to DS0 loopback.

778 **cveInNrEndLbNoFarEnd** — Cannot activate DS0 Far End Loopback because the logical port is already in DS0 Near End loopback.

779 **cveInNrEndLbNoStopFarEnd** — Cannot deactivate DS0 Far End Loopback because the logical port is in DS0 Near End loopback.

780 **cveInNrEndLbNoTestPtn** — Cannot start DS0 Far End Loopback test because the logical port is already in DS0 Near End loopback.

781 **cveInNrEndLbNoStopTestPtn** — Cannot stop DS0 Far End Loopback test because the logical port is in DS0 Near End loopback.

782 **cveInFarEndLbNoNrLb** — Cannot activate DS0 Near End Loopback because the logical port is already in DS0 Far End loopback.

783 **cveInFarEndLbNoStopNrLb** — Cannot deactivate DS0 Near End Loopback because the logical port is in DS0 Far End loopback.

784 **cveNotInFarEndLb** — The DS0 Far End Loopback is not activated on this logical port. Do you want to start DS0 Far End Loopback test?

785 **cveNoSmdsMaskDefined** — No Address Prefix has been defined yet.

786 **cveMvCktDiffServType** — Cannot move circuit to a logical port with different service type.

787 **cveDceETBigThanDceEC** — DCE Error Threshold must be less than or equal to DCE Event Count.

788 **cveDteETBigThanDteEC** — DTE Error Threshold must be less than or equal to DTE Event Count.

789 **veDcePVTLessThanDtePI** — DCE Poll Verify Timer must be greater than DTE Poll Interval.

790 **cveDlciId1InUse** — The DLCI entered for endpoint one is being used on the logical port.

791 **cveDlciId2InUse** — The DLCI entered for endpoint two is being used on the logical port.

792 **cveVpiVci1InUse** — The VPI/VCI entered for endpoint one is being used on the logical port.

793 **cveVpiVci2InUse** — The VPI/VCI entered for endpoint two is being used on the logical port.

794 **cveNoDelMcastCkt** — Cannot delete/move a circuit which is associated with a multicast DLCI.

795 **cveChannelInvalidClockSrc** — Invalid clock source setting for this channel type.

796 **cveChannelInvalidLineType** — Invalid line framing for this channel type.

797 **cveChannelInvalidZeroCoding** — Invalid zero coding for this channel type.

798 **cveChannelInvalidChannels** — Some of the allocated channels are not valid for this channel type.

799 **cveChannelObjectNotFound** — The channel object could not be found in the database.

800 **cveChannelBandwidthTooLow** — The specified bandwidth is too low to support all the logical ports defined on this channel.

801 **cveMaxLPortsPerChDs3Exceeded** — Maximum number of logical ports exceeded per ChDs3 channel.

802 **cveLPortIdChanInUse** — The specified logical port ID is already in use on this channel.

803 **cvePramULChanNoSuchPPort** — Unable to construct the channel from PRAM because its owning physical port does not exist.

804 **cveChannelInvalidds1CodeType** — Invalid ds1 loopback code type for channel type.

805 **cveDiagNoChannelSelected** — No channel has been selected yet, cannot perform the current operation.

806 **cveInvalidDs3IntervalInput** — Invalid input for Ds3 interval statistics.

807 **cveInvalidDs1IntervalInput** — Invalid input for Ds1 interval statistics.

808 **cveOptTrunkNoFeederChan** — An OPTimum trunk cannot be defined unless a feeder-type logical port of the same service type has already been defined on the same channel.

809 **cveLPortDelFeederChan** — A feeder port cannot be deleted as long as one or more OPTimum trunks of the same service type are defined on the same channel.

810 **cveLPortBandwidthTooHighChan** — More bandwidth was requested than the channel has available.

811 **cveDiagChanAdminNotDown** — The test cannot be run because the admin. status for the channel object is not ‘down’.

812 **cveInvalidAAL1Bw** — Both LPorts must have the same bandwidth.

813 **cveInvalidCUGrpName** — Invalid Close User Group obj name input (Max length 32 characters).

814 **cveInvalidCUGAddr** — Invalid SMDS Address input (Max length 15 digits).

815 **cveNoIScnDefined** — No Individual Screen defined for given lport.

816 **cveNoGScnDefined** — No Group Screen defined for given lport.

817 **cveAddrInOwnLport** — Address already defined in own lport*/

818 **cveNoMemoryActive** — swInfo is NULL in parseSwitchOperInfoRsp.

819 **cveInvalidSvcLoadBalance** — Svc Load Balance Eligibility Duration must be between 0..65535.

820 **cveCktInvalidMPVC** — Invalid end point for Multi hop MPVC.

821 **cveNoDelMPVCCkt** — Cannot delete a circuit which is used as a Management PVC NMS Path.

822 **cveTfdNameInUse** — The specified Traffic Descriptor Name is already in use on this Network.

823 **cveTfdInvalidName** — Empty Traffic Descriptor Name.

824 **cveNoDestAddress** — A destination Address must be entered.

825 **cveNullSmdsCUGName** — Smds Close User Group name cannot be empty.

826 **cveNullScnMbrName** — Smds IScreen/GScreen Member Address cannot be empty.

827 **cveMissingSharedSecret** — No shared secret has been entered yet.

828 **cveDomainServer1NotDefined** — The Primary Domain Server IpAddress cannot be empty.

829 **cveInvalidMaxRetries** — The maximum retries must be between 0 and 10.

830 **cveInvalidAuthenDomainType** — Invalid Authentication Domain type.

831 **cveDomainNameNotUnique** — The Authentication Domain name is not unique within the current map.

832 **cveIsdnBChanNoAuthenDomain** — No Authentication Domain name is specified yet.

833 **cveDelAuthenDomainInUsed** — Cannot delete the authentication domain because it is associated with logical port(s).

834 **cveMaxAuthenDomainReached** — Number of authentication domain cannot be more than 56 in a switch.

835 **cveSharedSecretTooLong** — The shared secret string cannot be longer than 32 char.

836 **cveInvalidTimedOutValue** — Timeout value must be between 1 to 10 seconds.

837 **cveIsdnAuthenNoAuthenDomain** — The modified Authentication Domain does not exist.

838 **cveLPortInvalidDChanTS16** — An ISDN D-Channel logical port must be defined on TS16 only.

839 **cvePPortTS16OffPriOn** — TS16 must be selected when ISDN PRI is on.

840 **cveInvalidIsdnEchoRqst** — Invalid ISDN Echo Request values.

841 **cveDupAuthenDomainIpAddr** — Duplicate Authen Domain Server IP Addresses within a domain.

842 **cvePramULIsdnAuthNoSuchCard** — Unable to construct the ISDN AuthenDomain Option from PRAM because its owning card does not exist.

843 **cvePramULIsdnAuthNoSuchPPort** — Unable to construct the ISDN AuthenDomain Option from PRAM because its owning physical port does not exist.

844 **cvePramULIsdnAuthNoSuchLPort** — Unable to construct the ISDN AuthenDomain Option from PRAM because its owning logical port does not exist.

845 **cvePramULIsdnAuthNoSuchAuthDomain** — Unable to construct the ISDN Authen Option from PRAM because its authentification domain does not exist.

846 **cveMgmtDlciDefinedOnLPort** — There is a Management DLCI already defined on the logical port.

847 **cveInvalidPrimaryIpAddr** — The Primary IP address is invalid.

848 **cveInvalidSecondaryIpAddr** — The Secondary IP address is invalid.

849 **cveSamePriSecAddr** — Primary and Secondary IP addresses may not be identical.

850 **cveSecondaryNullAddr** — While SVC or PVC Accounting is Enabled, the Secondary IP address must be assigned for Secondary Server Control.

851 **cvePrimaryNullAddr** — While SVC or PVC Accounting is Enabled, the Primary IP address must be assigned for Primary Server Control.

852 **cveNoDefaultAcctAddr** — Default Accounting address must be specified.

853 **cveCktAtmBillChgIdInvalid** — Invalid Chargeable Party ID.

854 **cveAtmBillInvalidASname** — Invalid Accounting Server Name.

855 **cveAtmBillInvalidASaddr** — Invalid Accounting Server IP Address.

856 **cveAcctServerDeleteInvalid** — Accounting Server may not be deleted while assigned to Switch.

857 **cveCktAtmBillChgIdInvDec** — Invalid Chargeable Party ID, only decimal digits allowed.

858 **cveCktAtmBillChgIdInvLen** — Invalid Chargeable Party ID, must be 15 decimal digits.

859 **cveLpAtmCarrIdInv** — Invalid PVC Carrier Party ID, its range is: (0..99999).

860 **cveLpAtmDefAddrInv** — Invalid Default Accounting Address.

861 **cveLpAtmDefAddrInvHex** — Invalid Default Accounting Address - expecting hexadecimal digits.

862 **cveNoMemoryCfgSyncFileGeneration** — CfgSyncFileGeneration return NULL in generateSyncFileReq.

863 **cveNoMemoryDownPdu** — downloadRunReq or downloadCheckReq return NULL in downloadSyncFileReq.

864 **cveNoMemoryCfgSyncFileDownload** — CfgSyncFileDownload return NULL in downloadSyncFileReq.

865 **cveNoMemoryiUpPdu** — downloadRunReq or downloadCheckReq return NULL in uploadSyncFileReq.

866 **cveNoMemoryCfgSyncFileUpload** — CfgSyncFileUpload return NULL in uploadSyncFileReq.

867 **cveNoMemorySerialNoSwitchSwRevNo** — serialNo or switchSwRevNo is NULL in parseSwitchOperInfoRsp.

868 **cveAcctServerDupName** — Accounting Server with same name exists.

869 **cveAcctServerDupIp** — Accounting Server with same IP address exists.

870 **cveNetworkAccounting_Warning** — Changing Network Accounting Parameters will affect all switches in the network. Are you sure?

871 **cveAcctServerDeleteConfirm** — About to DELETE the selected Accounting Server. Are you sure?

872 **cveCreateBackupVPIError** — To create a backup port, the number of VPI bits on the backup must be equal to the number of VPI bits on the primary.

873 **cveCreateBackupVCIError** — To create a backup port, the number of VCI bits on the backup must be equal to the number of VCI bits on the primary.

874 **cveConnIdInUse** — The entered Connection ID is being used by a PVC, SPVC or another SVC Address.

875 **cveLPortDelSvcAddress** — The logical port cannot be deleted as long as an SVC Address is associated with it.

876 **cveLPortDelSvcPrefix** — The logical port cannot be deleted as long as an SVC Prefix is associated with it.

877 **cveDefaultAcctAddrLen** — Bad length for Default Accounting address, must be less than 20.

878 **cveVpiNotSpecified** — Vpi must be specified.

879 **cveVciNotSpecified** — Vci must be specified.

880 **cveLPortDelSvcUserPart** — The logical port cannot be deleted as long as an SVC User Part is associated with it.

881 **cveSvcCugStateInvalid** — The CUG state value is invalid.

882 **cveSvcNotAvailable** — The selected SVC has cleared since this list was uploaded.

883 **cveSvcHasCleared** — The SVC has cleared.

884 **cveSharedThreadLPBwExceeded** — The IOM upon which this logical port is being configured shares a common switching fabric thread with an adjacent slot. Slots 3 and 4, slots 5 and 6, slots 7 and 8, slots 11 and 12, slots 13 and 14, and slots 15 and 16 share common switching fabric threads. The sum of the logical port bandwidth on the two IOMs sharing a common switching fabric thread must not exceed 599.040 Mbps. The bandwidth value entered for this logical port will cause that figure to be exceeded.

885 **cveDs3LoopbackActive** — Cannot activate the loopback test because the port is already in loopback.

886 **cveDs1LoopbackActive** — Cannot activate the loopback test because the port is already in loopback.

887 **cveLPortPvcRangeVPIError** — The specified VCC VPI Stop is not within the VCC VPI/VCI Ranges.

888 **cveInvalidSvcVccVpiVci** — VPI must be greater than or equal to 0 and less than 2 to the power N where N is the number of VPI bits valid for the ATM feeder logical port, and VCI (if required) must be greater than 31 and less than 2 to the power M where M is the number of VCI bits valid for the ATM feeder logical port.

889 **cveInvalidNniSvcVpcVpi** — VPIs for a VPC must be greater than or equal to 1 and less than 4096.

890 **cveInvalidUniSvcVpcVpi** — VPIs for a VPC must be greater than or equal to 1 and less than 256.

891 **cveOverSvcNPfxNodeCapC500** — The number of SVC node prefixes configured on a CBX 500 switch cannot exceed 256.

892 **cveOverSvcNPfxNodeCapBStdx** — The number of SVC node prefixes configured on a B-STDX switch cannot exceed 64.

893 **cveOverSvcAddrObjCap500Iom** — The sum of the number of configured SVC port prefixes, port addresses, and port user parts on a CBX 500 IOM cannot exceed 1024.

894 **cveOverSvcAddrObjCapMsIop** — The sum of the number of configured SVC port prefixes and port addresses on a B-STDX IOP with multiservice capability cannot exceed eight.

895 **cveOverSvcAddrObjCapFrIop** — The sum of the number of configured SVC port prefixes and port addresses on a B-STDX IOP with frame relay capability cannot exceed 512.

896 **cveNoCardCapaChgSvcAddrObjs** — The capability of a card cannot be changed to multiservice when the sum of the number of SVC port prefixes and port addresses configured on the card is greater than eight.

897 **cvePPTfShaperPacingModeInv** — The physical port traffic shaping pacing mode cannot be less than zero or greater than 32.

898 **cvePPortLPBwTooHighForTfShpr** — The effective bandwidth of the physical port cannot be reduced below that which is required by the member logical ports.

899 **cveVUniNotSupportedOnSw** — Virtual UNI logical ports are not supported on this switch type.

900 **cveCktLPortDirectUniVpiError** — The specified VPI of a Direct Uni Endpoint is invalid. It conflicts with an associated Virtual Uni's VPI Start/Stop Range or an associated OPTimum Cell Trunk's VPI.

901 **cveCktLPortVirtualUniVpiError** — The specified VPI of a Virtual Uni Endpoint is not within its VPI Start/Stop Range.

902 **cveLPortCktVUniDirectVpiError** — The specified VPI Start/Stop Range is invalid. The corresponding feeder is used as a circuit endpoint using a VPI within the VPI Start/Stop Range.

903 **cveLPortCktVUniSelfVpiError** — The specified VPI Start/Stop Range is invalid. The Virtual Uni is used as a circuit endpoint using a VPI within the old VPI Start/Stop Range.

904 **cveRevertBackupVPIError** — To revert a backup port, the number of VPI bits on the backup must be equal to the number of VPI bits on the primary.

905 **cveRevertBackupVCIError** — To revert a backup port, the number of VCI bits on the backup must be equal to the number of VCI bits on the primary.

906 **cveSpvcDeleteConfirm** — About to DELETE the selected Soft PVC. Are you sure?

907 **cveSpvcLeafDeleteConfirm** — About to DELETE the selected Soft PVC Leaf. Are you sure?

1848 **cveInvalidDefinedXface** — The defined interface is invalid for this card type.

1849 **cveInvalidClockSpeed** — The clock speed is invalid for this physical port type.

1850 **cveDeleteMaskWithSmdsDteAddr** — Cannot delete the Smds Address Prefix while there is an Smds DTE Address defined on it.

1851 **cveAddrPoolObjIsIA** — The SSI Dte Address is identical to an existing Individual Address.

1988 **cveDb_TRAN_LOG_FULL** — CascadeView database transaction log is full. Please see your database administrator immediately.

1989 **cveDb_NO_SUCH_SWITCH** — Error no such switch.

1990 **cveDb_NO_SUCH_TABLE** — Error no such table.

1991 **cveDb_FETCHSINGLE** — Error reading from database.

1992 **cveDb_UPDATE** — Error updating data.

1993 **cveDb_DELETE** — Error deleting data.

1994 **cveDb_INSERT** — Error inserting data into database.

1995 **cveDb_CLOSE** — Error closing cursor.

1996 **cveDb_FETCH** — Error reading from database.

1997 **cveDb_OPENCURSOR** — Error open cursor.

1998 **cveDb_USE** — Error opening database.

1999 **cveDb_CONNECT** — Error connecting to database.

2001 **cvePPortUIOMaxBwExceeded** — Bandwidth exceeds 8 Megs.

2002 **cvePPortUIOAdja4MgUsed** — Current port cannot be used since the adjacent port has a bandwidth configured at 4Mbps or greater.

2003 **cvePPortUIOMoreThan2Ports** — No more room to configure this port since only two UIO ports can be used when there is one port configured at 6Mbps or greater. There are two ports used in the card and one of the ports has a bandwidth 6Mbps or greater.

2004 **cveBadPramFile** — Error creating PRAM file, check disk space.

2005 **veCfm_CommitCardAdmin** — The change in Admin Status will cause the card to be rebooted. Are you sure?

2006 **cveCfm_CommitCardCapa** — The change in Service Capability will cause the card to be rebooted. Are you sure?

2007 **cveCfm_CommitCardType** — Change in card type will cause the card to be rebooted. Are you sure?

2008 **cvePPortAtmE3UniPeakCellRate** — The Peak Cell Rate must be between 256 and 80000 cells/sec.

2009 **cvePPortAtmUniPeakCellRateNoPlcp** — The Peak Cell Rate must be between 256 and 104268 cells/sec.

2010 **cveSvcNodePrefixNotUnique** — Duplicate SVC Node Prefix object.

2011 **cveSvcNodePrefixNotFound** — The SVC Node Prefix object was not found.

2012 **cveSvcPrefixNotUnique** — Duplicate SVC Prefix object.

2013 **cveSvcPrefixNotFound** — The SVC Prefix object was not found.

2014 **cveSvcAddressNotUnique** — Duplicate SVC Address object.

2015 **cveSvcAddressNotFound** — The SVC Address object was not found.

2016 **cveSvcUserPartNotUnique** — Duplicate SVC User Part object.

2017 **cveSvcUserPartNotFound** — The SVC User Part object was not found.

2018 **cveSvcBaseAddrNotFound** — The corresponding SVC Base Address of the current object was not found.

2019 **cveSvcConfigNotFound** — The SVC Config object was not found.

2020 **cveSvcTranslateFail** — Error translating data.

2021 **cveCktPcrClp0GreaterClp1** — PCR (CLP=0) cannot be greater than PCR (CLP=0+1).

2030 **cveLoopbackNotSupport** — PVC Loopback is not supported on a PVC endpoint that resides on switch running 4.0 or earlier version.

2031 **cveBChan56BitStuffInvalid** — Bit stuffing must be enabled with a sw56 bchannel lport type.

2032 **cveBChanIsdnBitStuffInvalid** — Bit stuffing must be disabled with an Isdn bchannel lport type.

2033 **cvePPortUIONo2MgAdjaUsed** — Current port cannot be configured at 2Mbps greater since the adjacent port is used.

2034 **cvePPortUIOPairBWSum** — Current port cannot be configured at this bandwidth since the sum of this bandwidth and the adjacent's exceeds 4Mbps. Check the bandwidth of the adjacent port.

2035 **cvePPortUIONo4MgPort2Used** — Current port cannot be configured at 4Mbps or greater since two ports already have been used on the card.

2036 **cvePPortLowMaxPCRWhenPLCP** — The Peak Cell Rate values used by one of the configured logical port(s) is higher than the maximum PCR of 96000 when PLCP Option is enabled.

2037 **cvePPortLowBwWhenPLCP** — The total bandwidth of the configured logical port(s) exceeds the bandwidth of 36864Kbps supported by the ATM DS3 port when PLCP Option is enabled.

2038 **cveDLCIOfFeederLPortDup** — The entered DLCI is being used by the feeder logical port in the physical port.

2039 **cveCacInvalidVBRs** — CLR values must be between 1.0e-12 and 1.0e-01.

2040 **cveCacInvalidPScaleFact** — Port Scale Factors must be non-zero.

2041 **cveCacScaleFactorInvalid** — SCR Scale Factor must be non-zero.

2042 **cveCacUpperLimitInvalid** — SCR Upper Limit must be non-zero.

2043 **cveCacBothNotDefined** — Upper Limit, Scale Factor, and Maximum MBS must all be defined.

2044 **cveTrunkNotManaged** — This trunk will not be managed by CascadeView.

2045 **cveLeafAlreadyExists** — A leaf using this logical port already exists.

2046 **cvePriClkSrcInvalid** — Primary Clock Source must be in the range 1..5.

2047 **cveSecClkSrcInvalid** — Secondary Clock Source must be in the range of 1..5.

2048 **cveRevModeInvalid** — Revertive Mode must be in the range of 1..2.

2049 **cveExtClkOutInvalid** — External Clock Out must be in the range of 1..2.

2050 **cveLineBldOutInvalid** — External Clock Out Line Build Out must be in the range of 1..5.

2051 **cveActSysTimingInvalid** — Active System Timing Reference must be in the range of 1..2.

2052 **cveDisabledSlotsForSp4** — Slots 3-6 and 13-16 cannot be defined when using this type of SP.

2053 **cveOuterCardSlotsNotEmpty** — Slots 3-6 and 13-16 must be empty when using this type of SP.

2054 **cvePPortUIONo6MgPort2Used** — Current port cannot be configured at 6Mbps or greater since two ports already have been used on the card.

2055 **cveTrunkNameInUseInOVDb** — The specified trunk name is already in used in the HP OpenView database.

2056 **cveIsdnAaMatchBChan** — ISDN B-Channel E.164 address should be different from any Authentication address.

2057 **cveDeleteFarEndLBLport** — Cannot delete a logical port while the logical port is in loopback state.

2058 **cveLPortDChan1MustBeFirst** — The ISDN B-Channel logical port cannot be added because there is no ISDN D-Channel logical port defined on physical port one.

2059 **cveTrunkLPort1InUse** — The logical port 1 is already in use.

2060 **cveTrunkLPort2InUse** — The logical port 2 is already in use.

2061 **cveSnmpNoSuchName** — The variable does not exist or is read only.

2062 **cveSnmpBadValue** — The value given has incorrect type or length.

2063 **cveSwitchRedundant** — There is no redundant unit.

2064 **cveFrOnlyCard** — Cannot change capability to multiple services on a frame relay only card.

2065 **cveCktBcMultCir** — When CIR is non-zero, BC must be non-zero and a multiple of CIR.

2066 **cveInvalidNodeIPAddr** — Invalid IP address for a node.

2067 **cveInvalidIPAddr** — Invalid IP address.

2068 **cveAddMaxMcastMembers** — Maximum 100 multicast members can be added when create a Multicast DLCI. If you want to add more than 100 members, add 100 members this time and use Modify Multicast DLCI to add the rest of members.

2069 **cveModMaxMcastMembers** — You can only modify (add or delete) 100 multicast members at a time.

2070 **cveDelMaxMcastMembers** — The selected multicast DLCI has more than 100 members. Please use Modify Multicast DLCI to reduce the number of members first.

2071 **cveSwMaxCktsExceeded** — The maximum number of circuit endpoints (including multicast and management DLCIs) cannot exceed 938 for a switch.

2072 **cvePPortUIONo4MgAdjaUsed** — Current port cannot be configured at 4Mbps or greater since the adjacent port is used.

2073 **cveTrunkOutOfRangeKAErrorThreshold** — Keep alive error threshold is out of range. Valid range is 3 - 255.

2074 **cveLPortCongCtrlInvalid** — Congestion control values must be in the range of (mild < severe < absolute).

2075 **cveLPortCongCtrlMaxExceeded** — The Absolute Threshold value exceeds the maximum value allowable for this card.

2076 **cveCktDeltaBcOutOfRange** — Forward delta Bc is out of range. Valid range is 0 - 65528.

2077 **cveCktDeltaRBcOutOfRange** — Reverse delta Bc is out of range. Valid range is 0 - 65528.

2078 **cveCktDeltaBeOutOfRange** — Forward delta Be is out of range. Valid range is 0 - 65528.

2079 **cveCktDeltaRBeOutOfRange** — Reverse delta Be is out of range. Valid range is 0 - 65528.

2080 **cveCktRedFramePercentOutOfRange** — Forward red frame percent is out of range. Valid percentage is 0 - 100.

2081 **cveCktRRedFramePercentOutOfRange** — Reverse red frame percent is out of range. Valid percentage is 0 - 100.

2082 **cveCktRedFramePercentShouldBeZero** — Forward red frame percent should be zero since forward Be and Bc are also zero.

2083 **cveCktRRedFramePercentShouldBeZero** — Reverse red frame percent should be zero since forward Be and Bc are also zero.

2084 **cveInvalidIpMask** — The specified IP mask is invalid.

2085 **cveSubNetIdInUse** — The specified subnet ID is already in use on this map.

2086 **cveSubNetDelClusterDef** — The subnet cannot be deleted because a cluster is defined on it.

2087 **cveSubNetDelSwitchDef** — The subnet cannot be deleted because a switch is defined on it.

2088 **cveInvalidClusterId** — Cluster ID must be between 0 and 7.

2089 **cveClusterIdInUse** — The specified cluster ID is already in use on this subnet.

2090 **cveClusterNameInUse** — The specified cluster name is not unique.

2091 **cveClusterDelSwitchDef** — The cluster cannot be deleted because a switch is defined on it.

2092 **cveMustBeGSwOnNonCluster** — The switch must be a gateway switch if the subnet is a non cluster subnet.

2093 **cveOnly500CanBeGSw** — Only CBX-500 switch can be a gateway switch on a cluster subnet.

2094 **cveMaxGSwExceeded** — The maximum number of gateway switches cannot exceed 2 for a cluster.

2095 **cveInvalidIpAddrInSubNet** — The IP address is invalid within the selected subnet.

2096 **cveInvalidIpAddrInCluster** — The IP address is invalid within the selected cluster.

2097 **cveMaxSwInClusterExceeded** — The maximum number of switches cannot exceed 30 for a cluster.

2098 **cveMaxSwInSubNetExceeded** — The maximum number of switches cannot exceed 255 for a subnet.

2099 **cveCktFromBtoC** — Two endpoints are on different subnets and one endpoint switch does not support class B IP address.

2100 **cveNonGatewayTrk** — Non gateway switch cannot connect to a switch in different cluster/subnet.

2101 **cveSmdsCountryCodeUsed** — Cannot remove Country Code because it is in use.

2102 **cveSmdsCountryCodeRangeError** — Code value must be between 0 and 9999 inclusive.

2103 **cveSmdsMaskStartPosRangeError** — Mask start position value must be between 1 and 15 inclusive.

2104 **cveSmdsMaskSizeRangeError** — Mask size value must be between 0 and (16 - Mask start position).

2105 **cveCktZeroCirFactorError** — Zero Cir Be/Routing Factors must be between 0 and 100 inclusive.

2106 **cveInvalidSubNetId** — SubNet ID must be between 1 and 255.

2107 **cveInvalidTrkFailureThreshld** — Trunk failure threshold cannot be less then 2 sec.

2108 **cveInvalidTrkRestThreshld** — Trunk restoration threshold cannot be less than 2 sec.

2109 **cveNumBuTrkLgrThn8** — Number of backup trunk excess 8 for this primary trunk.

2110 **cveNoPriForBackup** — Must select an existing primary trunk for this backup trunk.

2111 **cveBackupTrkExisted** — Cannot delete this primary trunk. It is referenced by some backup trunks.

2112 **cveNegPriTrkAttr** — All the Primary Trunk related attributes must have non-negative values.

2113 **cveZeroCirFactorChange** — Cannot modify Zero CIR Factors when this LPort is used in a circuit.

2114 **cveBadPvcFactorError** — Bad Pvc Factor value must be between 0 and 32 inclusive.

2115 **cveAmberReductionPmError** — Amber Reduction Pm value must be between 0 and 100 inclusive.

2116 **cveAmberReductionPsError** — Amber Reduction Ps value must be between 0 and 100 inclusive.

2117 **cveCgCheckIntervalError** — Congestion Check Interval value must be between 0 and 60 inclusive.

2118 **cveCgClearDelayError** — Congestion Clear Delay value must be between 0 and 60 inclusive.

2119 **cveCustomerNameInUse** — Customer name must be unique.

2120 **cveCustomerIdInUse** — Customer Identifier must be a unique number in the range of 1..65535.

2121 **cveVPNNameInUse** — Virtual private network name must be unique.

2122 **cveVPNNumberInUse** — Virtual private network number must be unique.

2123 **cveDelCustomerInUseByLPort** — Cannot delete customer. It is still referenced by a logical port.

2124 **cveDelCustomerInUseByCkt** — Cannot delete customer. It is still referenced by a circuit.

2125 **cveDelVPNInUseByTrunk** — Cannot delete virtual private network. It is still referenced by a trunk.

2126 **cveDelVPNInUseByLPort** — Cannot delete virtual private network. It is still referenced by a logical port.

2127 **cveDelVPNInUseByCkt** — Cannot delete virtual private network. It is still referenced by a circuit.

2128 **cveTrunkEndpointDiffVPN** — The two endpoints of the trunk have different virtual private networks.

2129 **cveCircuitEndpointDiffVPN** — The two endpoints of the circuit have different virtual private networks.

2130 **cveInvalidSonetIntervalInput** — Invalid input for Sonet MIB interval statistics.

2131 **cveCfm_CommitSP4SP8CardTypeChange** — The change in Card Type will cause the card to be rebooted. Are you sure?

2132 **cveCACModify_Warning** — Changing the CAC could have an impact on bandwidth allocation if circuits are present on the switch. Are you sure?

2133 **cveSWSyncFileFailedCRC** — Uploaded PRAM file contains corrupted data. Please try again.

2134 **cveSWSyncFileProcessError** — Error encountered while processing uploaded PRAM file. Please try again.

2135 **cveInvalidSubNet** — Invalid Subnet.

2136 **cveInvalidCluster** — Invalid Cluster.

2137 **cveAtmDirectTrkDefined** — An ATM Direct Trunk is already defined. Cannot define any other logical port type.

2138 **cveAtmVPILengthOutOfRange** — VPI length is out of range.

2139 **cveCktInvalidBwPrio** — Invalid circuit bandwidth priority value (0 - 15).

2140 **cveCktInvalidBumpPrio** — Invalid circuit bumping priority value (0 - 7).

2141 **cveNodeAuthDomainMismatch** — Authentication Domain Name invalid.

2142 **cvePPortShaperNegativeScr** — Negative Sustainable Cell Rate.

2143 **cvePPortShaperNegativePcr** — Negative Peak Cell Rate.

2144 **cveAtmCsLPortBandwidthExceedsLPortScr** — LPort's bandwidth exceeds LPort's sustainable cell rate. LPort's sustainable cell rate comes from the traffic shaper selection.

2145 **cveSmdsTotalAddrLenExcess** — The maximum number of digits for a Smds address is 15.

2146 **cveSmdsTotalAddrLenBelow** — Total number of digits is too few to represent a valid SMDS address.

2147 **cveFeederAddrAlreadyDefined** — Cannot have more than one feeder address defined per SSI lport.

2148 **cvePriClkEqSecClk** — The primary clock and the secondary clock cannot be the same.

2149 **cveExtClkInfInvalid** — External Clock Interface Type must be in the range of 1..2.

2150 **cveDtePollTimerInv2** — Invalid DTE Poll Interval value (5 - 255).

2151 **cveDceVerifTimerInv2** — Invalid DCE Poll Verify Timer value (5 - 255).

2152 **cveCardIomChanged** — If the IOM is present in the switch, it must not be \"out-of-sync\" prior to deleting the card from the NMS.

2153 **cveNmsPathAddrZero** — Zero entry for management VPI/VCI IP address in NMS path.

2154 **cveMaxCktsExceededAtm** — The maximum number of circuits (including multicast and management connections) cannot exceed 3000 for a card.

2155 **cveTrunkNameMissing** — Trunk name is missing.

2156 **cveT1E1FifteenMinThreshEslInvalid** — Fifteen Minute ES-L threshold value must be within the range of 1 to 900.

2157 **cveT1E1FifteenMinThreshCvpInvalid** — Fifteen Minute CV-P threshold value must be within the range of 1 to 16383.

2158 **cveT1E1FifteenMinThreshEspInvalid** — Fifteen Minute ES-P threshold value must be within the range of 1 to 900.

2159 **cveT1E1FifteenMinThreshSespInvalid** — Fifteen Minute SES-P threshold value must be within the range of 1 to 63.

2160 **cveT1E1FifteenMinThreshSaspInvalid** — Fifteen Minute SAS-P threshold value must be within the range of 1 to 63.

2161 **cveT1E1FifteenMinThreshCsspInvalid** — Fifteen Minute CSS-P threshold value must be within the range of 1 to 63.

2163 **cveT1E1OneDayThreshEslInvalid** — One Day ES-L threshold value must be within the range of 1 to 65535.

2164 **cveT1E1OneDayThreshCvpInvalid** — One Day CV-P threshold value must be within the range of 1 to 1048575.

2165 **cveT1E1OneDayThreshEspInvalid** — One Day ES-P threshold value must be within the range of 1 to 65535.

2166 **cveT1E1OneDayThreshSespInvalid** — One Day SES-P threshold value must be within the range of 1 to 4095.

2167 **cveT1E1OneDayThreshSaspInvalid** — One Day SAS-P threshold value must be within the range of 1 to 4095.

2168 **cveT1E1OneDayThreshCsspInvalid** — One Day CSS-P threshold value must be within the range of 1 to 4095.

2169 **cveT1E1OneDayThreshUaspInvalid** — One Day UAS-P threshold value must be within the range of 1 to 4095.

2170 **cvePPerformanceThresholdsObjectNotFound** — The performance thresholds object could not be found in the database.

2171 **cvePPortInvalidCircuitType** — Invalid circuit type for this port type.

2172 **cvePPortInvalidInBandLineLoopbackCode** — Invalid In Band Line Loopback Code for this port type.

2173 **cvePFdlObjNotFound** — The physical port fdl object could not be found in the database.

2174 **cvePPortNoOc12CbitParity** — C-Bit parity is not allowed on OC-12 cards.

2175 **cveOc12FifteenMinThreshCvsInvalid** — Fifteen Minute CV-S threshold value must be within the range of 1 to 16383.

2176 **cveOc12FifteenMinThreshEssInvalid** — Fifteen Minute ES-S threshold value must be within the range of 1 to 900.

2177 **cveOc12FifteenMinThreshSessInvalid** — Fifteen Minute SES-S threshold value must be within the range of 1 to 63.

2178 **cveOc12FifteenMinThreshCvlInvalid** — Fifteen Minute CV-L threshold value must be within the range of 1 to 16383.

2179 **cveOc12FifteenMinThreshEslInvalid** — Fifteen Minute ES-L threshold value must be within the range of 1 to 900.

2180 **cveOc12FifteenMinThreshSeslInvalid** — Fifteen Minute SES-L threshold value must be within the range of 1 to 63.

2181 **cveOc12FifteenMinThreshUaslInvalid** — Fifteen Minute UAS-L threshold value must be within the range of 1 to 63.

2182 **cveOc12FifteenMinThreshCvpInvalid** — Fifteen Minute CV-P threshold value must be within the range of 1 to 16383.

2183 **cveOc12FifteenMinThreshEspInvalid** — Fifteen Minute ES-P threshold value must be within the range of 1 to 900.

2184 **cveOc12FifteenMinThreshSespInvalid** — Fifteen Minute SES-P threshold value must be within the range of 1 to 63.

2185 **cveOc12FifteenMinThreshUaspInvalid** — Fifteen Minute UAS-P threshold value must be within the range of 1 to 63.

2186 **cveOc12OneDayThreshCvsInvalid** — One Day CV-S threshold value must be within the range of 1 to 1048575.

2187 **cveOc12OneDayThreshEssInvalid** — One Day ES-S threshold value must be within the range of 1 to 65535.

2188 **cveOc12OneDayThreshSessInvalid** — One Day SES-S threshold value must be within the range of 1 to 4095.

2189 **cveOc12OneDayThreshCvlInvalid** — One Day CV-L threshold value must be within the range of 1 to 1048575.

2190 **cveOc12OneDayThreshEslInvalid** — One Day ES-L threshold value must be within the range of 1 to 65535.

2191 **cveOc12OneDayThreshSeslInvalid** — One Day SES-L threshold value must be within the range of 1 to 4095.

2192 **cveOc12OneDayThreshUaslInvalid** — One Day UAS-L threshold value must be within the range of 1 to 4095.

2193 **cveOc12OneDayThreshCvpInvalid** — One Day CV-P threshold value must be within the range of 1 to 1048575.

2194 **cveOc12OneDayThreshEspInvalid** — One Day ES-P threshold value must be within the range of 1 to 65535.

2195 **cveOc12OneDayThreshSespInvalid** — One Day SES-P threshold value must be within the range of 1 to 4095.

2196 **cveOc12OneDayThreshUaspInvalid** — One Day UAS-P threshold value must be within the range of 1 to 4095.

2197 **cveSESThresholdSettingModify_Warning** — Changing the SES Threshold Setting will affect how the SES counts are calculated and it will invalidate interval and total data collected to this point. Are you sure?

2198 **cveLPortInvalidOc12TrkCdv** — Invalid OC-12 logical port direct trunk cell delay variation.

2199 **cveTrafficShaperPriorityOutOfOrder** — Shaper priority must be ordered from highest (0) to lowest (15) according to the shaper ID.

2200 **cveShaperDivisorLessThanDividend** — The traffic shaper dividend is greater than the divisor.

2201 **cveLPortMildThrhldEqZero** — Congestion Control: Mild Threshold must be greater than zero.

2202 **cveNoValidFile** — No valid file is selected.

2203 **cveBigDlciExist** — There is a circuit defined on this logical port with DLCI > 991. The Link Mgmt Protocol must be LMI Rev 1.

2204 **cveInvalidSwImage** — The selected file is not a valid switch software image file.

2205 **cveSmdsAddrNameInvalid** — The Address Name cannot contain the character ‘.

2206 **cveAssignIdOfSwitchWithSameName** — Switch with the same name existed. IP Address has been assigned to that of the existing switch. Verification has completed.

2207 **cveAssignNextIdCauseSwitchWithSameId** — Switch with the same IP Address existed. Next available IP Address is now assigned. Verification has completed.

2208 **cveSnmpNoSuchNamePossiblyDown** — There is no such variable name in this MIB - possibly the card is down or not present.

2209 **cveLPortZeroCodingNx56** — The zero coding is set to ‘N x 56’ on the channel; bit stuffing must be on.

2210 **cveCirBcBeAllZero** — CIR, Bc, and Be cannot all be zero.

2211 **cveCir1Bc0Be1** — It is not valid for Bc to be zero while CIR and Be are both not zero.

2212 **cveCir0Bc1** — It is not valid for CIR to be zero while Bc is not zero.

2213 **cvePathSameAsSwitchIp** — The Nms Path Ip address cannot be the same as the switch’s Ip address.

2214 **cveSmdsAddrLenError** — The total length of Country Code, Address Prefix and Address cannot be less than 11 digits or more than 15 digits.

2215 **cveInvalidSwitchNameApostrophe** — Invalid Switch Name: Cannot have \”\’\” in Switch Name.

2216 **cveSmdsAddrLportError** — Cannot add the second Management Address to the same lport.

2217 **cveSmdsNoAddr** — No input for Management Address.

2218 **cveCdvBTmaxCdv** — CDV value cannot be more than 5000 (microsec).

2219 **cveCardTypeChgToNonEmpty** — Changing directly from one card type to another is not allowed. The card type must first be set to \"Empty Card\".

2220 **cvInvalidVpiRange** — The VPI entered is out of range.

2221 **cvInvalidVciRange** — The VCI entered is out of range.

2222 **cveIsdnDChannelDeselect** — Cannot deselect the ISDN D-Channel from this physical port.

2223 **cveIsdnDChannelNotSelected24** — Channel 24 is not selected as the designated ISDN D-Channel.

2224 **cveIsdnDChannelNotSelected16** — Channel 16 is not selected as the designated ISDN D-Channel.

2225 **cveMaxBufferSizeRangeExceeded** — Maximum buffer size value must be between 2048 and 8192 inclusive.

2226 **cveNoCustForTrunk** — Trunk does not contain any selected customer name.

2227 **cveLPortTooManyVpiVciBitsCsIwu** — The total number of valid VPI and VCI bits must be 11 or less.

2228 **cveAuthenDomainSelected** — This Authentication Domain is selected for use by this switch and cannot be deleted. To delete, you must first de-select it at the switch.

2229 **cveRipOnStateIpAddrMissing** — Cannot Set RIP State on. You must first enter an Ethernet IP Address to indicate that this switch is a gateway switch.

2230 **cveRipSendHostRoutesOnStateIpAddrMissing** — Cannot Set Send Host Routes on. You must first enter an Ethernet IP Address to indicate that this switch is a gateway switch.

2231 **cveSmdsCountryCodeExists** — Cannot add Country Code because one already exists.

2232 **cveVpnNameMissing** — Virtual Private Network name is missing.

2233 **cveLPortInvalidIlmiVci0_31_not_16** — The ILMI VCI must be either be 16 or greater than or equal to 32.

2234 **cvChangeServiceAffecting** — Changing the Keep alive error threshold value is service affecting. Do you want to continue ?

2235 **cveRedFramePercentNeedChange** — The Red Frame Percentage is reset to 100.

2236 **cveCktBcEqCir** — When jump is selected as rate enforcement scheme, BC must be equal to CIR.

2237 **cveMgmtVpiVciDefinedOnLPort** — There is a Management VPI/VCI already defined on the logical port.

2238 **cveCktRootFwdRevTfDNull** — Zero-valued root traffic descriptors are not allowed.

2239 **cveCardIopChanged** — If the IOP is present in the switch, it must not be \"out-of-sync\" prior to deleting the card from the NMS.

2240 **cveCascadeViewAttrs** — Please set CascadeView Object Attributes.

2241 **cveMoveLoopbackCkt** — This circuit is set to loopback, cannot be removed.

2242 **cveT1E1FifteenMinThreshUaspInvalid** — Fifteen Minute UAS-P threshold value must be within the range of 1 to 63.

2247 **cveCustomerNameMissing** — The Customer Name is missing.

2248 **cveInvalidVpnName** — Public is not a valid vpn name.

2249 **cveInvalidCustomerName** — Public is not a valid customer name.

2250 **cveCcrmProtocolIDInvalid** — CCRM Protocol ID value must be within the range of 0 to 255.

2251 **cveBcmProtocolIDInvalid** — BCM Protocol ID value must be within the range of 0 to 255.

2252 **cveRmCellXmitIntervalInvalid** — RM Cell Xmit Interval value must be within the range of 30 to 250 millisec.

2253 **cveIdleVCFactorInvalid** — Idle VC Factor value must be within the range of 1 to 8.

2254 **cveIcrConstantInvalid** — ICR Constant value must be within the range of 0 to 15.

2255 **cveMulticastDiscardInvalid** — Multicast Discard Threshold value must be within the range of 0 to 64K.

2256 **cveModCcrmProtocolIDNeedPramSync** — The CCRM Protocol ID cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization.

2257 **cveModBcmProtocolIDNeedPramSync** — The BCM Protocol ID cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization.

2258 **cveModIdleVCFactorNeedPramSync** — The Idle VC Factor cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization.

2259 **cveModMulticastDiscNeedPramSync** — The Multicast Discard Threshold cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization.

2260 **cveModIcrConstantNeedPramSync** — The ICR Constant cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization.

2261 **cveModClp01NeedPramSync** — The CLP0+1 Threshold cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization.

2262 **cveModDiscardNeedPramSync** — The Discard Threshold cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization.

2263 **cveModEfciNeedPramSync** — The Congestion Threshold cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization.

2264 **cveModPortBuffersNeedPramSync** — Port Buffers cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization.

2265 **cveProtocolIDsNotUnique** — The CCRM and BCM Protocol IDs must not be equal.

2266 **cveCktMcrInvalid** — MCR must be non-zero.

2267 **cveCktDiscardSelectInvalid** — Discard Select is invalid.

2268 **cveNrtsCardDisableDlciDefined** — The ATM Flow Control Processor cannot be disabled because a circuit and/or management connection is defined on it.

2269 **cveNrtsManageVBRnrtDlciDefined** — Manage VBRnrt Traffic cannot be disabled because a circuit and/or management connection is defined on it.

2270 **cveTotalLPortBuffersExceeded** — The specified Port Buffers is invalid because it would exceed the Total ATM FCP Cell Buffers limitation on the card.

2271 **cveDiscardValueInvalid** — The range of values for the Discard Thresholds is 1 - 255.

2272 **cveEfciValueInvalid** — Congestion Threshold values must be less than their corresponding Discard Threshold values. Congestion(i) must be less than Discard(i).

2273 **cveRifValueInvalid** — The range of values for the Rate Increase Factor Thresholds is 1 - 15.

2274 **cveRdfValueInvalid** — The range of values for the Rate Decrease Factor Thresholds is 1 - 15.

2275 **cveCardLoadProfileObjectsNotFound** — The Rate Profile Threshold objects were not found.

2276 **cveInvalidNoRdfProfileEntries** — Invalid number of entries in the Rate Decrease Table.

2277 **cveInvalidNoRifProfileEntries** — Invalid number of entries in the Rate Increase Table.

2278 **cveInvalidNoEfciProfileEntries** — Invalid number of entries in the Rate Congestion Table.

2279 **cveInvalidNoDiscardProfileEntries** — Invalid number of entries in the Rate Discard Table.

2280 **cveInvalidLPortClp01Threshold** — The Clp0+1 value must be less than the Port Buffers configured for this logical port.

2281 **cveInvalidLPortDiscardThreshold** — The Discard value must be less than the CLP0+1 value configured for this logical port.

2282 **cveInvalidLPortEfciThreshold** — The Congestion value must be less than the Discard value configured for this logical port.

2283 **cveDs3FifteenMinThreshCvlInvalid** — Fifteen Minute CV-L threshold value must be within the range of 1 to 16383.

2284 **cveDs3FifteenMinThreshEslInvalid** — Fifteen Minute ES-L threshold value must be within the range of 1 to 900.

2285 **cveDs3FifteenMinThreshSeslInvalid** — Fifteen Minute SES-L threshold value must be within the range of 1 to 63.

2286 **cveDs3FifteenMinThreshCvpInvalid** — Fifteen Minute CV-P threshold value must be within the range of 1 to 16383.

2287 **cveDs3FifteenMinThreshEspInvalid** — Fifteen Minute ES-P threshold value must be within the range of 1 to 900.

2288 **cveDs3FifteenMinThreshSespInvalid** — Fifteen Minute SES-P threshold value must be within the range of 1 to 63.

2289 **cveDs3FifteenMinThreshSaspInvalid** — Fifteen Minute SAS-P threshold value must be within the range of 1 to 63.

2290 **cveDs3FifteenMinThreshUaspInvalid** — Fifteen Minute UAS-P threshold value must be within the range of 1 to 63.

2291 **cveDs3FifteenMinThreshCvsInvalid** — Fifteen Minute CV-S threshold value must be within the range of 1 to 16383.

2292 **cveDs3FifteenMinThreshEssInvalid** — Fifteen Minute ES-S threshold value must be within the range of 1 to 900.

2293 **cveDs3FifteenMinThreshSessInvalid** — Fifteen Minute SES-S threshold value must be within the range of 1 to 63.

2294 **cveDs3FifteenMinThreshSassInvalid** — Fifteen Minute SAS-S threshold value must be within the range of 1 to 63.

2295 **cveDs3FifteenMinThreshUassInvalid** — Fifteen Minute UAS-S threshold value must be within the range of 1 to 63.

2296 **cveDs3FifteenMinThreshCssInvalid** — Fifteen Minute CS-S threshold value must be within the range of 1 to 255.

2297 **cveDs3OneDayThreshCvlInvalid** — One Day CV-L threshold value must be within the range of 1 to 1048575.

2298 **cveDs3OneDayThreshEslInvalid** — One Day ES-L threshold value must be within the range of 1 to 65535.

2299 **cveDs3OneDayThreshSeslInvalid** — One Day SES-L threshold value must be within the range of 1 to 4095.

2300 **cveCacInvalidAlpha** — CDV Alpha values must be between 1.0e-12 and 1.0e-01.

2301 **cveCacMaximumMBSInvalid** — Maximum Burst Size must be non-zero.

3000 **cveDs3OneDayThreshCvpInvalid** — One Day CV-P threshold value must be within the range of 1 to 1048575.

3001 **cveDs3OneDayThreshEspInvalid** — One Day ES-P threshold value must be within the range of 1 to 65535.

3002 **cveDs3OneDayThreshSespInvalid** — One Day SES-P threshold value must be within the range of 1 to 4095.

3003 **cveDs3OneDayThreshSaspInvalid** — One Day SAS-P threshold value must be within the range of 1 to 4095.

3004 **cveDs3OneDayThreshUaspInvalid** — One Day UAS-P threshold value must be within the range of 1 to 4095.

3005 **cveDs3OneDayThreshCvsInvalid** — One Day CV-S threshold value must be within the range of 1 to 1048575.

3006 **cveDs3OneDayThreshEssInvalid** — One Day ES-S threshold value must be within the range of 1 to 65535.

3007 **cveDs3OneDayThreshSessInvalid** — One Day SES-S threshold value must be within the range of 1 to 4095.

3008 **cveDs3OneDayThreshSassInvalid** — One Day SAS-S threshold value must be within the range of 1 to 4095.

3009 **cveDs3OneDayThreshUassInvalid** — One Day UAS-S threshold value must be within the range of 1 to 4095.

3010 **cveDs3OneDayThreshCssInvalid** — One Day CS-S threshold value must be within the range of 1 to 65535.

3011 **cveInvalidNoFiles** — There are no files specified to load.

3012 **cveErrorOpeningInputFile** — Error opening input file.

3013 **cveInvalidProfileToken** — Default file not specified.

3014 **cveModRateProfileNeedPramSync** — The Rate Profile Tables cannot be changed dynamically in the switch. They can only be maintained through PRAM synchronization.

3015 **cveInvalidRedundancyCombination** — APS 1+1 Redundancy is not available for this card type.

3016 **cveLPortsExistOnProtectionPort** — APS 1+1 Redundancy cannot be enabled because the protection port has logical ports defined on it.

3017 **cveApsObjNotFound** — The physical port aps object could not be found in the database.

3018 **cveCongNotifyTimeSpecify** — Congestion Notify Time must be specified.

3019 **cveCongNotifyTimeRange** — Congestion Notify Time must be within the range of 1..86400.

3020 **cveSmdsAreaDupOnMultiSwitch** — The change is invalid because it results in duplicated SMDS area.

3021 **cveModFcpStatusNeedPramSync** — The ATM Flow Control Processor cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization.

3022 **cveModNrtsManageVBRnrtNeedPramSync** — Manage VBRnrt Traffic cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization.

3023 **cvePPortInvalidFeacLoopback** — Invalid FEAC Loopback setting for this port type.

3024 **cveAlarmFailureOutOfRange** — The Alarm Failure setting is out of range. Valid range is 0 - 65535.

3025 **cveAlarmClearOutOfRange** — The Alarm Clear setting is out of range. Valid range is 0 - 65535.

3026 **cveSpvcRetryLimitInvalid** — The Retry Limit value must be within the range of 0 to 65535.

3027 **cveSpvcRetryIntervalInvalid** — The Retry Interval value must be within the range of 0 to 3600.

3028 **cveModMulticastRateNeedPramSync** — The Multicast Rate cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization.

3029 **cveInvalidNrtsParent** — The Atm FCP cannot be enabled on this card.

3030 **cveApsNotEnabled** — APS must be Enabled on the PPort screen before changing these parameters.

3031 **cveCfm_CommitISDNswtype** — The change in ISDN CO Switch Type will cause the card to be rebooted. Are you sure?

3032 **cveCardTypeMismatch** — An attempt to configure a card type that conflicts with the actual card type is not allowed. This can potentially put the CP/SP out of synchronization.

3032 **cveLPortNotSelected** — A logical port must be selected in order to do this operation.

3033 **cveIwuCsLPortTooManyVpiVciBits** — The total number of VPI and VCI bits must not exceed 11.

3034 **cveIwuCsLPortTooFewVciBits** — The number of VCI bits has to be a minimum of 6 and cannot exceed 11.

3035 **cveIwuCsLPortTooManyVciBits** — The number of VCI bits has to be a minimum of 6 and cannot exceed 11.

3036 **cveIwuCsLPortTooManyVpiBits** — The number of VPI bits can vary from 0 through 5.

3037 **cveIwuCsLPortTooFewVpiBits** — The number of VPI bits can vary from 0 through 5.

3038 **cveModDisableFcpStatusNeedPramSync** — Disabling the ATM Flow Control Processor will cause the FCP LPort parameters to be reset for all LPorts defined on the card. In addition, the ATM Flow Control Processor cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization. Do you wish to continue ?

3039 **cveAcctServerMoveConfirm** — This switch will be unassigned from its current accounting server prior to being assigned to the selected accounting server. Do you wish to continue?

3040 **cveLANIdleTimeoutOutOfRange** — The LAN Idle Timeout value must be between 0 - 900.

3041 **cveLPortNotSelected** — A logical port must be selected in order to do this operation.

3042 **cveSvcFrameDiscardInvalid** — The Frame Discard value is invalid.

3043 **cveLPortInvalidCtrlUpc** — The Control UPC Function option is invalid.

3044 **cveCfm_CommitSwitchReboot** — Performing this action will cause the switch to be rebooted. Are you sure?

3045 **cveInvalidCllmInterval** — Invalid CLLM Interval value. Valid range is 5-30.

3046 **cveBadCllmThrhldNone** — Invalid CLLM ThresholdNone value. Valid range is 1-100.

3047 **cveBadCllmThrhldMild** — Invalid CLLM ThresholdMild value. Valid range is 1-100 and greater than CLLM ThresholdNone value.

3048 **cveSwitchObjNameIsNull** — Switch Object Name is NULL in MaintenanceDlgData.

3049 **cveCvListReadFailed** — CvList Read is Failed Due to NameSList is NULL.

3050 **cveIfNoAllocFailure** — Cannot allocate IfIndex for the lport, contention with another process. Wait for 30 seconds and try again.

NavisXtend Provisioning Server Errors

4100 **cveBadArgumentId** — Invalid Argument Id in request to server.

4101 **cveBadArgumentType** — Invalid argument type in request to server.

4102 **cveBadArgumentValue** — Invalid argument value in request to server.

4103 **cveArgumentNotPresent** — Required argument is missing in request to server.

4104 **cveObjectDoesNotExist** — Requested object request does not exist.

4105 **cveUnsupportedCommand** — Unsupported command for this object type or attempt to write create-only or read-only attributes.

4106 **cveUnableToSend** — Unable to establish communications with server.

4107 **cveListDone** — End of list reached.

4108 **cveNotAuthorized** — Client is not authorized to perform this request.

4109 **cveRequestTimedOut** — Request timed out.

4110 **cveEndpoint2Needed** — The second endpoint must be specified as a DLCI in parameter CVA_CircuitEndpoint2.

4111 **cveEndpoint2Invalid** — The object specified as Endpoint2 does not exist.

4112 **cveListInvalidType** — An invalid type of object was requested in the List command.

4113 **cveTooManyLocks** — The server has too many locks allocated.

4114 **cveNoChildren** — The requested type does not have any listable children.

4115 **cveParentDoesNotExist** — The object's parent record does not exist.

4116 **cveAddressNotInList** — The specified address is not in the list.

4117 **cveAddressAlreadyInList** — The specified address is already in the list.

4118 **cveBadNGAMemberType** — Only Local Individual Addresses may be added to or deleted from a Netwide Group Address.

4120 **cveBadIScreenMemberType** — Only Local Individual Addresses or Alien Individual Addresses may be added to or deleted from an Individual Screen.

4121 **cveBadGScreenMemberType** — Only Switch Group Addresses or Alien Group Addresses may be added to or deleted from a Group Screen.

4122 **cveMemberDoesNotExist** — The members's record does not exist.

4123 **cveMemberNotSpecified** — The member was not specified, or was specified incorrectly.

4124 **cveNotAMember** — The requested member is not in the Group Address or screen.

4125 **cveAlreadyAMember** — The requested member is already in the Group Address or screen.

4026 **cveMemberNotOnSwitch** — The requested member must be on the same switch as the Screen.

4127 **cveBadScreenMemberType** — The type of the specified member is not valid for a screen.

4128 **cveCard1NotCp** — Card 1 must be a CP.

4129 **cveCard2NotCpOrEmpty** — Card 2 must be a CP or be empty.

4130 **cveInvalidCpCard** — CP cards may be placed only in slots 1 and 2.

4131 **cveInvalidEndpoint** — Circuit endpoint is of invalid type.

4132 **cveInvalidAccessToArg** — Attempt to write a read-only or create-only argument.

4134 **cveArgDoesNotApply** — Argument does not apply to this subtype or configuration.

4135 **cveEndpoint1Mismatch** — The type specified for Endpoint 1 doesn't match the LPort type.

4136 **cveEndpoint2Mismatch** — The type specified for Endpoint 2 doesn't match the LPort type.

4137 **cveAtmQosInvalid** — ATM QOS parameters (Cps, Mbs, Scr) are invalid.

4138 **cveCircuitSamePPort** — The circuit's endpoints must be on different physical ports.

4140 **cvePsrvSnmpErr** — Start of SNMP error codes.

4141 **cveSnmpTooBig** — SNMP PDU was too big.

4142 **cveSnmpNoSuchName** — A variable does not exist or is read only.

4143 **cveSnmpBadValue** — A value given has incorrect type, value, or length.

4144 **cveSnmpReadOnly** — Write access was requested to a read-only variable.

4145 **cveSnmpGenericError** — An unspecified error occurred in the SNMP request.

4150 **cveAddrNotInAddrPool** — The address is not in the pool of the SSI DTE LPort.

4151 **cveNoRoomForListCmd** — Server is processing too many ListContained commands; try later.

4152 **cveContextNotFound** — The Context for the Get Next command is invalid. (Timed out?)

4153 **cveIaInUseByDte** — The IA is in use by an SSI DTE Lport.

4154 **cveAddrInScreen** — Cannot delete the address while it is in use by a screen.

4155 **cvePrefixNotOnSwitch** — The address prefix is not defined on the same switch as the Logical Port.

4156 **cveFeederAlreadyDefined** — A feeder address is already defined on the SMDS SSI DTE LPort.

4157 **cveScreenMemberIsLocal** — Cannot add a local address to the screen of its own LPort.

4158 **cveAfiHexNotSpecified** — Afi or Hex string not specified.

4159 **cveInvalidParentType** — The listcontained is not supported for this particular parent type for PMPCkt.

4160 **cveInvalidSpCard** — SP cards may be placed only in slots 1 and 2 -- for 500 switch.

4161 **cveCard1NotSp** — Card 1 must be a SP -- for 500 switch.

4162 **cveCard2NotSpOrEmpty** — Card 2 must be a SP or be empty -- for 500 switch.

4163 **cveInvalidShaperId** — The Traffic Shaper Id must be in (1, 16).

4164 **cveAscNotSpecified** — Asc string not specified.

4165 **cveHexNotSpecified** — Hex string not specified.

4166 **cveInvalidFormat** — Invalid format for the given object.

4167 **cveFormatNotSpecified** — Format not specified.

4168 **cveSmdsCCodeObjectNotFound** — Smds Country Code object not found.

4169 **cveWtrPeriodOutOfRange** — WtrPeriod for APS should be within 5 to 12.

4170 **cveSfBerExponentOutOfRange** — SfBerExponent for APS should be within 3 to 5.

4171 **cveSdBerExponentOutOfRange** — SdBerExponent for APS should be within 6 to 9.

4172 **cveInvalidRoutingMetric** — No <Cell Delay Variation> for NRT-VBR and UBR routing metric.

4173 **cveNameTooLong** — Name string too long, (Max = 64 chars).

4174 **cveCommentsTooLong** — Comment string too long (Max = 100 chars).

4175 **cveCUGNameMissing** — Missing Close User Group Name.

4176 **cveCUGMbrRuleNameMissing** — Missing Close User Group Member Rule's name.

4177 **cveAfiNotSpecified** — Afi string not specified.

4178 **cveNbitsNotSpecified** — Nbits not specified.

4179 **cveHexNotRequired** — Hex not required for this type of Format.

4180 **cveAscNotRequired** — Asc not required for this type of Format.

4181 **cveTrafficDescNameRequired** — Traffic Descriptor Name Required.

4182 **cveNullSpvcName** — Spvc Name cannot be empty.

4183 **cveNullFwdTrafficDescName** — Fwd Traffic Descriptor Name cannot be empty.

4184 **cveNullRevTrafficDescName** — Rev Traffic Descriptor Name cannot be empty.

4185 **cveNullPMPSpvcRootName** — Root name cannot be empty.

4186 **cveInvalidQosTrfDescType** — Invalid Qos class and/or Traffic Descriptor type.

4187 **cveCannotSetRedFramePercent** — Cannot set Red FramePercent when Graceful Discard is Off.

4188 **cveLastLeafDelete** — Cannot delete the last leaf in an Point to Multipoint Soft PVC circuit.

4189 **cveSmdsNotApply** — Smds configuration does not apply to this object.

4190 **cveBackupLPortNotValid** — BackupLPort cannot be specified in the add request.

4191 **cveBadBindingValue** — Active Binding has to be ‘Backup’ when setting/modifying backup LPort.

4192 **cveCannotRevert** — Primary Binding already active.

4193 **cveArgsForModifyNotSpecified** — Either BackupLPort or ActiveBinding should be given for modify.

4194 **cveRSNotFirstVarbind** — Server agent : RowStatus has to be the first varbind in the Pdu at the time of creation.

4195 **cvePrimaryLPortMissing** — Server agent : Primary LPort needs to be specified for addition.

4196 **cvePsrvSnmpAgentInvalidPdu** — Server agent: invalid PDU.

4197 **cvePsrvSnmpAgentTooManyRowStatusVbs** — Server agent: Too many row status varbinds.

4198 **cvePsrvSnmpAgentTooFewRowStatusVbs** — Server agent: Not enough row status varbinds, row status for both endpoints needed.

4199 **cvePsrvSnmpAgentVbWrongType** — Server agent: Varbinds of wrong type.

4200 **cvePsrvSnmpAgentBadRowStatus** — Server agent: Bad RowSatus state, see description for RowStatus state limits.

4201 **cvePsrvSnmpAgentCktEndptNull** — Server agent: Circuit endpoints have not been set.

4202 **cvePsrvSnmpAgentCktBadIndex** — Server agent: Bad index for circuit endpoint one.

4203 **cvePsrvSnmpAgentRowCreationFailure** — Server agent: Row creation failed, possibly due to a bad index?

4204 **cvePsrvSnmpAgentUnlockFailure** — Server agent: Unlocking object failed.

4205 **cvePsrvSnmpAgentObjectExists** — Server agent: Object already exists, cannot create.

4206 **cveNoRedFrameOnSimpleRateEnf** — Cannot set Red FramePercent when Rate Enforcement scheme is Simple.

4207 **cveBadValueForNdcAttr** — Bad value for Ndc attribute.

4208 **cveBadValueForNtmAttr** — Bad value for Ntm attribute.

4209 **cveNtmAttrDoesNotApply** — Ntm attributes do apply for this type of lport.

4210 **cveInvSwForSvcNodePrefix** — Invalid switch, svcnodeprefix can be configured only on casc500, 9000 or 8000 switches.

4211 **cveInvLPForSvcAddress** — Invalid lport, svcaddress can be configured only on uniDte, uniDce and nni lport types on casc500, 9000 or 8000 switches.

4212 **cveInvLPForSvcPrefix** — Invalid lport, svcprefix can be configured only on uniDte, uniDce and nni lport types on casc500, 9000 or 8000 switches.

4213 **cveInvLPForSvcConfig** — Invalid lport, svcconfig can be configured only on uniDte & uniDce lport types on casc500 switch and also on uniDte, uniDce and nni lport types on 9000 or 8000 switches.

4214 **cveInvLPForUP** — Invalid lport, svcconfig can be can be configured only on uniDte lport type on casc500 switches.

4215 **cveInvSvcAddrString** — couldn't parse string to represent svc address.

4216 **cveInstanceDoesNotExist** — Instance provided does not exist for the table.

4217 **cveInvalidVciForOPTCellTrk** — For Atm OPTimum Cell Trunk lports, the VCI field must be zero in the provisioning server. (indicate that VCI is not required)

4218 **cveIncorrectLeafInstanceNo** — The instance number provided is not valid for add.

4219 **cvePPortRedundancyDisabled** — APS attributes cannot be set as PPort redundancy is disabled.

4220 **cvePsrvSnmpBadCommunityName** — Server agent: Bad community name.

4221 **cveCIRNonZero** — Cir cannot be set when ZeroCIR is on.

4222 **cvePsrvSnmpOnePDUCreateOrModify** — Server agent: in one PDU, complete object creation or modification (using rowstatus) is not allowed.

4223 **cveBcNonZero** — Bc cannot be set when ZeroCIR is on.

4224 **cveBeNonZero** — Be cannot be set when ZeroCIR is on.

Overridable Error Messages

The user can overrride these messages:

6000 **cveoOverrideError**

6001 **cveoSw500PwrThreshExceeded** — The configuration of this additional IOM will cause the total power requirements of all configured IOMs to exceed the capacity of a single chassis power supply. In the event that the redundant chassis power supply is lost, exceeding the capacity of a single power supply could have serious impact on the operation of the switch. Do you wish to continue?

6002 **cveoSharedThreadCondition** — The IOM being configured shares a common switch fabric thread with an adjacent card. Slots 3 and 4, slots 5 and 6, slots 7 and 8, slots 11 and 12, slots 13 and 14, and slots 15 and 16 share common switch fabric threads. If you wish to configure an IOM in this slot you must ensure that no more than 589 Mbps traverses its thread from any combination of traffic from all physical ports on this card and the adjacent card sharing the thread. Do you wish to continue?

6003 **cveoSharedSpThreadCondition** — The IOM being configured shares a common switch fabric thread with the adjacent SP card. If you wish to configure an IOM in this slot you must ensure that no more than 589 Mbps traverses its thread from any combination of traffic from all physical ports on this card. Do you wish to continue?

6004 **cveoSharedThreadBwExceeded** — The sum of bandwidth on the logical ports defined on the adjacent cards shared a common switch fabric thread which has exceeded 600 Mbps.

6005 **cveoSharedSpThreadBwExceeded** — The sum of bandwidth on the logical ports defined on the IOM shared a common switch fabric thread with the adjacent SP card which has exceeded 589 Mbps.

Index

A

Adding
- a member 2-11 to 2-12, 2-38 to 2-39, 3-11 to 3-14
- an argument to an argument list 2-75 to 2-76
- an object 2-13 to 2-15, 2-40 to 2-42, 3-15 to 3-18
- two argument lists 2-77, 3-129

API, overview 1-1

APS, creating ID 2-103

Argument list
- adding an argument to 2-75 to 2-76
- adding two lists 2-77, 3-129
- comparing lists for equality 2-79, 3-128
- comparing lists for inequality 3-128
- converting to printable string format 2-101
- creating a list 2-88
- creating attribute IDs 2-89, 3-128
- creating values for attributes 2-90, 3-128
- deleting pointer to 2-81
- determining error status 2-80, 3-131
- determining if specified argument exists 3-132
- printing 2-92
- reading a value's type at a location 2-102
- reading an integer that represents the data type of an argument 3-134
- reading argument ID at a location 2-87, 3-135
- reading argument value at a location 3-128, 3-138
- reading integer value of an argument 2-82
- reading IP address value of an argument 2-83
- reading object ID value of an argument 2-84
- reading string value of an argument 2-86
- reading the list returned by a synchronous function 2-37, 3-32
- retrieving error status of an argument 2-85, 2-100, 3-133, 3-137
- retrieving number of arguments 2-78, 3-130
- returning the intersection of two lists 2-94, 3-129
- setting integer value for an argument 2-95
- setting IP address value for an argument 2-96
- setting null value for an argument 2-97, 3-136
- setting object ID value for an argument 2-98
- setting string value for an argument 2-99
- specifying variable arguments 1-4, 3-127
- subtracting two lists 2-93, 3-129
- writing text descriptions to an ostream 3-129

Argument, converting to printable string format 2-176

AssignedSvcSecScn, creating ID 2-104 to 2-105

Asynchronous functions, description 1-1 to

1-3
Attribute list. *See* Argument list

C

C interface for API functions 1-1
C++
  classes 3-1 to 3-2
  interface for API functions 1-1
Callback handler 2-65, 3-61
Card
  creating an object's card ID 2-110
  reading slot number from object ID 2-177, 3-71
Channel
  creating an object's channel ID 2-111, 3-96 to 3-97
  reading channel number from object ID 2-178, 3-72
Circuit
  creating an object's ATM circuit ID 2-106 to 2-107, 3-92
  creating an object's circuit ID 3-94 to 3-95
  creating an object's Frame Relay circuit ID 2-113 to 2-114, 3-98 to 3-99
  reading DLCI number from object ID 2-179, 3-73
  reading VCI number from object ID 2-194, 3-87
  reading VPI number from object ID 2-195, 3-88
Converting
  an argument list to printable string format 2-101
  an argument to printable string format 2-176
  an enumerated value to printable string format 2-180
  an object ID to printable string format 2-197
  an object type to printable string format 2-186
  text descriptions of an object ID to an object ID 2-198
  text descriptions to an object type to an object type 2-199
  writing text descriptions of an argument list to an ostream 3-129
  writing text descriptions of an object ID to an ostream 3-70
Creating
  a null object ID 2-122
  an argument list 2-88
  an object's APS ID 2-103
  an object's AssignedSvcSecScn ID 2-104 to 2-105
  an object's ATM circuit ID 2-106 to 2-107, 3-92
  an object's ATM LPort ID 2-108
  an object's ATM Virtual UNI LPort start VPI value 2-172, 3-123
  an object's card ID 2-110, 3-94 to 3-95
  an object's channel ID 2-111, 3-96 to 3-97
  an object's Customer ID 2-112
  an object's E.164 address 3-100
  an object's Frame Relay circuit ID 2-113 to 2-114, 3-98 to 3-99
  an object's Frame Relay LPort ID 2-116 to 2-117, 3-102 to 3-103
  an object's name 3-104 to 3-105
  an object's name and type 3-108 to 3-109
  an object's NetCac ID 2-118
  an object's network ID 2-119 to 2-121, 3-110 to 3-111

an object's PerformanceMonitor ID 2-123
an object's Pfdl ID 2-115
an object's PMP circuit leaf ID 2-124 to 2-125
an object's PMP circuit root ID 2-126 to 2-127
an object's PMP soft PVC root ID 2-129 to 2-130
an object's PMP SpvcLeaf ID 3-112 to 3-113
an object's PMPSpvcLeaf ID 2-128
an object's PPort ID 2-131, 3-114 to 3-115
an object's Service Name ID 2-132
an object's SMDS address prefix ID 2-146 to 2-147
an object's SMDS alien group address ID 2-133 to 2-134
an object's SMDS alien individual address ID 2-135 to 2-136
an object's SMDS country code ID 2-137
an object's SMDS group address ID 2-138 to 2-139
an object's SMDS group screen ID 2-140
an object's SMDS individual screen ID 2-141
an object's SMDS local individual address ID 2-142 to 2-143
an object's SMDS netwide group address ID 2-144 to 2-145
an object's SMDS SSI individual address ID 2-148 to 2-149
an object's soft PVC circuit ID 2-150 to 2-151
an object's SvcAddress 3-116
an object's SvcAddress ID 2-152 to 2-153
an object's SvcConfig ID 2-154
an object's SvcCUG ID 2-155
an object's SvcCUGMbr ID 2-156 to 2-157
an object's SvcCUGMbr member name 3-106 to 3-107
an object's SvcCUGMbrRule ID 2-158
an object's SvcNodePrefix ID 2-159 to 2-160
an object's SvcPrefix ID 2-161 to 2-162
an object's SvcSecScn ID 2-163
an object's SvcSecScnActParam ID 2-164
an object's SvcUserPart ID 2-165 to 2-166
an object's switch ID 2-167 to 2-168, 3-118 to 3-119
an object's TrafficDescriptor ID 2-169
an object's TrafficShaper ID 2-170 to 2-171, 3-120 to 3-121
an object's VPN ID 2-174
an SMDS object's E.164 address 3-89 to 3-91
attribute IDs in an argument list 2-89, 3-128
values for attributes in an argument list 2-90, 3-128
Customer, creating ID 2-112
CvClient class 3-1, 3-6 to 3-7
CvClient::ApiStatus class 3-2
CvClient::Args class 3-1, 3-126 to 3-129
CvClient::Callback class 3-1, 3-61
CvClient::IpAddr class 3-2, 3-139 to 3-141
CvClient::ObjectId class 3-1, 3-69 to 3-70
CvClient::Value class 3-2
CvSVCAddress class 3-2, 3-142 to 3-145

D

Deleting
- a member 2-16 to 2-17, 2-43 to 2-44, 3-19 to 3-22
- a pointer to a string 2-200
- a pointer to an argument list 2-81
- an object 2-18 to 2-19, 2-45 to 2-46, 3-23 to 3-26

Determining if timeout occurred 2-71, 3-67

DLCI
- creating an object's Frame Relay circuit ID 2-113 to 2-114, 3-98 to 3-99
- reading DLCI number from object ID 2-179, 3-73

E

E.164 address
- creating an object's address 3-100
- creating for an SMDS object ID 3-89 to 3-91
- reading from object ID 3-74

Enumerated value, converting to printable string format 2-180

Equality, comparing argument lists for 2-79, 3-128

Error codes returned by the Provisioning Server B-73 to B-79

Errors
- determining if an error occurred in the response for a synchronous request 3-33
- determining if any argument in an argument list contains an error status 2-80, 3-131
- obtaining an error message 2-36, 3-27
- retrieving error code for an argument in argument list 2-85, 2-100, 3-133, 3-137

Establishing a session with the Provisioning Server 2-6 to 2-7, 3-6 to 3-7

Existence of an argument in an argument list 3-132

Extended Super Frame. *See* Pfdl

F

File descriptors, processing activity on 2-69 to 2-70, 3-65 to 3-66

Functions
- asynchronous, description 1-1 to 1-3
- naming conventions 1-4
- operational functions 1-4, 1-5 to 1-6, 2-1, 2-10 to 2-63, 3-10 to 3-59
- select loop processing functions 1-4, 1-6, 2-1, 2-64 to 2-70, 3-60 to 3-67
- session control functions 1-4, 1-5, 2-1, 2-4 to 2-9, 3-5 to 3-9
- synchronous, description 1-1 to 1-2
- utility functions 1-4, 1-7 to 1-9, 2-1, 2-72 to 2-200, 3-68 to 3-145

G

Getting
- an error message 2-36, 3-27
- an object 2-20 to 2-21, 2-47 to 2-48, 3-28 to 3-31
- information for select system call 2-67 to 2-68, 3-63 to 3-64

I

Inequality, comparing argument lists for 3-128

Intersection, of two argument lists 2-94, 3-129

IP address
- reading from object ID 2-184, 2-192, 3-79, 3-85

reading value from argument list 2-83
setting value in argument list 2-96

L

Listing an object 2-22 to 2-32, 2-49 to 2-59, 3-34 to 3-53
LPort
creating an object's ATM LPort ID 2-108
creating an object's ATM Virtual UNI LPort start VPI value 2-172, 3-123
creating an object's Frame Relay LPort ID 2-116 to 2-117, 3-102 to 3-103
reading an object's ATM Virtual UNI LPort start VPI value 2-190, 3-82
reading number from object ID 2-181, 3-76

M

Member
adding 2-11 to 2-12, 2-38 to 2-39, 3-11 to 3-14
deleting 2-16 to 2-17, 2-43 to 2-44, 3-19 to 3-22
Modifying an object 2-33 to 2-35, 2-60 to 2-61, 3-54 to 3-57

N

Name
creating an object's name 3-104 to 3-105
creating an object's SvcCUGMbr member name 3-106 to 3-107
reading name from object ID 3-77
Name and type, creating 3-108 to 3-109
Naming conventions for functions 1-4
NavisXtend Provisioning Server. *See* Provisioning Server
NetCac, creating ID 2-118
Network ID
creating an object's network ID 2-119 to 2-121, 3-110 to 3-111
reading IP address from object ID 2-184, 3-79
Next object, retrieving 2-62 to 2-63, 3-58 to 3-59
Null object, creating 2-122
Null value, setting 2-97, 3-136

O

Object
adding 2-13 to 2-15, 2-40 to 2-42, 3-15 to 3-18
deleting 2-18 to 2-19, 2-45 to 2-46, 3-23 to 3-26
listing 2-22 to 2-32, 2-49 to 2-59, 3-34 to 3-53
modifying 2-33 to 2-35, 2-60 to 2-61, 3-54 to 3-57
retrieving 2-20 to 2-21, 2-47 to 2-48, 3-28 to 3-31
retrieving next object 2-62 to 2-63, 3-58 to 3-59
Object ID
converting from text descriptions 2-198
converting to printable string format 2-197
creating E.164 address for 3-89 to 3-91
printing 2-196
reading a name from 2-182
reading an SMDS address from 2-175
reading an SvcAddress from 2-191
reading an SvcCUGMbr member name from 2-183, 3-78

reading card slot number from 2-177, 3-71
reading channeI number from 2-178
reading channel number from 3-72
reading DLCI number from 2-179, 3-73
reading E.164 address from 3-74
reading LPort number from 2-181, 3-76
reading name from 3-77
reading network IP address from 2-184, 3-79
reading object type from 2-185, 3-125
reading PMP SpvcLeaf number from 3-80
reading PMPSpvcLeaf number from 2-187
reading PPort number from 2-188, 3-81
reading SMDS address prefix from 2-189
reading SvcAddress from 3-83
reading switch IP address from 2-192, 3-85
reading TrafficShaper identifier value from 2-193
reading VCI number from 2-194, 3-87
reading VPI number from 2-195, 3-88
setting the object type 3-122
writing text descriptions of an object ID to an ostream 3-70

Object type
converting from text description 2-199
converting to printable string format 2-186
setting 3-122

Operational functions 1-4, 1-5 to 1-6, 2-1, 2-10 to 2-63, 3-10 to 3-59

Ostream
writing text descriptions of an argument list to 3-129
writing text descriptions of an object ID to 3-70

P

PerformanceMonitor, creating ID 2-123
Pfld, creating ID 2-115
PMP circuit leaf, creating ID 2-124 to 2-125
PMP circuit root, creating ID 2-126 to 2-127
PMP soft PVC root, creating ID 2-129 to 2-130

PMPSpvcLeaf
creating an object's PMP SpvcLeaf ID 3-112 to 3-113
creating an object's PMPSpvcLeaf ID 2-128
reading number from object ID 3-80
reading PMPSpvcLeaf number from object ID 2-187

PPort
creating an object's PPort ID 2-131, 3-114 to 3-115
reading number from object ID 2-188, 3-81

Printing
an argument list 2-92
an object ID 2-196

Processing activity on file descriptors 2-69 to 2-70, 3-65 to 3-66

Provisioning Server
adding a member 2-11 to 2-12, 2-38 to 2-39, 3-11 to 3-14
adding an object 2-13 to 2-15, 2-40 to 2-42, 3-15 to 3-18
callback handler 2-65, 3-61
deleting a member 2-16 to 2-17, 2-43 to 2-44, 3-19 to 3-22
deleting an object 2-18 to 2-19, 2-45 to 2-46, 3-23 to 3-26

determining if timeout occurred 2-71, 3-67
listing an object 2-22 to 2-32, 2-49 to 2-59, 3-34 to 3-53
modifying an object 2-33 to 2-35, 2-60 to 2-61, 3-54 to 3-57
obtaining an error message 2-36, 3-27
obtaining information for select system call 2-67 to 2-68, 3-63 to 3-64
processing activity on file descriptors 2-69 to 2-70, 3-65 to 3-66
retrieving an object 2-20 to 2-21, 2-47 to 2-48, 3-28 to 3-31
retrieving the next object 2-62 to 2-63, 3-58 to 3-59
session, establishing 2-6 to 2-7, 3-6 to 3-7
session, terminating 2-5, 3-9
updating the database 2-8, 3-8

R

Reading
a card slot number out of an object ID 2-177, 3-71
a channel number out of an object ID 2-178, 3-72
a circuit VCI out of an object ID 2-194, 3-87
a circuit VPI out of an object ID 2-195, 3-88
a DLCI number out of an object ID 2-179, 3-73
a name out of an object ID 2-182, 3-77
a network IP address out of an object ID 2-184, 3-79
a PMP SpvcLeaf number out of an object ID 3-80
a PPort number out of an object ID 2-188, 3-81
a string value of argument in an argument list 2-86
a switch IP address out of an object ID 2-192, 3-85
a TrafficShaper identifier value out of an object ID 2-193
a value's type at a location in an argument list 2-102
an argument ID at a location in an argument list 2-87, 3-135
an argument value at a location in an argument list 3-138
an E.164 address out of an object ID 3-74
an integer that represents the data type of an argument in an argument list 3-134
an integer value of argument in an argument list 2-82
an IP address value of argument in an argument list 2-83
an LPort number out of an object ID 2-181, 3-76
an object ID value of argument in an argument list 2-84
an object type out of an object ID 2-185, 3-125
an PMPSpvcLeaf number out of an object ID 2-187
an SMDS address out of an object ID 2-175
an SMDS address prefix out of an object ID 2-189
an SvcAddress out of an object ID 2-191, 3-83
an SvcCUGMbr member name out of an object ID 2-183, 3-78

argument list returned by a synchronous function 2-37, 3-32
argument value at a location in an argument list 3-128
the start VPI value out of an object ID for Virtual UNI LPort 2-190, 3-82
Required parameters, determining if one was omitted 2-37, 3-32
Retrieving
an object 2-20 to 2-21, 2-47 to 2-48, 3-28 to 3-31
error status of an argument in an argument list 2-85, 2-100, 3-133, 3-137
number of arguments in an argument list 2-78, 3-130
the next object 2-62 to 2-63, 3-58 to 3-59
Returning the intersection of two argument lists 2-94, 3-129

S

Select loop processing functions 1-4, 1-6, 2-1, 2-64 to 2-70, 3-60 to 3-67
Service Name, creating ID 2-132
Session
establishing 2-6 to 2-7, 3-6 to 3-7
terminating 2-5, 3-9
Session control functions 1-4, 1-5, 2-1, 2-4 to 2-9, 3-5 to 3-9
Setting
a null value for argument in an argument list 2-97, 3-136
a string value for argument in an argument list 2-99
an integer value for argument in an argument list 2-95
an IP address value for argument in an argument list 2-96
an object ID value for argument in an argument list 2-98
the type of an object ID 3-122
SMDS address prefix, creating ID 2-146 to 2-147
SMDS address prefix, reading from object ID 2-189
SMDS alien group address, creating ID 2-133 to 2-134
SMDS alien individual address, creating ID 2-135 to 2-136
SMDS country code, creating ID 2-137
SMDS group address, creating ID 2-138 to 2-139
SMDS group screen, creating ID 2-140
SMDS individual screen, creating ID 2-141
SMDS local individual address, creating ID 2-142 to 2-143
SMDS netwide group address, creating ID 2-144 to 2-145
SMDS objects, creating an object's E.164 address 3-89 to 3-91
SMDS SSI individual address, creating ID 2-148 to 2-149
Soft PVC circuit, creating ID 2-150 to 2-151
Start VPI for Virtual UNI LPort 2-172, 2-190, 3-82, 3-123
Starting a session with the Provisioning Server 2-6 to 2-7, 3-6 to 3-7
String, deleting pointer to 2-200
Subtracting two argument lists 2-93, 3-129
SvcAddress
creating an object's address 2-152 to 2-153, 3-116
reading from object ID 3-83
SvcConfig, creating ID 2-154
SvcCUG, creating ID 2-155
SvcCUGMbr, creating ID 2-156 to 2-157

SvcCUGMbrRule, creating ID 2-158
SvcNodePrefix, creating ID 2-159 to 2-160
SvcPrefix, creating ID 2-161 to 2-162
SvcSecScn, creating ID 2-163
SvcSecScnActParam, creating ID 2-164
SvcUserPart, creating ID 2-165 to 2-166
Switch
  creating ID 2-167 to 2-168, 3-118 to 3-119
  reading IP address from object ID 2-192, 3-85
Synchronous functions, description 1-1 to 1-2

T

Terminating a session with the Provisioning Server 2-5, 3-9
Timeout, determining if occurred 2-71, 3-67
TrafficDescriptor, creating ID 2-169
TrafficShaper
  creating an object's TrafficShaper ID 2-170 to 2-171, 3-120 to 3-121
  reading index value from object ID 2-193

U

Updating the database 2-8, 3-8
Utility functions 1-4, 1-7 to 1-9, 2-1, 2-72 to 2-200, 3-68 to 3-145

V

Variable argument list 3-127
VPI
  creating the start VPI value for Virtual UNI LPort 2-172, 3-123
  reading the start VPI value for Virtual UNI LPort 2-190, 3-82
VPN, creating ID 2-174

W

Writing
  text descriptions of an argument list to an ostream 3-129
  text descriptions of an object ID to an ostream 3-70