NavisXtend Provisioning Server Programmer's Reference

Ascend Communications, Inc.

Product Code: 80066
Revision 00
November 1998

Copyright © 1998 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 xvii
Documentation Reading Path xviii
How to Use This Guide xix
What's New in This Release? xix
What's New in This Guide? xxii
Conventions xxii
Related Documents xxiii
Customer Comments xxiv
Customer Support xxv
    Terminology xxv

Section 1 API Overview

Chapter 1 Overview

The NavisXtend Provisioning Server API 1-1
    Servicing Asynchronous Functions 1-2
    Function Names 1-3
    Functions That Take an Argument List 1-4
    API Functions by Group 1-4
Valid Object Types for Operational Functions 1-10
    Reusing an Existing Object ID 1-12
        Example 1 1-12
        Example 2 1-12

Section 2 C Functions

Chapter 2 Session Control Functions

CvClose 2-2
CvConnect 2-3
CvSetModifyType 2-5

CvSetNumRetries 2-6

Chapter 3 Operational Functions

CvaAddMember 3-2
CvaAddObject 3-4
CvaDeleteMember 3-6
CvaDeleteObject 3-8
CvaGetDiagObject 3-10
CvaGetList 3-12
CvaGetObject 3-14
CvaGetOperInfoObject 3-16
CvaListAllContainedObjects 3-18
CvaListContainedObjects 3-21
CvaModifyList 3-26
CvaModifyObject 3-28
CvaStartDiagObject 3-30
CvaStopDiagObject 3-32
CvaUpdateDiagObject 3-33
CvsAddMember 3-35
CvsAddObject 3-37
CvsDeleteMember 3-39
CvsDeleteObject 3-41
CvsGetDiagObject 3-42
CvsGetList 3-44
CvsGetObject 3-46
CvsGetOperInfoObject 3-48
CvsListAllContainedObjects 3-49
CvsListContainedObjects 3-53
CvsModifyListObject 3-58
CvsModifyObject 3-60
CvsNextObject 3-62
CvsStartDiagObject 3-64
CvsStopDiagObject 3-66
CvsUpdateDiagObject 3-67

Chapter 4 Select Loop Functions

CVCBH 4-2
CvGetSelectInfo 4-3
CvProcessEvents 4-5
CvTimeout 4-7
OLCBH 4-8

Chapter 5 Utility Functions

CvAddArgumentByName 5-2
CvAddArgumentByNameValue 5-3
CvArgsCombine 5-4
CvArgsCount 5-5

CvArgsEqual 5-6
CvArgsErrorIndex 5-7
CvArgsFree 5-8
CvArgsGetInt 5-9
CvArgsGetIpAddr 5-10
CvArgsGetObjectId 5-11
CvArgsGetStatus 5-12
CvArgsGetStr 5-13
CvArgsIdAt 5-14
CvArgsMake 5-15
CvArgsMakeIds 5-16
CvArgsMakeVals 5-17
CvArgsPrint 5-18
CvArgsRemove 5-19
CvArgsSelect 5-20
CvArgsSetInt 5-21
CvArgsSetIpAddress 5-22
CvArgsSetNull 5-23
CvArgsSetObjectId 5-24
CvArgsSetStr 5-25
CvArgsStatusAt 5-26
CvArgsToString 5-27
CvArgsTypeAt 5-28
CvCreateApsId 5-29
CvCreateAssignedSvcSecScnId 5-30
CvCreateAtmEndpointId 5-31
CvCreateAtmLPortId 5-32
CvCreateCardId 5-33
CvCreateCardTcaId 5-34
CvCreateChannelId 5-35
CvCreateCircuitDefinedPathId 5-36
CvCreateCustomerId 5-37
CvCreateDlciId 5-38
CvCreateFdlId 5-39
CvCreateLPortId 5-40
CvCreateNetCacId 5-41
CvCreateNetworkId 5-42
CvCreateNetworkIdFromString 5-43
CvCreateNullId 5-44
CvCreatePerformanceMonitorId 5-45
CvCreatePMPCktLeafId 5-46
CvCreatePMPCktRootId 5-47
CvCreatePMPSpvcLeafId 5-48
CvCreatePMPSpvcRootId 5-49
CvCreatePnniNodeId 5-50
CvCreatePPortId 5-51
CvCreatePPortTcaId 5-52
CvCreateRefTimeServer 5-53

CvCreateServiceNameId 5-54
CvCreateSmdsAlienGroupAddrId 5-55
CvCreateSmdsAlienIndivAddrId 5-56
CvCreateSmdsCountryCodeId 5-57
CvCreateSmdsGroupAddrId 5-58
CvCreateSmdsGroupScreenId 5-59
CvCreateSmdsIndivScreenId 5-60
CvCreateSmdsLocalIndivAddrId 5-61
CvCreateSmdsNetwideGroupAddrId 5-62
CvCreateSmdsPrefixId 5-63
CvCreateSmdsSsiIndivAddrAddrId 5-64
CvCreateSpvcId 5-65
CvCreateSvcAddressId 5-66
CvCreateSvcConfigId 5-67
CvCreateSvcCUGId 5-68
CvCreateSvcCUGMbrId 5-69
CvCreateSvcCUGMbrRuleId 5-70
CvCreateSvcNetworkId 5-71
CvCreateSvcNodePrefixId 5-72
CvCreateSvcPrefixId 5-73
CvCreateSvcSecScnId 5-74
CvCreateSvcSecScnActParamId 5-75
CvCreateSvcUserPartId 5-76
CvCreateSwitchId 5-77
CvCreateSwitchIdFromString 5-78
CvCreateTrafficDescId 5-79
CvCreateTrafficShaperId 5-80
CvCreateTrunkId 5-81
CvCreateVirtualUniLPortId 5-82
CvCreateVpciTableId 5-83
CvCreateVPNId 5-84
CvGetAddress 5-85
CvGetArgumentName 5-86
CvGetCardNumber 5-87
CvGetChannelNumber 5-88
CvGetDlciNumber 5-89
CvGetEnumName 5-90
CvGetErrorMsg 5-91
CvGetLPortNumber 5-92
CvGetName 5-93
CvGetName2 5-94
CvGetNetworkNumber 5-95
CvGetNumRetries 5-96
CvGetObjectType 5-97
CvGetObjectTypeName 5-98
CvGetPMPSpvcLeafId 5-99
CvGetPnniNodeId 5-100
CvGetPPortNumber 5-101

CvGetPrefix 5-102
CvGetRefTimeServerIpAddr 5-103
CvGetResponseArgs 5-104
CvGetResponseObjList 5-105
CvGetStartVpi 5-106
CvGetSvcAddress 5-107
CvGetSwitchId 5-108
CvGetTrafficShaperId 5-109
CvGetVciNumber 5-110
CvGetVpciNumber 5-111
CvGetVpiNumber 5-112
CvObjectIdPrint 5-113
CvObjectIdToString 5-114
CvObjectListAdd 5-115
CvObjectListCombine 5-116
CvObjectListCount 5-117
CvObjectListErrorIndex 5-118
CvObjectListFree 5-119
CvObjectListGetArgs 5-120
CvObjectListGetStatus 5-121
CvObjectListIdAt 5-122
CvObjectListMake 5-123
CvObjectListPrint 5-124
CvObjectListRemove 5-125
CvObjectListSelect 5-126
CvObjectListSetNull 5-127
CvObjectListStatusAt 5-128
CvObjectListSubtract 5-129
CvParseObjectId 5-130
CvParseObjectType 5-131
CvSetNumRetries 5-132
CvStringFree 5-133

Section 3 C++ Functions

Chapter 6 CvClient

Session Control 6-2
Establishing a Session 6-3
setModifyType 6-5
setNumRetries 6-6
Terminating a Session 6-7
Operational Functions 6-8
addMember (asynchronous) 6-9
addMember (synchronous) 6-11
addObject (asynchronous) 6-13
addObject (synchronous) 6-15

deleteMember (asynchronous) 6-17
deleteMember (synchronous) 6-19
deleteObject (asynchronous) 6-21
deleteObject (synchronous) 6-22
getDiagObject (asynchronous) 6-23
getDiagObject (synchronous) 6-25
getErrorMsg 6-26
getObject (asynchronous) 6-27
getObject (asynchronous, ObjectList) 6-29
getObject (synchronous, Args) 6-31
getObject (synchronous, ObjectList) 6-32
getOperInfoObject (asynchronous) 6-33
getOperInfoObject (synchronous) 6-35
getResponseArgs 6-36
getResponseErrorIndex 6-37
listContainedObjects (asynchronous, list all) 6-38
listContainedObjects (synchronous, list all) 6-41
listContainedObject (asynchronous, list) 6-44
listContainedObject (synchronous, list) 6-49
modifyObject (asynchronous, Args) 6-54
modifyObject (asynchronous, ObjectList) 6-56
modifyObject (synchronous, Args) 6-58
modifyObject (synchronous, ObjectList) 6-60
nextObject 6-61
startDiagObject (asynchronous) 6-62
startDiagObject (synchronous) 6-64
stopDiagObject (asynchronous) 6-66
stopDiagObject (synchronous) 6-67
updateDiagObject (asynchronous) 6-68
updateDiagObject (synchronous) 6-70

Chapter 7 CvClient::Callback

CvClient::Callback (Client Callback Handler class) 7-2
getSelectInfo 7-4
processEvents 7-6
timeout 7-8

Chapter 8 CvClient::ObjectId

CvClient::ObjectId 8-2
getCard 8-4
getChannel 8-5
getDlci 8-6
getE164Address 8-7
getLPort 8-8
getName 8-9
getName2 8-10
getNetwork 8-11
getNumRetries 8-12

getPMPSpvcLeafNo 8-13
getPnniNodeId 8-14
getPPort 8-15
getRefTimeServerIpAddr 8-16
getStartVpi 8-17
getSVCAddress 8-18
getSwitch 8-19
getTrafficShaper 8-20
getVci 8-21
getVpciNumber 8-22
getVpi 8-23
set 8-24
setAtmInfo 8-26
setCard 8-27
setChannel 8-28
setDlci 8-29
setE164Address 8-30
setLPort 8-32
setName 8-33
setName2 8-35
setNameAndType 8-36
setNetwork 8-38
setPMPSpvcLeafNo 8-40
setPnniNodeId 8-42
setPPort 8-44
setRefTimeServerIpAddr 8-45
setSVCAddressAndType 8-47
setSwitch 8-49
setTrafficShaper 8-50
setType 8-51
setVirtualUniLPort 8-52
setVpciNumber 8-53
type 8-55

Chapter 9 CvClient::Args

CvClient::Args 9-2
count 9-4
errorIndex 9-5
exists 9-6
getStatus 9-7
getType 9-8
idAt 9-9
setNull 9-10
statusAt 9-11
valueAt 9-12

Chapter 10 CvClient::ObjectList

CvClient::ObjectList 10-2
addToObjectList 10-4
argsAt 10-5
count 10-6
errorIndex 10-7
exists 10-8
idAt 10-9
removeFromObjectList 10-10
statusAt (CvClient::ObjectId&) 10-11
statusAt (int) 10-12

Chapter 11 CvClient::IpAddr

CvClient::IpAddr 11-2

Chapter 12 CvSVCAddress

CvSVCAddress 12-2

Section 4 Appendixes

Appendix A Sample Code

C Sample Code A-1
C++ Sample Code A-17
Sample Code for SMDS Objects A-17
Sample Code for Other Objects A-32

Appendix B Error Codes

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 1-4
Table 1-2. Operational Functions 1-5
Table 1-3. Select Loop Processing Functions 1-6
Table 1-4. Utility Functions 1-6
Table 1-5. Valid Object Types for Operational Functions 1-10
Table 3-1. Valid Parent and Child Object Types 3-18
Table 3-2. Valid Parent and Child Object Types 3-22
Table 3-3. Valid Parent and Child Object Types 3-50
Table 3-4. Valid Parent and Child Object Types 3-54
Table 6-1. Valid Parent and Child Object Types 6-38
Table 6-2. Valid Parent and Child Object Types 6-41
Table 6-3. Valid Parent and Child Object Types 6-45
Table 6-4. Valid Parent and Child Object Types 6-50
Table 12-1. Calculated nBits Values 12-4

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 NavisCore is available through the API. You use the client to query and configure switch nodes, cards, physical ports, circuits, and other objects.

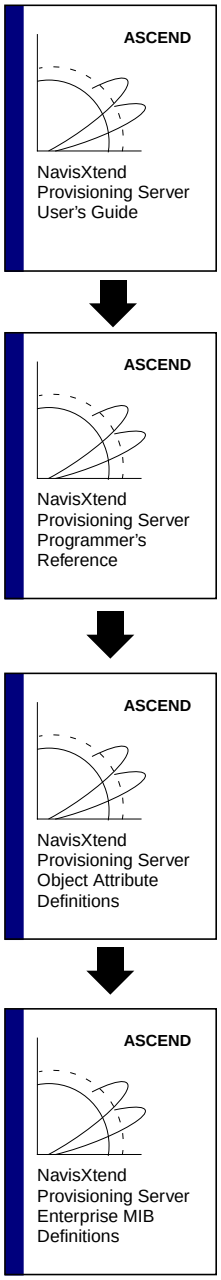
What You Need to Know

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.

Be sure to read the *Software Release Notice (SRN) for NavisXtend Provisioning Server* that accompanies this product. The SRN contains the most current product information and requirements.

Documentation Reading Path

The NavisXtend Provisioning Server document set 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
Section 1 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.
Section 2 Chapter 2 Chapter 3 Chapter 4 Chapter 5	The C functions of the NavisXtend Provisioning Server API.
Section 3 Chapter 6 Chapter 7 Chapter 8 Chapter 9 Chapter 10 Chapter 11 Chapter 12	The C++ classes and member functions of the NavisXtend Provisioning Server API.
Section 4 Appendix A Appendix B	Commented source listings of a sample NavisXtend client. Use this information as a guide when you write your program. Error codes that can be returned by the functions of the NavisXtend Provisioning Server API.

What's New in This Release?

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

New Features/Functions	Description
Compatibility with NavisCore database, version 04.01.01.00	The Provisioning Server Release 4.1 interacts with the NavisCore version 04.01.01.00 database. Keep in mind that the server does not support **all** the new NavisCore 04.01.01.00 object types.

New Features/Functions	Description
Support of the following new cards: • 1-port channelized DS3-1-0 card (on B-STDX) • 6-port DS3 Frame card (on CBX) • CP40, CP50 • SP30, SP 40	The Provisioning Server Release 4.1 can manage these cards.
Limited support of GX 550 switch	The Provisioning Server Release 4.1 supports configuration of elements that are part of GX 550 switches.
Support of the following new objects: • Card Threshold Crossing Alarm • Circuit Defined Path • PPort Threshold Crossing Alarm • Private Network-to-Network (PNNI) Node • Reference Time Server • SvcNetwork ID • Trunk • VPCI Table	The Provisioning Server Release 4.1 supports these new objects and their associated attributes.
SNMPv2c protocol	The Provisioning Server SNMP agent supports the SNMPv2c protocol.
Improved reliability and accuracy of circuit provisioning	Circuit provisioning on the B-STDX 8000/9000 and the CBX 500 is improved, preventing circuits from being partially provisioned and the database from becoming out-of-sync with the switch.
Diagnostic trace information added	Diagnostic trace information has been added to the Provisioning Server to print MIB interface related interaction. This information can assist in troubleshooting problems.

New Features/Functions	Description
Support of a VPI value of 0 or greater	The Provisioning Server supports a VPI value of 0 or greater when provisioning an ATM OPT Cell Trunk LPort on the CBX 500 switch.
Real time status of NavisCore objects	**GetOperInfoObject**, an operational function, retrieves the real time status of NavisCore objects at the PVC level. API, CLI, and MIB interfaces are available.
Diagnostics	A set of operational functions performs diagnostic services for troubleshooting at the Circuit, Channel, PPort, and LPort levels.

What's New in This Guide?

The following table lists the enhancements to this guide:

Changes/Enhancements to this Guide	Described in Chapter
New session control function: SetNumRetries	2
New operational functions: GetDiagObject, GetList, GetOperInfoObject, ModifyList, StartDiagObject, StopDiagObject, UpdateDiagObject	3,
New select loop function: OLCBH	4
New utility functions: GetNumRetries, GetResponseObjList, ObjectListAdd, ObjectListCombine, ObjectListCount, ObjectListErrorIndex, ObjectListFree, ObjectListGetArgs, ObjectListGetStatus, ObjectListIdAt, ObjectListMake, ObjectListPrint, SetNumRetries	5
New CvClient functions: getDiagObject, getObject, getOperInfoObject, modifyObject, setNumRetries, startDiagObject, stopDiagObject, updateDiagObject	6
New CvClient::ObjectId function: getNumRetries	8
New CvClient::ObjectList class and functions:	10
New CvSvcAddress formatting	12
Updated error codes	Appendix B

Conventions

This guide uses the following conventions:

Convention	Indicates	Example
`Courier`	Program source code. User input on a separate line and screen or system output.	`unsigned long` `Please wait...`
Helvetica	Structure names or other source code in body text.	CvObjectId structure

Convention	Indicates	Example
Bold	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*
Boxes around text	Notes, warnings, cautions.	See examples below.



Notes provide additional information or helpful suggestions that may apply to the subject text.



Cautions notify the reader to proceed carefully to avoid possible equipment damage or data loss.



Warnings notify the reader to proceed carefully to avoid possible personal injury.

Related Documents

This section lists the related Ascend documentation that you may find helpful to read.

- *Network Management Station Installation Guide* (Product code: #80014)
- *NavisCore Frame Relay Configuration Guide* (Product code: #80071)
- *NavisCore ATM Configuration Guide* (Product code: #80072)
- *NavisCore SMDS Configuration Guide* (Product code: #80073)

Customer Comments

Customer comments are welcome. Please respond in one of the following ways:

- Fill out the Customer Comment Form located at the back of this guide and return it to us.
- E-mail your comments to cspubs@ascend.com
- FAX your comments to 978-692-1510, attention Technical Publications.

Customer Support

To obtain release notes, technical tips, or support, access the Ascend FTP Server or contact the Technical Assistance Center at:

- 1-800-DIAL-WAN or 1-978-952-7299 (U.S. and Canada)
- 0-800-96-2229 (U.K.)
- 1-978-952-7299 (all other areas)

Terminology

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*.

Section 1
API Overview

Chapter 1 Overview

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 NavisCore 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 (see 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, see Chapter 4, “Select Loop Functions.” For more information on these functions in C++, see Chapter 7, “CvClient::Callback.”

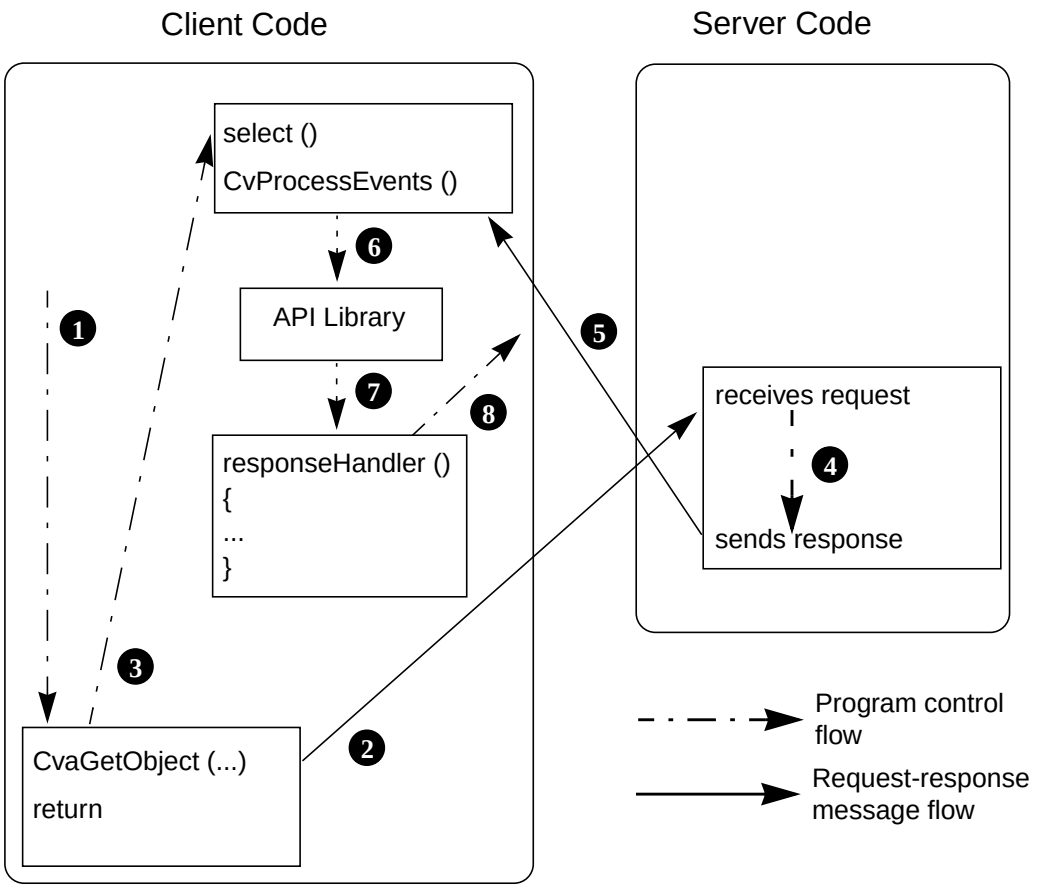


Figure 1-1. Flow of Control for an Asynchronous Function

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 any function, see the appropriate function chapter in Section 2 or Section 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.
Set Number of Retries	**CvSetNumRetries**	**setNumRetries**	Sets the number of retries to check card status preceding circuit provisioning requests.

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 a member to an object list.
Modify	**CvaModifyList** **CvsModifyList** **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 a member from an object list.
DiagObject	**CvaStartDiagObject** **CvsStartDiagObject** **CvaUpdateDiagObject** **CvsUpdateDiagObject** **CvaGetDiagObject** **CvsGetDiagObject** **CvaStopDiagObject** **CvsStopDiagObject**	**startDiagObject** **updateDiagObject** **getDiagObject** **stopDiagObject**	Starts diagnostics on an object in the network. Modifies the diagnostic parameter on an object in the network. Retrieves diagnostic results on an object in the network. Stops diagnostics on an object in the network.
Get	**CvaGetList** **CvsGetList** **CvaGetObject** **CvsGetObject**	**getObject**	Retrieves the values of specific attributes.
GetOperInfoObject	**CvaGetOperInfoObject** **CvsGetOperInfoObject**	**getOperInfoObject**	Retrieves the values of specific OperInfo attributes from the switch.
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.

Table 1-2. Operational Functions (Continued)

Operation	C Function	CvClient Member Function	Description
List Next	**CvsNextObject**	**nextObject**	Retrieves the next object in a list of objects.
Get Error Msg	**CvGetErrorMsg**	**getErrorMsg**	Returns an error message.

Table 1-3. Select Loop Processing Functions

Operation	C Function	CvClient Member Function	Description
Client Callback Handler	**CVCBH** **OLCBH**	**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** **CvObjectListMake**	Construct an **Args** class.	Creates argument list or ObjectList.
Free	**CvArgsFree** **CvObjectListFree**	—	Deletes a pointer to an argument or ObjectList list.

Table 1-4. Utility Functions (Continued)

Operation	C Function	CvClient Member Function	Description
Set	**CvArgsSetInt** **CvArgsSetStr** **CvArgsSetIpAddr** **CvArgsSetObjectId** **CvArgsSetNull** **CvArgsSetResponseArgs**	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** **CvObjectListGetArgs**	operator **[ ]**	Reads a value for a given argument ID out of the argument or object list.
Get	**CvGetResponseArgs** **CvGetResponseObjList**	**getResponseArgs**	Returns the argument list returned by a synchronous function.
Count	**CvArgsCount** **CvObjectListCount**	**Args::count** **ObjectList::count**	Retrieves the number of arguments in the argument list, or the number of objects in an ObjectList.
Get ID At Index	**CvArgsIdAt** **CvObjectListIdAt**	**Args::idAt** **ObjectList::idAt**	Retrieves the argument ID or ObjectId at a specified index location in the argument list or ObjectList.
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** **CvObjectListErrorIndex**	**Args::errorIndex** **ObjectList::error Index**	Determines if any argument in an argument list, or object in an object list, has an error status.
Test for Existence	—	**Args::exists** **ObjectList::exists**	Determines if a specified argument exists in an argument list.
Get Status	**CvArgsGetStatus** **CvArgsStatusAt** **CvObjectListGetStatus**	**Args::getStatus** **Args::statusAt** **ObjectList::getStatus**	Returns the error status code of a specified argument or object in an argument list or ObjectList.
Compare	**CvArgsEqual**	operator **=** operator **!=**	Compares two argument lists for equality.

Table 1-4. Utility Functions (Continued)

Operation	C Function	CvClient Member Function	Description
Add	**CvArgsCombine** **CvObjectListCombine**	operator +	Adds two argument lists or ObjectLists together, resulting in a new argument list that combines both sets of arguments.
Add	**CvObjectListAdd**	**ObjectList::addTo ObjectList**	Adds an ObjectId and Args to an ObjectList.
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** **CvObjectListPrint**	operator <<	Prints the text description of an argument list or CvObjectList to a file.
Make	**CvCreateChanPerforma nceMonitorId**	—	Creates a CVT_ChanPerformanceMonitor object
Make	**CvCreate***ObjectType***Id**	**set***ObjectType*	Creates an object ID of a specified type, based on the containment hierarchy.
Make	**CvCreateNetworkId FromString** **CvCreateSwitchIdFrom String**	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.

Table 1-4. Utility Functions (Continued)

Operation	C Function	CvClient Member Function	Description
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***ObjectTypeValue*	**get***ObjectType*	Reads a specified value out of the object ID.
Get Number of Retries	**CvGetNumRetries**	**getNumRetires**	Gets the number of retries to check card status preceding circuit provisioning requests.
Print	**CvObjectIdPrint**	operator <<	Prints the text description of an object ID to a file.

Valid Object Types for Operational Functions

Table 1-5 lists the object types you can use when you issue the operational functions and commands of the API and CLI.

Table 1-5. Valid Object Types for Operational Functions

Object Type	Add Object	Add Member	Delete Object	Delete Member	Get	List All	List	Modify	Diag Object	Get Oper Info
APS					✓		✓	✓		
AssignedSvcSecScn	✓		✓		✓	✓	✓	✓		
Card					✓	✓	✓	✓		
CardTca					✓			✓		
Channel					✓	✓	✓	✓	✓	
Channel Performance Monitor					✓		✓	✓		
Circuit	✓		✓		✓		✓	✓	✓	✓
Customer	✓		✓		✓	✓	✓	✓		
DefinedPath			✓		✓			✓		
LPort	✓	✓[1]	✓	✓[2]	✓	✓	✓	✓	✓	
NetCac					✓		✓	✓		
Performance Monitor					✓		✓	✓		
PFdl					✓		✓	✓		
PMP Circuit Leaf Endpt	✓		✓		✓		✓	✓		
PMP Circuit Root Endpt	✓		✓		✓	✓	✓			
PMP SPVC Leaf Endpt	✓		✓		✓		✓	✓		
PMP SPVC Root Endpt	✓		✓		✓	✓	✓	✓		
PnniNode	✓		✓		✓		✓	✓		
PPort					✓	✓	✓	✓	✓	
PPortTca					✓			✓		
RefTimeServer	✓		✓		✓		✓	✓		
Service Name	✓		✓		✓	✓	✓	✓		
SMDS Address Prefix	✓		✓		✓	✓	✓	✓		
SMDS Alien Group Address	✓		✓		✓	✓	✓	✓		

Table 1-5. Valid Object Types for Operational Functions (Continued)

Object Type	Add Object	Add Member	Delete Object	Delete Member	Get	List All	List	Modify	Diag Object	Get Oper Info
SMDS Alien Individual Address	✓		✓		✓	✓	✓	✓		
SMDS Country Code	✓		✓		✓	✓	✓	✓		
SMDS Group Screen	✓	✓	✓	✓	✓	✓	✓	✓		
SMDS Individual Screen	✓	✓	✓	✓	✓	✓	✓	✓		
SMDS Local Individual Address	✓		✓		✓	✓	✓	✓		
SMDS Netwide Group Address	✓	✓	✓	✓	✓	✓	✓	✓		
SMDS Switch Group Address			✓		✓	✓	✓	✓		
Soft PVC Circuit	✓		✓		✓		✓	✓		
SVC Address	✓		✓		✓		✓	✓		
SVC Config					✓			✓		
SVC CUG	✓		✓		✓		✓	✓		
SVC CUG Member	✓		✓		✓		✓	✓		
SVC CUG Mbr Rule	✓		✓		✓		✓	✓		
SVC Network ID	✓		✓		✓	✓	✓	✓		
SVC Node Prefix	✓		✓		✓		✓	✓		
SVC Prefix	✓		✓		✓		✓	✓		
SVC Security Screen	✓		✓		✓	✓	✓	✓		
SVC SecScnActParam					✓	✓	✓	✓		
SVC UserPart	✓		✓		✓		✓			
Switch					✓	✓	✓	✓		
Traffic Descriptor	✓		✓		✓		✓			
Traffic Shaper					✓		✓	✓		
Trunk	✓		✓		✓	✓	✓	✓		
VPCI Table	✓		✓		✓	✓	✓	✓		
Vitual Private Network	✓		✓		✓	✓	✓	✓		

1 For binding an MLFRMember LPort to an MLFRBundle LPort

2 For unbinding an MLFRMember LPort to an MLFRBundle LPort

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
```

Section 2
C Functions

Chapter 2	Session Control Functions
Chapter 3	Operational Functions
Chapter 4	Select Loop Functions
Chapter 5	Utility Functions

2

Session Control Functions

Session Control functions open and close sockets and control session settings.

CvClose

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 NavisCore.

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

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 (see Chapter 2 of the *NavisXtend Provisioning Server User's Guide* for details). When issuing this function, pass in the character strings required by NavisCore 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. Always use modifyNormal when **startDiag**, **getDiag**, **updateDiag**, **stopDiag**, or **GetOperInfo** are used.

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.

CvSetNumRetries

Purpose

Sets the number of retries to check card status preceding circuit provisioning requests.

Syntax

CvStatus CvSetNumRetries(
CvServerHandle *Handle***,**
int *NumRetries* **);**

Description

CvSetNumRetries sets the number of retries to check card status preceding circuit provisioning requests.

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

NumRetries is the number of retries. The value applies to requests at either endpoint: when a retry is sent to obtain the card status of one endpoint, the number of retries decrements for either endpoint. Valid values 0 - 5.

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.

For details on the Provisioning Server's retry behavior to ensure reliability and accuracy of circuit provisioning, see Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

If you do not want the Provisioning Server to obtain card status prior to provisioning circuits, you can override the server's default behavior. Set the server environment variable CV_CARD_STATS to DISABLE to disable card status checking on circuit endpoints. For details, see Chapter 2 of the *NavisXtend Provisioning Server User's Guide*.

Prerequisites

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

Post-processing

None.

See Also

CvGetNumRetries.

3

Operational Functions

Operational functions perform provisioning operations on the network components.

CvaAddMember

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.

Member is a pointer to the CvObjectId structure that specifies the member object.

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, see Table 1-5 on page 1-10.

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_SmdsNetwideGroupAdddress 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.

CvaAddObject

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 *Arg*s **);**

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 (see 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, see Table 1-5 on page 1-10.

To create a card or PPort, use a Modify function (**CvaModifyObject** or **CvsModifyObject**). The NavisCore 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, NavisCore automatically populates the card with all necessary physical ports. Use **Modify** to change the PPort specifications. In the case of channelized cards, once the card has been configured, Provisioning Server automatically populates the card with all necessary channels. Use **Modify** 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 a member from an object list. Use this function to remove an address from a screen or netwide group address or to unbind an MLFRMember LPort from a MLFRBundle LPort. Upon completion of the function, the address or LPort 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.

Member is a pointer to the CvObjectId structure that specifies the member object.

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, see Table 1-5 on page 1-10.

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.

When you specify the object CVT_LPort as the container object, the container must be an MLFRBundle LPort and the member to be removed must be an MLFRMember LPort.

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, see Table 1-5 on page 1-10.

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**.

CvaGetDiagObject

Purpose

Retrieves object diagnostic results in the network.

Syntax

CvReqId CvaGetDiagObject(
CvServerHandle *Server*,
**const CvObjectId ***Object*,*
CVCBH *Callbk*,
**void ***UserArg*,*
CvConstArgs *Args* **);**

Description

CvaUpdateDiagObject is an asynchronous function that retreives object diagnostic results in the network.

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

Object* is a pointer to the **CvObjectId structure that specifies the object to be diagnosed.

Callbk 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 diagnosed. 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 (see the *NavisXtend Provisioning Server Object Attribute Definitions*).

Notes

For a list of object types that you can specify with this function, see Table 1-5 on page 1-10.

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

CvsGetDiagObject.

CvaGetList

Purpose

Retrieves the values of specific attributes from the database.

Syntax

CvReqId CvaGetList(
CvServerHandle *Server*,
**const CvObjectId ***Container*,*
CvObjectList *List*,
CvArgs *Args*)
OLCBH *Callback*,
void **Usrlist* **);**

Description

CvaGetList is an asynchronous function that retrieves the values of specific attributes from the database. Use this function whenever you wish to retrieve an object attribute that is represented as an ObjectList.

Server 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 object whose attributes are to be retrieved.

List should be set to NULL.

Args is the CvArgs class that specifies the attributes to be retrieved. Only one of these attributes can be an attribute whos representation is an ObjectList. The API will modify this argument list by removing the ArgId that specified the attribute containing the ObjectList. The ObjectList attribute is retrieved by the client using the **CvGetResponseArgs** function.

Callback is the callback function. Be sure to use the proper set of arguments in the callback function definition.

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

Notes

Use this function with the DefinedPath object.

Prerequisites

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

Make sure that the container and the list are valid.

Use **CvObjectListMake** to initialize **CvObjectList**. Use **CvObjectListAdd** and **CvObjectListSetNull** to add objects to the CvObjectList.

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.

Call **CvObjectListFree** to free memory.

See Also

CvsGetList

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 (see 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, see Table 1-5 on page 1-10.

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.

CvaGetOperInfoObject

Purpose

Retrieves the values of specific OperInfo attributes from the switch.

Syntax

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

Description

CvaGetOperInfoObject 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 objects 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 (see the attribute tables in *Provisioning Server Object Attribute Definitions*). Specify only attribute IDs with null values.

Notes

The valid object type for this function is CVT_Circuit

Prerequisites

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

Before you issue this function:

- Make sure the object ID is valid.
- 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, or issue a series of utility functions that create (**CvArgsMake**) and fill in (**CvArgsSetAttrType**) 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 **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

CvsGetOperInfoObject

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 addres, 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
Card	CardTca PPort

Table 3-1. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
Channel	LPort PerformanceMonitor ChanPerformanceMonitor
Customer	Circuit LPort
LPort	AssignedSvcSecScn Circuit LPort (only for listing MLFR Members on an MLFR Bundle) PMPCktRoot PMPSpvcRoot SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress Spvc SvcAddress SvcConfig SvcNetworkId SvcPrefix SvcSecScnActParam SvcUserPart VpciTable
Network	Customer NetCac ServiceName SmdsCountryCode SmdsNetwideGroupAddress SvcCUG SvcCUGMbrRule SvcSecScn Switch TrafficDesc Trunk VPN
PMPCktRoot	PMPCktLeaf
PMPSpvcRoot	PMPSpvcLeaf
PPort	Aps (1-port OC-12c/STM-4 and 4-port OC-3/STM-1 cards only) Channel ChannelPerformanceMonitor LPort PerformanceMonitor PFdl (8-port ATM T1 card only) PPortTca TrafficShaper
ServiceName	Circuit

Table 3-1. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
SmdsGroupScreen	SmdsAlienGroupAddress SmdsSwitchGroupAddress
SmdsIndividualScreen	SmdsAlienIndividualAddress SmdsLocalIndividualAddress
SmdsNetwideGroupAddress	SmdsLocalIndividualAddress SmdsSwitchGroupAddress
SvcCUG	SvcCUGMbr
SvcCUGMbrRule	SvcCUGMbr
Switch	Card PnniNode RefTimeServer SmdsAddressPrefix SmdsAlienGroupAddress SmdsAlienIndividualAddress SmdsSwitchGroupAddress SvcNodePrefix
Trunk	Circuit
VPN	Circuit LPort

Prerequisites

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

Make sure that the container and the list 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

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 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
Card	Aps CardTca Channel Circuit LPort Performance Monitor PFdl PMPSpvcRoot PPort Spvc SmdsAlienGroupAddress SmdsAlienIndividualAddress SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress SmdsSwitchGroupAddress SvcConfig SvcNodePrefix SvcUserPart Trunk
Channel	Circuit LPort Performance Monitor Trunk
Customer	Circuit LPort

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

Parent Object Type	Child Object Types
LPort	AssignedSvcSecScn Circuit PMPCktRoot PMPSpvcRoot SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress Spvc SvcAddress SvcConfig SvcNetworkId SvcPrefix SvcSecScnActParam SvcUserPart VpciTable Trunk
Network	Circuit Customer NetCac PMPCktLeaf PMPCktRoot PMPSpvcRoot ServiceName SmdsAddressPrefix SmdsCountryCode SmdsLocalIndividualAddress SmdsNetwideGroupAddress SvcCUG SvcCUGMbrRule SvcSecScn Switch TrafficDesc Trunk VPN
PMPCktRoot	PMPCktLeaf
PMPSpvcRoot	PMPSpvcLeaf
PPort	Aps (1-port OC-12c/STM-4 and 4-port OC-3/STM-1 cards only) Channel Circuit LPort PerformanceMonitor PFdl (8-port ATM T1 card only)| PMPCktRoot PMPSpvcRoot PPortTca Spvc TrafficShaper Trunk

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

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

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**.

CvaModifyList

Purpose

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

Syntax

CvReqId CvaModifyListObject(
CvServerHandle *Server*,
**const CvObjectId ***Container*,*
CvConstObjectList *List*,
CvConstArgs *Args*,
OLCBH *Callback*,
void ***Usrlist* **);

Description

CvaModifyList is an asynchronous function that modifies the specific attributes of an object in the database and (optionally) in the switch. Use this function to modify an object's attribute that is specified as an ObjectList.

Server 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 contains the ObjectList.

List specifies the new ObjectList.

Args is the argument list associated with the container object. Each member of the list has its own CvArgs structure.

Callback is the callback function.

Usrlist 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 modify with this function, see Table 1-5 on page 1-10.

Prerequisites

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

Make sure that the container and the list are valid.

Use the function **CvObjectListMake** to initialize the CvObjectList. Use CvObjectListAdd and CvObjectListSetNull to add objects to the CvObjectList.

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

CvsModifyList.

CvaModifyObject

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 (see 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, see Table 1-5 on page 1-10.

Use this function to create a card or PPort. The NavisCore 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, NavisCore automatically populates the card with all necessary physical ports. Use this function to change the PPort specifications. In the case of channelized cards, once the card has been configured, Provisioning Server 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.

CvaStartDiagObject

Purpose

Starts diagnostics on an object in the network.

Syntax

CvReqId CvaStartDiagObject(
CvServerHandle *Server*,
**const CvObjectId ***Object*,*
CVCBH *Callbk*,
void *UserArg*,
CvConstArgs *Args* **);**

Description

CvaStartDiagObject is an asynchronous function that starts diagnostics on an object in the network.

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

Object is a pointer to the **CvObjectId** structure that specifies the object to be diagnosed.

Callbk 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 diagnosed. 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 (see the *NavisXtend Provisioning Server Object Attribute Definitions*).

Notes

For a list of object types that you can specify with this function, see Table 1-5 on page 1-10.

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

Prerequisites

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

Initialize the *Status* argument with **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.

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

CvsStartDiagObject.

CvaStopDiagObject

Purpose

Stops diagnostics on an object in the network.

Syntax

CvReqId CvaStopDiagObject(
CvServerHandle *Server*,
**const CvObjectId ***Object*,*
CVCBH *Callbk*,
void ***UserArg*,);

Description

CvaStopDiagObject is an asynchronous function that stops diagnostics on an object in the network.

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

Object* is a pointer to the **CvObjectId structure that specifies the object to be diagnosed.

Callbk 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 diagnosed. 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 (see the *NavisXtend Provisioning Server Object Attribute Definitions*).

Notes

For a list of object types that you can specify with this function, see Table 1-5 on page 1-10.

Prerequisites

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

CvsStopDiagObject.

CvaUpdateDiagObject

Purpose

Updates diagnostic operations on an object in the network.

Syntax

CvReqId CvaUpdateDiagObject(
CvServerHandle *Server*,
**const CvObjectId ***Object*,*
CVCBH *Callbk*,
**void ***UserArg*,*
CvConstArgs *Args* **);**

Description

CvaUpdateDiagObject is an asynchronous function that modifies the object diagnostics in the network.

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

Object* is a pointer to the **CvObjectId structure that specifies the object to be diagnosed.

Callbk 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 diagnosed. 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 (see the *NavisXtend Provisioning Server Object Attribute Definitions*).

Notes

For a list of object types that you can specify with this function, see Table 1-5 on page 1-10.

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

CvsUpdateDiagObject.

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 a member to an object list. Use this function to add an address to a screen or netwide group address or to bind an MLFRMember LPort to a MLFRBundle LPort. Upon completion of the function, the address or LPort 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.

Member is a pointer to the CvObjectId structure that specifies the member object.

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, see Table 1-5 on page 1-10.

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.

When you specify the object CVT_LPort as the container object, the container must be an MLFRBundle LPort and the member to be added must be an MLFRMember LPort.

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 (see 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, see Table 1-5 on page 1-10.

To create a card or PPort, use a Modify function (**CvaModifyObject** or **CvsModifyObject**). The NavisCore 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, NavisCore automatically populates the card with all necessary physical ports. Use the Modify function to change the PPort specifications. In the case of channelized cards, once the card has been configured, Provisioning Server 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**.

CvsDeleteMember

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 a member from an object list. Use this function to remove an address from a screen or netwide group address or to unbind an MLFRMember LPort from a MLFRBundle LPort. Upon completion of the function, the address or LPort 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.

Member is a pointer to the CvObjectId structure that specifies the member object.

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, see Table 1-5 on page 1-10.

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.

When you specify the object CVT_LPort as the container object, the container must be an MLFRBundle LPort and the member to be removed must be an MLFRMember LPort.

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.

CvsDeleteObject

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, see Table 1-5 on page 1-10.

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**.

CvsGetDiagObject

Purpose

Retrieves object diagnostic results in the network.

Syntax

CvArgs CvsGetDiagObject(
CvServerHandle *Server*,
**const CvObjectId ***Object*,*
CvStatus **Status*,
CvConstArgs *Args* **);**

Description

CvsGetDiagObject is a synchronous function that retreives object diagnostic results in the network.

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

Object* is a pointer to the **CvObjectId structure that specifies the object to be diagnosed.

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

Args is a constant **CvArgs** structure that specifies the attributes to be diagnosed. 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 (see the *NavisXtend Provisioning Server Object Attribute Definitions*). Specify only attributes with null values.

Notes

For a list of object types that you can specify with this function, see Table 1-5 on page 1-10.

Prerequisites

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

You must initialize the *Status* argument with **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 **CvaGetDiagObject**.

CvsGetList

Purpose

Retrieves the values of specific attributes from the database.

Syntax

void CvsGetList(
CvServerHandle *Server*,
**const CvObjectId ***Container*,*
CvObjectList *List*,
CvArgs *Args*,
CvStatus **Status* **);**

Description

CvsGetList is a synchronous function that retrieves the values of specific attributes from the database. Use this function whenever you wish to retrieve an object attribute that is represented as an ObjectList.

Server 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 object whose attributes are to be retrieved.

List should be set to NULL.

Args is the CvArgs class that specifies the attributes to be retrieved. Only one of these attributes can be an attribute whose representation is an ObjectList. The API will modify this argument list by removing the ArgId that specified the attribute containing the ObjectList. The ObjectList attribute is retrieved by the client using the CvGetResponseArgs function.

Status, an output function, is a pointer to the CvStatus variable that contains the status returned by the function. Initialize this argument to cveNoError. This variable reports any error that occurs in the actual ObjectList.

Notes

Use this function with the DefinedPath object.

Prerequisites

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

Make sure that the container and the list are valid.

Use the function CvObjectListMake( ) to initialize the CvObjectList. Use CvObjectListAdd and CvObjectListSetNull to add objects to the CvObjectList.

Post-processing

If the *Status* variable has a value other than cveNoError, issue a **CvGetResponseObjList**(CvServerHandle *Handle*) function to get the CvObjectList value that was returned from the server. This list will be altered according to the error that occurs. Check the status of each object in the list, and each argument of each object in the list, to determine where the error occurred.

Call **CvObjectListFree** to free memory.

See Also

CvaGetList.

CvsGetObject

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 (see 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, see Table 1-5 on page 1-10.

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**.

CvsGetOperInfoObject

Purpose

Retrieves the values of specific OperInfo attributes from the switch.

Syntax

CvArgs CvsGetOperInfoObject (
CvServerHandle *Handle*,
const CvObjectId **ObjId*,
CvStatus **Status*,
CvConstArgs *Args* **);**

Description

CvsGetOperInfoObject is an synchronous function that retrieves the values of specific attributes 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 objects 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..

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 (see the attribute tables in *Provisioning Server Object Attribute Definitions*). Specify only attribute IDs with null values.

Notes

The valid object type for this function is CVT_Circuit.

Prerequisites

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

Before you issue this function, make sure the object ID is 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

CvaGetOperInfoObject

CvsListAllContainedObjects

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 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
Card	CardTca PPort
Channel	LPort PerformanceMonitor
Customer	Circuit LPort
LPort	AssignedSvcSecScn Circuit PMPCktRoot PMPSpvcRoot SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress Spvc SvcAddress SvcConfig SvcNetworkId SvcPrefix SvcSecScnActParam SvcUserPart VpciTable
Network	Customer NetCac ServiceName SmdsCountryCode SmdsNetwideGroupAddress SvcCUG SvcCUGMbrRule SvcSecScn Switch TrafficDesc Trunk VPN
PMPCktRoot	PMPCktLeaf
PMPSpvcRoot	PMPSpvcLeaf
PPort	Aps (1-port OC-12c/STM-4 and OC-3c/STM-1 cards) Channel LPort PerformanceMonitor PFdl (8-port ATM T1 card only) PPortTca TrafficShaper
ServiceName	Circuit

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

Parent Object Type	Child Object Types
SmdsGroupScreen	SmdsAlienGroupAddress SmdsSwitchGroupAddress
SmdsIndividualScreen	SmdsAlienIndividualAddress SmdsLocalIndividualAddress
SmdsNetwideGroupAddress	SmdsLocalIndividualAddress SmdsSwitchGroupAddress
SvcCUG	SvcCUGMbr
SvcCUGMbrRule	SvcCUGMbr
Switch	Card PnniNode RefTimeServer SmdsAddressPrefix SmdsAlienGroupAddress SmdsAlienIndividualAddress SmdsSwitchGroupAddress SvcNodePrefix
Trunk	Circuit
VPN	Circuit LPort

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 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
Card	Aps CardTca Channel Circuit LPort Performance Monitor PFdl PMPSpvcRoot PPort Spvc SmdsAlienGroupAddress SmdsAlienIndividualAddress SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress SmdsSwitchGroupAddress SvcConfig SvcNodePrefix SvcUserPart Trunk
Channel	Circuit LPort Performance Monitor Trunk
Customer	Circuit LPort
LPort	AssignedSvcSecScn Circuit PMPCktRoot PMPSpvcRoot SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress Spvc SvcAddress SvcConfig SvcNetworkId SvcPrefix SvcSecScnActParam SvcUserPart VpciTable Trunk

Table 3-4. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
Network	Circuit Customer NetCac PMPCktLeaf PMPCktRoot PMPSpvcRoot ServiceName SmdsAddressPrefix SmdsCountryCode SmdsLocalIndividualAddress SmdsNetwideGroupAddress SvcCUG SvcCUGMbrRule SvcSecScn Switch TrafficDesc Trunk VPN
PMPCktRoot	PMPCktLeaf
PMPSpvcRoot	PMPSpvcLeaf
PPort	Aps Channel Circuit LPort PerformanceMonitor PFdl (8-port ATM T1 card only)| PMPCktRoot PMPSpvcRoot PPortTca Spvc Trunk TrafficShaper
SmdsGroupScreen	SmdsAlienGroupAddress SmdsSwitchGroupAddress
SmdsIndividualScreen	SmdsAlienIndividualAddress SmdsLocalIndividualAddress
SmdsNetwideGroupAddress	SmdsLocalIndividualAddress SmdsSwitchGroupAddress
SvcCUG	SvcCUGMbr
SvcCUGMbrRule	SvcCUGMbr

Table 3-4. Valid Parent and Child Object Types (Continued)

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

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**.

CvsModifyListObject

Purpose

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

Syntax

void CvsModifyListObject(
CvServerHandle *Server*,
**const CvObjectId ***Container*,*
CvConstObjectList *List*,
CvConstArgs *Args*,
**CvStatus ***Status*);*

Description

CvsModifyListObject is a synchronous function that modifies the specific attributes of an object in the database and (optionally) in the switch. Use this function to modify an object's attribute that is specified as an ObjectList.

Server 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 contains the ObjectList to be modified.

List is the new ObjectList.

Args is the argument list associated with the container object. Each member of the list has its own CvArgs structure.

Status, an output parameter, is a pointer to the CvStatus variable that contains the status returned by the function. Initialize this argument to **cveNoError**. Status reports any error that occurs in the actual ObjectList.

Notes

Use this function with the DefinedPath object.

Prerequisites

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

Make sure that the container and the list are valid.

Use the function **CvObjectListMake** to initialize the CvObjectList. Use **CvObjectListAdd** and **CvObjectListSetNull** to add objects to the CvObjectList.

Post-processing

If the *Status* variable has a value other than **cveNoError**, issue a **CvGetResponseObjList**(CvServerHandle *Handle*) function to get the CvObjectList value that was returned from the server. This list will be altered according to the error that occurs. Check the status of each object in the list, and each argument of each object in the list, to determine where the error occurred.

See Also

CvaModifyList.

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 (see 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, see Table 1-5 on page 1-10.

Use this function to create a card or PPort. The NavisCore 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, NavisCore automatically populates the card with all necessary physical ports. Use this function to change the PPort specifications. In the case of channelized cards, once the card has been configured, Provisioning Server 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**.

CvsStartDiagObject

Purpose

Starts diagnostics on an object in the network.

Syntax

void CvsStartDiagObject(
CvServerHandle *Server*,
**const CvObjectId ***Object*,*
**CvStatus ***Status*,*
CvConstArgs *Args* **);**

Description

CvsStartDiagObject is a synchronous function that starts diagnostics on an object in the network.

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

Object* is a pointer to the **CvObjectId structure that specifies the object to be diagnosed.

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

Args is a constant **CvArgs** structure that specifies the attributes to be diagnosed. 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 (see the *NavisXtend Provisioning Server Object Attribute Definitions*).

Notes

For a list of object types that you can specify with this function, see Table 1-5 on page 1-10.

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

Prerequisites

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

Initialize the *Status* argument with **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 omit.

See Also

CvaStartDiagObject.

CvsStopDiagObject

Purpose

Stops diagnostics on an object in the network.

Syntax

void CvsStopDiagObject(
CvServerHandle *Server*,
**const CvObjectId ***Object*,*
CvStatus **Status*, **);**

Description

CvsStopDiagObject is a synchronous function that stops diagnostics on an object in the network.

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

Object* is a pointer to the **CvObjectId structure that specifies the object to be diagnosed.

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

Notes

For a list of object types that you can specify with this function, see Table 1-5 on page 1-10.

Prerequisites

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

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

See Also

CvaStopDiagObject.

CvsUpdateDiagObject

Purpose

Updates diagnostic operations on an object in the network.

Syntax

void CvsUpdateDiagObject(
CvServerHandle *Server*,
**const CvObjectId ***Object*,*
**CvStatus ***Status*,*
CvConstArgs *Args* **);**

Description

CvsUpdateDiagObject is a synchronous function that modifies the object diagnostics in the network.

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

Object* is a pointer to the **CvObjectId structure that specifies the object to be diagnosed.

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

Args is a constant **CvArgs** structure that specifies the attributes to be diagnosed. 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 (see the *NavisXtend Provisioning Server Object Attribute Definitions*).

Notes

For a list of object types that you can specify with this function, see Table 1-5 on page 1-10.

Prerequisites

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

Initialize the Status argument with **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 **CvaUpdateDiagObject**.

4

Select Loop Functions

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

CVCBH

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.

CvGetSelectInfo

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**.

CvProcessEvents

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**.

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**.

OLCBH

Purpose

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

Syntax

typedef void (*OLCBH) (
CvObjectId *Id*,
CvStatus *Status*,
CvReqId *RequestId*,
void * *UserArg*,
CvConstObjectList *List*,
CvConstArgs *Args* **);**

Description

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

Id 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.

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

Args is a constant CvArgs structure that contains the objects and 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.

5

Utility Functions

Utility functions build argument lists, handle initialization, and manage storage.

CvAddArgumentByName

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.

CvArgsCombine

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**.

CvArgsEqual

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 index to the first argument with an error. The index value ranges from 1 to the number of arguments.

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

Notes

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.

CvArgsFree

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**.

CvArgsGetIpAddr

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**.

CvArgsGetStatus

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**.

CvArgsIdAt

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**.

CvArgsMakeIds

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 (see 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**.

CvArgsMakeVals

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 (see 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.

CvArgsRemove

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.

CvArgsSetInt

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**.

CvArgsSetIpAddress

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**.

CvArgsSetNull

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**.

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**.

CvArgsToString

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**.

CvCreateApsId

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.

CvCreateCardTcaId

Purpose

Fills in a CvObjectId structure with a card's TCA values.

Syntax

CvObjectId CvCreateCardTcaId(
CvObjectId *CardID* **);**

Description

CvCreateCardTcaId fills in the CvObjectId structure with a card's TCA values. 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 CardTca object. This structure is built by **CvCreateCardId**.

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.

CvCreateChannelId

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.

CvCreateCircuitDefinedPathId

Purpose

Fills in a CvObjectId structure with a value for a circuit DefinedPath.

Syntax

CvObjectId CvCreateCircuitDefinedPathId(
CvObjectId *CircuitID* **);**

Description

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

CircuitID is the CvObjectId structure that specifies the circuit that contains the DefinedPath object. This structure is built by **CvCreateDlciId** or **CvCreateAtmEndpointId**.

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.

CvCreateDlciId

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.

CvCreateFdlId

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**.

When the LPort is a MLFRBundle LPort, specify the card that contains the bundle LPort. In this case, the structure is built by **CvCreateCardId**. Set the PPort ID to zero.

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.

CvCreateNetworkId

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.

CvCreatePerformanceMonitorId

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.

CvCreatePMPSpvcRootId

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.

CvCreatePnniNodeId

Purpose

Fills in a CvObjectId structure with values for an object's PnniNodeId.

Syntax

CvObjectId CvCreatePnniNodeId(
CvObjectId *SwitchID***,**
const char ***PgString **);

Description

CvCreatePnniNodeId fills in the CvObjectId structure with values for an object's PnniNodeId. 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 PNNINode object. This structure is built by **CvCreateSwitchId**.

PgString is a pointer to a PNNI Peer Group string consisting of a Peer Group Level and a Peer Group ID, separated by a dash (-). The Peer Group Level value is a decimal number representing the number of bits allowed in the Peer Group ID field (range 0 - 104). The Peer Group ID value is a hexidecimal number up to 28 characters (limited to the characters 0 - f).

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.

CvCreatePPortId

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.

CvCreatePPortTcaId

Purpose

Fills in a CvObjectId structure with a PPort's TCA values.

Syntax

CvObjectId CvCreatePPortTcaId(
CvObjectId *PPortID* **);**

Description

CvCreatePPortTcaId fills in the CvObjectId structure with a PPort's TCA values. 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 PPortTca object. 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.

CvCreateRefTimeServer

Purpose

Fills in a CvObjectId structure with a value for an object's reference time server.

Syntax

CvObjectId CvCreateRefTimeServer(
CvObjectId *SwitchID***,**
CvIpAddr *ServerIpAddr* **);**

Description

CvCreateRefTimeServer fills in the CvObjectId structure with a value for an object's reference time server IP address. 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 reference time server. This structure is built by **CvCreateSwitchId**.

IpAddr is the IP address of the reference time server.

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.

The function returns 0 if the parent object does not support the specified field.

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.

CvCreateSmdsAlienGroupAddrId

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.

CvCreateSmdsAlienIndivAddrId

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.

CvCreateSmdsCountryCodeId

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.

CvCreateSmdsIndivScreenId

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.

CvCreateSmdsSsiIndivAddrAddrId

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 (see 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.

CvCreateSvcCUGId

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.

CvCreateSvcNetworkId

Purpose

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

Syntax

CvObjectId CvCreateSvcNetworkId(
CvObjectId *LportID*,
const char ***_Addr_ **);

Description

CvCreateSvcNetworkId fills in the CvObjectId structure with a value for an object's SvcNetworkId. 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 SvcNetworkId. 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 (see 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.

The function returns 0 if the parent object does not support the specified field.

Prerequisites

None.

Post-processing

None.

See Also

CvCreate*ObjectType***Id** functions.

CvCreateSvcNodePrefixId

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 (see 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.

CvCreateSvcPrefixId

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 (see 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.

CvCreateSvcSecScnId

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.

CvCreateSvcUserPartId

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 (see 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.

CvCreateSwitchId

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.

CvCreateTrafficDescId

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.

CvCreateTrunkId

Purpose

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

Syntax

CvObjectId CvCreateTrunkId(
CvObjectId *NetworkID*,
const char ***_TrunkName_ **);

Description

CvCreateTrunkId fills in the CvObjectId structure with a value for an object's trunk 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 trunk. This structure is built by **CvCreateNetworkId** or **CvCreateNetworkIdFromString**.

TrunkName is a pointer to the trunk 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.

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.

CvCreateVpciTableId

Purpose

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

Syntax

CvObjectId CvCreateVpciTableId(
CvObjectId *LportID*,
const int *VpciNumber* **);**

Description

CvCreateVpciTableId fills in the CvObjectId structure with a value for an object's VpciTable 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 VpciTable. This structure is built by **CvCreateLPortId**.

VpciNumber is the Vpci number, in the range 0 - 65535.

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.

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.

CvGetAddress

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**.

CvGetCardNumber

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.

CvGetDlciNumber

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**.

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.

CvGetLPortNumber

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.

CvGetName2

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.

CvGetNumRetries

Purpose

Obtains the current retry setting for card status checks preceding circuit provisioning requests.

Syntax

int CvGetNumRetries(
CvServerHandle *Handle* **);**

Description

CvGetNumRetries obtains the current retry setting for card status checks preceding circuit provisioning requests.

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

Notes

For details on the Provisioning Server's retry behavior to ensure reliability and accuracy of circuit provisioning, see Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

If you do not want the Provisioning Server to obtain card status prior to provisioning circuits, you can override the server's default behavior. Set the server environment variable CV_CARD_STATS to DISABLE to disable card status checking on circuit endpoints. For details, see Chapter 2 of the *NavisXtend Provisioning Server User's Guide*.

Prerequisites

None.

Post-processing

None.

See Also

CvSetNumRetries.

CvGetObjectType

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**.

CvGetPMPSpvcLeafId

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.

CvGetPnniNodeId

Purpose

Reads the PnniNodeId out of a CvObjectId structure.

Syntax

const char *CvGetPnniNodeId(
CvObjectId *ObjId* **);**

Description

CvGetPnniNodeId reads the PNNI Peer Group string 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.

The PNNI Peer Group string consists of a Peer Group Level and a Peer Group ID, separated by a dash (-). The Peer Group Level value is a decimal number representing the number of bits allowed in the Peer Group ID field (range 0 - 104). The Peer Group ID value is a hexidecimal number up to 28 characters (limited to the characters 0 - f).

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.

CvGetPrefix

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.

CvGetRefTimeServerIpAddr

Purpose

Reads the reference time server IP address out of a CvObjectId structure.

Syntax

CvIpAddr CvGetRefTimeServerIpAddr(
CvObjectId *ObjId* **);**

Description

CvGetRefTimeServerIpAddr reads the reference time server 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.

CvGetResponseArgs

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.

CvGetResponseObjList

Purpose

Reads the ObjectList from the last synchronous response sent by the server.

Syntax

CvConstObjectList CvGetResponseObjList(
CvServerHandle *Handle*);

Description

CvGetResponseObjList reads the ObjectList from the last synchronous response sent by the server. Since the ObjectList sent to the server in an add or modify request is const, the server needs a way of letting the client know of any part of the ObjectList has been changed. The server adds to the list any attributes that are required that have been omitted, and it sets the status of a list member or attribute if an error occurred. The server returns this new list to the client for user access.

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**.

Issue a synchronous add, modify, or insert request.

Post-processing

None.

See Also

CvArgsErrorIndex.

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.

CvGetSvcAddress

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
CVT_SvcNetworkId

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.

CvGetTrafficShaperId

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.

CvGetVpciNumber

Purpose

Reads the Vpci number out of a CvObjectId structure.

Syntax

int CvGetVpciNumber(
CvObjectId *ObjId* **);**

Description

CvGetVpciNumber reads the Management DLCI 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.

CvGetVpiNumber

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**.

CvObjectIdToString

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**.

CvObjectListAdd

Purpose

Adds a CvObjectId/CvArgs to an ObjectList.

Syntax

void CvObjectListAdd(
CvObjectList *List,*
CVObjectId *Id,*
CvArgs *Args)***;**

Description

.CvObjectListAdd adds a CvObjectId/CvArgs to an ObjectList. If the ObjectId already exists, this function adds the CvArgs to the given object. If CvArgs already exists, it will be written over.

List is the ObjectList to which the CvObjectId/CvArgs is added.

Id is the CvObjectId symbol for the object to be read.

Prerequisites

None.

Post-processing

None.

See Also

CvArgs.

CvObjectListCombine

Purpose

Returns the union of two ObjectLists.

Syntax

CvObjectList CvObjectListCombine(
CvConstObjectList *eva1***,**
CvConstObjectList *eva2* **);**

Description

CvObjectListCombine retrurns the union of two ObjectLists. The result is a new ObjectList that combines both ObjectLists. When an argument exists in both ObjectLists, the value from the second list is used.

eva1 and *eva2* are the CvObjectList structures to be combined.

Prerequisites

Make sure that the ObjectLists represented by *eva1* and *eva2* are valid.

Post-processing

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

See Also

CvObjectListRemove and **CvObjectListSelect**.

CvObjectListCount

Purpose

Retrieves the number of objects in the ObjectList.

Syntax

int CvObjectListCount(
CvObjectList *List* **);**

Description

CvObjectListCount retrieves the number of objects in the ObjectList.

List is a constant CvObjectList structure that contains the list of objects. The API does not modify or delete this argument.

Prerequisites

Make sure that the ObjectList represented by *List* is valid.

Post-processing

None.

See Also

CvObjectListIdAt.

CvObjectListErrorIndex

Purpose

Determines if any object in an ObjectList has an error status.

Syntax

int CvObjectListErrorIndex(
CvConstObjectList *List* **);**

Description

CvObjectListErrorIndex determines if any object in an ObjectList has an error status. If an error status exists, the function returns the one-based index to the first object with an error. If no error status exists, the function returns a 0.

Notes

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 **CvObjectListStatusAt**.

See Also

CvObjectListStatusAt.

CvObjectListFree

Purpose

Frees the memory.

Syntax

void CvObjectListFree(
CvObjectList *List* **);**

Description

CvObjectListFree deletes an opaque pointer to a CvObjectList structure. It frees the memory used by an ObjectList that was either created explicitly or implicitly by another function.

List is a pointer to a CvObjectList structure that contains the ObjectList. This pointer is no longer valid if **CvObjectListFree** is successful.

Prerequisites

None.

Post-processing

None.

See Also

CvObjectListMake.

CvObjectListGetArgs

Purpose

Returns the Args, given the CvObjectId.

Syntax

CvArgs CvObjectListGetArgs(
CvObjectList *List***,**
CvObjectId *Id* **);**

Description

CvObjectListGetArgs returns the Args, given the CvObjectId.

Id is the CvObjectId symbol for the object to be read.

Notes

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

Prerequisites

None.

Post-processing

None.

CvObjectListGetStatus

Purpose

Retrieves the error status code of an object in an ObjectList.

Syntax

CvStatus CvObjectListGetStatus(
CvConstObjectList *List***,**
CvObjectId *Id* **);**

Description

CvObjectListGetStatus retrieves the error status code of an object in an ObjectList.

List is a constant CvObjectList structure that contains the ObjectList for which status is reported. The API does not modify or delete this argument.

Id is the CvObjectId symbol for the object in the ObjectList. If *Id* is not in the argument list, the status -1 is returned.

Prerequisites

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

1. Obtain the ObjectList (CvObjectList structure) by appropriate means, such as with **CvObjectListMake** or **CvaGetObject**.
2. Issue **CvObjectListErrorIndex** to determine if any object in the ObjectList has an error status. To identify a retrieved object, call CvObjectListIdAt.
3. Issue this function to retrieve the error status code of the object.

Post-processing

None.

See Also

CvObjectListStatusAt.

CvObjectListIdAt

Purpose

Retrieves the ObjectId at a specified position in the ObjectList.

Syntax

const CvObjectId CvObjectListIdAt(
CvObjectList *List*, *Int I* **);**

Description

CvObjectListIdAt retrieves the ObjectId at the specified position in the ObjectList. If the index is invalid, the function returns 0.

List is the ObjectList that contains the ObjectId.

Int I is the index into the ObjectList.

Notes

Use this function to read values returned by queries, such as **CvaGetList**.

Prerequisites

Make sure that the ObjectList represented by *List* is valid.

To count the number of objects in the ObjectList, issue **CvObjectListCount**.

Post-processing

None.

See Also

CvObjectListCount and **CvObjectListStatusAt**.

CvObjectListMake

Purpose

Initializes a CvObjectList.

Syntax

CvObjectList CvObjectListMake( )

Description

CvObjectListMake initializes a CvObjectList. It returns an opaque pointer to a CvObjectList structure.

Prerequisites

None.

Post-processing

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

1. Issue this function to create the ObjectList.
2. Use **CvObjectListAdd** to fill in the CvObjectList structure.
3. Issue one or more operational functions (such as **CvaInsertObject**).
4. Issue **CvObjectListFree** to free the memory used by the argument list.

See Also

None.

CvObjectListPrint

Purpose

Prints the contents of CvObjectList to a file.

Syntax

void CvObjectListPrint(
FILE ****f*,**
CvConstObjectList *List***);**

Description

CvObjectListPrint prints the contents of CvObjectList to a file. No newline characters are inserted into the file.

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

List is a constant CvObjectList structure that holds the objects to be printed to a file. The API does not modify or delete this argument.

Prerequisites

Make sure that the ObjectList represented by *List* is valid.

Post-processing

None.

See Also

CvObjectListToString.

CvObjectListRemove

Purpose

Removes an object from the ObjectList.

Syntax

void CvObjectListRemove(
CvObjectList *List***,**
CvObjectId *Id* **);**

Description

CvObjectListRemove removes an object from the ObjectList, given the CvObjectId.

Id is the CvObjectListId symbol that identifies the object.

Prerequisites

Make sure that the CvObjectId is valid.

Post-processing

None.

See Also

CvObjectListSelect.

CvObjectListSelect

Purpose

Returns the intersection of two ObjectLists.

Syntax

CvObjectList CvObjectListSelect(
CvConstObjectList *eva1***,**
CvConstObjectList *eva2* **);**

Description

CvObjectListSelect returns the intersection of two ObjectLists. The result is a new ObjectList containing objects that exist in both ObjectLists. When an object exists in both lists, the value from the first list is used.

eva1 and *eva2* are the CvObjectList structures to be added.

Prerequisites

Make sure that the ObjectLists represented by *eva1* and *eva2* are valid.

Post-processing

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

See Also

CvObjectListCombine.

CvObjectListSetNull

Purpose

Sets the CvArgs structure to NULL when adding an ob ject with a given CvObjectId.

Syntax

void CvObjectListSetNull(
CvObjectList *List*,
CvObjectId *Id* **);**

Description

CvObjectListSetNull sets the CvArgs field of an object to NULL. If the object is already on the list, CvArgs is nulled out. If the object is not already on the list, it is added with a NULL CvArgs. Use this function to add an object with no CvArgs value to the ObjectList in the event that each object does not have its own set of attributes.

List is the ObjectList to be set.

Id is the ObjectListId symbol for the ObjectList to be set.

Prerequisites

Use **CvObjectListMake** to create the ObjectList.

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 CvObjectList structure.
2. Issue **CvObjectListFree** to free the memory used by the ObjectList.

See Also

CvObjectListStatusAt

Purpose

Retrieves the error status code of an object at a specified location in the ObjectList.

Syntax

CvStatus CvObjectListStatusAt(
CvConstObjectList *List***,** *Int N* **);**

Description

CvObjectListStatusAt retrieves the error status code of an object at the specified nth position in the ObjectList.

List specifies the ObjectList.

Int N is the zero-based index of the object in the ObjectList. If *N* is out of range of the ObjectList, the status -1 is returned.

Prerequisites

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

1. Obtain the ObjectList (CvObjectList structure) by appropriate means, such as with **CvObjectListMake** or **CvaGetList**.
2. Issue **CvObjectListErrorIndex** to determine if any object in the ObjectList has an error status.
3. Issue this function to retrieve the error status code of the object.

Post-processing

None.

See Also

CvObjectListGetStatus.

CvObjectListSubtract

Purpose

Returns a list of objects that are in one ObjectList but not in another ObjectList.

Syntax

CvObjectList CvObjectListSubtract(
CvConstObjectList *eva1***,**
CvConstObjectList *eva2* **);**

Description

CvObjectListSubtract subtracts one ObjectList (eva2) from another (eva1) and returns a list of objects that are in eva1 and not in eva2.

eva1 is the ObjectList to be subtracted from, and *eva2* is the ObjectList that is subtracted.

Prerequisites

Make sure that the ObjectLists represented by *eva1* and *eva2* are valid.

Post-processing

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

See Also

CvObjectListRemove, CvObjectListCombine, and **CvObjectListSelect**.

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**.

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**.

CvSetNumRetries

Purpose

Sets the number of retries to check card status preceding circuit provisioning requests.

Syntax

CvStatus CvSetNumRetries(
CvServerHandle *Handle***,**
int *NumRetries* **);**

Description

CvSetNumRetries sets the number of retries to check card status preceding circuit provisioning requests.

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

NumRetries is the number of retries. The value applies to requests at either endpoint: when a retry is sent to obtain the card status of one endpoint, the number of retries decrements for either endpoint. Valid values 0 - 5.

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.

For details on the Provisioning Server's retry behavior to ensure reliability and accuracy of circuit provisioning, see Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

If you do not want the Provisioning Server to obtain card status prior to provisioning circuits, you can override the server's default behavior. Set the server environment variable CV_CARD_STATS to DISABLE to disable card status checking on circuit endpoints. For details, see Chapter 2 of the *NavisXtend Provisioning Server User's Guide*.

Prerequisites

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

Post-processing

None.

See Also

CvGetNumRetries.

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**.

Section 3
C++ Functions

Chapter 6	CvClient
Chapter 7	CvClient::Callback
Chapter 8	CvClient::ObjectId
Chapter 9	CvClient::Args
Chapter 10	CvClient::ObjectList
Chapter 11	CvClient::IpAddr
Chapter 12	CvSVCAddress

6

CvClient

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 (see Chapter 7, “CvClient::Callback”).

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 (see Chapter 8, “CvClient::ObjectId”).

CvClient::Args — A class that represents the argument list and contains member functions to add values to and read values from the argument list (see Chapter 9, “CvClient::Args”).

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::ObjectList — used to build a list of object identifier (CvClient::ObjectId) argument list (CvClient::Args) pairs. The class has member functions to set and read specific objectId-Args pairs to and from the ObjectList. (see Chapter 10, “CvClient::ObjectList”).

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 (see Chapter 11, “CvClient::IpAddr”).

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 describes **CvClient**’s session control and operational member functions.

Session Control

This section describes how to use **CvClient** to manage a session with the Provisioning Server.

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 reference to the name of the user starting the session. This value is the same as that required by NavisCore.

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

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 (see Chapter 2 of the *NavisXtend Provisioning Server User's Guide* for details). When issuing this function, pass in the character strings required by NavisCore 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. Always use modifyNormal when **startDiag**, **getDiag**, **updateDiag**, **stopDiag**, or **GetOperInfoObject** are used.

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.

setNumRetries

Purpose

Sets the number of retries to check card status preceding circuit provisioning requests.

Syntax

CvClient::ApiStatus CvClient::setNumRetries(int *numRetries* **);**

Description

setNumRetries sets the number of retries to check card status preceding circuit provisioning requests.

NumRetries is the number of retries. The value applies to requests at either endpoint: when a retry is sent to obtain the card status of one endpoint, the number of retries decrements for either endpoint. Valid values 0 - 5.

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.

For details on the Provisioning Server's retry behavior to ensure reliability and accuracy of circuit provisioning, see Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

If you do not want the Provisioning Server to obtain card status prior to provisioning circuits, you can override the server's default behavior. Set the server environment variable CV_CARD_STATS to DISABLE to disable card status checking on circuit endpoints. For details, see Chapter 2 of the *NavisXtend Provisioning Server User's Guide*.

Prerequisites

You must establish a session with the Provisioning Server.

Post-processing

None.

Terminating a Session

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

This section describes **CvClient**'s operational member functions.

addMember (asynchronous)

Purpose

Adds a member to a list of objects.

Syntax

CvReqId CvClient::addMember(
const CvClient::ObjectId &C*ontainer*,
const CvClient::ObjectId &*Member*,
CvClient::Callback ***Callback*);**

Description

addMember is an asynchronous function that adds a member to an object list. Use this function to add an address to a screen or netwide group address or to add an MLFRMember LPort to a MLFRBundle LPort. Upon completion of the function, the address or LPort 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.

Member is the **CvClient::CvObjectId** class that specifies the member object.

Callback is a reference to the callback function of the class **CvClient::Callback**.

Notes

For a list of object types that you can specify with this function, see Table 1-5 on page 1-10.

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.

When you specify the object CVT_LPort as the container object, the container must be an MLFRBundle LPort and the member to be added must be an MLFRMember LPort.

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).

addMember (synchronous)

Purpose

Adds a member to a list of objects.

Syntax

void CvClient::addMember(
const CvClient::ObjectId &*Container*,
const CvClient::ObjectId &*Member*,
CvClient::ApiStatus ***Status** **);**

Description

addMember is a synchronous function that adds a member to an object list. Use this function to add an address to a screen or netwide group address or to add an MLFRMember LPort to a MLFRBundle LPort. Upon completion of the function, the address or LPort 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.

Member is the **CvClient::CvObjectId** class that specifies the member object.

Status, an output parameter, is a reference 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, see Table 1-5 on page 1-10.

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.

When you specify the object CVT_LPort as the container object, the container must be an MLFRBundle LPort and the member to be added must be an MLFRMember LPort.

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).

addObject (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 &*Arg*s **);**

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 reference 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 (see 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, see Table 1-5 on page 1-10.

To create a card or PPort, use a Modify function (asynchronous or synchronous version of **modifyObject**). The NavisCore 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, NavisCore automatically populates the card with all necessary physical ports. Use the Modify function to change the PPort specifications. In the case of channelized DS3 cards, once the card has been configured, NavisCore 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).

addObject (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 (see 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 reference 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, see Table 1-5 on page 1-10.

To create a card or PPort, use a Modify function (asynchronous or synchronous version of **modifyObject**). The NavisCore 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, NavisCore 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, NavisCore 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).

deleteMember (asynchronous)

Purpose

Removes a member from a list of objects.

Syntax

CvReqId CvClient::deleteMember(
const CvClient::ObjectId &*Container*,
const CvClient::ObjectId &*Member*,
CvClient::Callback ***Callback** **);**

Description

deleteMember is an asynchronous function that removes a member from an object list. Use this function to remove an address from a screen or netwide group address or to unbind an MLFRMember LPort from a MLFRBundle LPort. Upon completion of the function, the address or LPort 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.

Member is the **CvClient::CvObjectId** class.

Callback is a reference to the callback function of the class **CvClient::Callback**.

Notes

For a list of object types that you can specify with this function, see Table 1-5 on page 1-10.

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.

When you specify the object CVT_LPort as the container object, the container must be an MLFRBundle LPort and the member to be removed must be an MLFRMember LPort.

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).

deleteMember (synchronous)

Purpose

Removes a member from a list of objects.

Syntax

void CvClient::deleteMember(
const CvClient::ObjectId &*Container*,
const CvClient::ObjectId &*Member*,
CvClient::ApiStatus ***Status** **);**

Description

deleteMember is a synchronous function that removes a member from an object list. Use this function to remove an address from a screen or netwide group address or to unbind an MLFRMember LPort from a MLFRBundle LPort. Upon completion of the function, the address or LPort 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.

Member is the **CvClient::CvObjectId** class.

Status, an output parameter, is a reference 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, see Table 1-5 on page 1-10.

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.

When you specify the object CVT_LPort as the container object, the container must be an MLFRBundle LPort and the member to be removed must be an MLFRMember LPort.

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).

deleteObject (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 reference to the **CvClient::ObjectId** class that specifies the object to be deleted.

Callback is a reference to the callback function of the class **CvClient::Callback**.

Notes

For a list of object types that you can specify with this function, see Table 1-5 on page 1-10.

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).

deleteObject (synchronous)

Purpose

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

Syntax

void CvClient::deleteObject(
const CvClient::ObjectId &*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 reference to the **CvClient::ObjectId** class that specifies the object to be deleted.

Status, an output parameter, is a reference 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, see Table 1-5 on page 1-10.

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).

getDiagObject (asynchronous)

Purpose

Retrieves the values of specific diagnostic attributes.

Syntax

CvReqId getDiagObject(
const CvClient::ObjectId &*Object*,
CvClient::Callback ***Callback**,
const CvClient::Args &*Args* **);**

Description

getDiagObject is an asynchronous function that retrieves the diagnostic results on an object being diagnosed in the network.

Object is a reference to the **CvClient::ObjectId** class that specifies the object to be diagnosed.

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 (see 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, see Table 1-5 on page 1-10.

Prerequisites

You must establish a session with the Provisioning Server.

Create each of the following:

- A class instance of **CvClient::ObjectId** for the object to be diagnosed.
- A class instance of **CvClient::Args** for the list of attributes with which to set up the diagnostics.

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 **getDiagObject** (synchronous).

getDiagObject (synchronous)

Purpose

Retrieves the values of specific diagnostic attributes.

Syntax

Args CvClient::getDiagObject(
const CvClient::ObjectId *&Object*,
const CvClient::Args *&Args*,
CvClient::ApiStatus ****Status* **);

Description

getDiagObject is a synchronous function that retrieves the diagnostic results on an object being diagnosed in the network.

Object is a reference to the **CvClient::ObjectId** class that specifies the object to be diagnosed.

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 (see 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, see Table 1-5 on page 1-10.

Prerequisites

You must establish a session with the Provisioning Server.

Create each of the following:

- A class instance of **CvClient::ObjectId** for the object to be diagnosed.
- A class instance of **CvClient::Args** for the list of attributes to be retrieved.

Post-processing

None.

See Also

getDiagObject (asynchronous).

getErrorMsg

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

CvClient::ObjectList::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 reference to the **CvClient::ObjectId** class that specifies the object whose attributes are to be retrieved.

Callback is a reference 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 (see 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, see Table 1-5 on page 1-10.

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 (asynchronous, ObjectList)

Purpose

Retrieves the values of specific attributes from the database.

Syntax

CvReqId CvClient::getObject(
const CvClient::ObjectId &*ObjId***,**
CvClient::ObjectList &*ObjList***,**
CvClient::Args &*Args***,**
CvClient::Callback ***Callback*);**

Description

getObject is an asynchronous function that retrieves the values of specific attributes from the database. Use this function whenever you wish to retrieve an object attribute that is represented as an ObjectList.

ObjId is a reference to the **CvClient::ObjectId** class that specifies the object whose attributes are to be retrieved.

ObjList should be set to NULL.

Args is the **CvClient::Args** class that specifies the attributes to be retrieved. Only one of these attributes can be an attribute whos representation is an ObjectList. The API will modify this argument list by removing the ArgId that specified the attribute containing the ObjectList. The ObjectList attribute is retrieved by the client using the function **CvClient::getResponseObjList**.

Callback is the callback function.

Notes

Use this function with the DefinedPath object.

Prerequisites

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

Post-processing

The application must return to the **select** loop. Invoke **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, Args)

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 reference 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 (see 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 reference 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, see Table 1-5 on page 1-10.

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).

getObject (synchronous, ObjectList)

Purpose

Retrieves the values of specific attributes from the database.

Syntax

CvClient::Args& CvClient::getObject(
const CvClient::ObjectId &*ObjId*,
CvClient::ObjectList &*ObjList*,
CvClient::Args &*Args*,
CvClient::ApiStatus *Status* **);**

Description

getObject is a synchronous function that retrieves the values of specific attributes from the database. Use this function whenever you wish to retrieve an object attribute that is represented as an ObjectList.

ObjId is a reference to the **CvClient::ObjectId** class that specifies the object whose attributes are to be retrieved.

ObjList should be set to NULL.

Args is the **CvClient::Args** class that specifies the attributes to be retrieved. Only one of these attributes can be an attribute whos representation is an ObjectList. The API will modify this argument list by removing the ArgId that specified the attribute containing the ObjectList. The ObjectList attribute is retrieved by the client using the function **CvClient::getResponseObjList**.

Status, an output parameter, is a reference to the **CvClient::Status** variable that contains the status returned by the function.

Notes

Use this function with the DefinedPath class.

Prerequisites

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

Initialize status to **cveNoError**.

Post-processing

If the status has a value other than **cveNoError**, issue a **CvClient::getresponseObjList** function to get the ObjectList value that was returned from the server. This list will be altered according to the error that occurred. Check the status of each object in the list, and each argument of each object in the list, to determine where the error occurred.

See Also

getObject (asynchronous).

getOperInfoObject (asynchronous)

Purpose

Retrieves the values of specific OperInfo (real time status) attributes from the switch.

Syntax

CvReqId CvClient::getOperInfoObject (
const CvClient::ObjectId &*ObjId*,
CvClient::Callback **Callback*,*
const CvClient::Args &*Args* **);**

Description

getOperInfoObject is an asynchronous function that retrieves the values of specific OperInfo (real time status) attributes.

ObjId is a pointer to the **CvClient::CvObjectId** class that specifies the objects whose attributes are to be retrieved.

Callback is a reference to the callback function of the class **CvClient::Callback**.

Args is a constant **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 (see the appropriate attribute tables in *Provisioning Server Object Attribute Definitions*). Specify one of the following attribute IDs with null values:

- The value-less constructor for the **CvClient::Args** class.
- The **CvClient::Value::setNull** function.

Notes

The valid object type for this function is **CVT_Circuit**.

Prerequisites

You must establish a session with the Provisioning Server.

Before you issue this function, 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 **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 **getOperInfoObject** (synchronous).

getOperInfoObject (synchronous)

Purpose

Retrieves the values of specific OperInfo attributes from the switch.

Syntax

CvClient::Args CvClient::getOperInfoObject(
const CvClient::ObjectId &*ObjId*,
const CvClient::Args &*Args*,
CvClient::ApiStatus ***Status** **);**

Description

getOperInfoObject is a synchronous function that retrieves the values of specific attributes from the switch.

ObjId is a pointer to the **CvClient::CvObjectId** class that specifies the objects whose attributes are to be retrieved.

Args is a constant **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 attribute tables in *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 reference to the **CvClient::ApiStatus** variable that contains the status returned by the function.

Notes

The valid object type for this function is **CVT_Circuit**

Prerequisites

You must establish a session with the Provisioning Server.

Before you issue this function, 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

getOperInfoObject (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

CvClient::ObjectList::errorIndex.

getResponseErrorIndex

Purpose

Returns the error index of the response for a synchronous request.

Syntax

const CvClient::getResponseErrorIndex( );

Description

getResponseErrorIndex returns the index to the first parameter that has an error in the response of a synchronous request.

Notes

The value 0 indicates a non-error status. Index values range from 1 to the number of args.

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

CvClient::ObjectList::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 as a parent of multiple LPorts).

Callback is a reference 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 6-1 lists the valid parent and child object types you can specify with this function.

Table 6-1. Valid Parent and Child Object Types

Parent Object Type	Child Object Types
Card	CardTca PPort
Channel	LPort PerformanceMonitor
Customer	Circuit LPort

Table 6-1. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
LPort	AssignedSvcSecScn Circuit LPort (only for listing MLFR Members on an MLFR Bundle) PMPCktRoot PMPSpvcRoot SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress Spvc SvcAddress SvcConfig SvcNetworkId SvcPrefix SvcSecScnActParam SvcUserPart VpciTable
Network	Customer NetCac ServiceName SmdsCountryCode SmdsNetwideGroupAddress SvcCUG SvcCUGMbrRule SvcSecScn Switch TrafficDesc Trunk VPN
PMPCktRoot	PMPCktLeaf
PMPSpvcRoot	PMPSpvcLeaf
PPort	Aps (1-port OC-12c/STM-4 and OC-3/STM-1 cards) Channel LPort PerformanceMonitor PFdl (8-port ATM T1 card only) PPortTca TrafficShaper
ServiceName	Circuit
SmdsGroupScreen	SmdsAlienGroupAddress SmdsSwitchGroupAddress
SmdsIndividualScreen	SmdsAlienIndividualAddress SmdsLocalIndividualAddress

Table 6-1. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
SmdsNetwideGroupAddress	SmdsLocalIndividualAddress SmdsSwitchGroupAddress
SvcCUG	SvcCUGMbr
SvcCUGMbrRule	SvcCUGMbr
Switch	Card PnniNode RefTimeServer SmdsAddressPrefix SmdsAlienGroupAddress SmdsAlienIndividualAddress SmdsSwitchGroupAddress SvcNodePrefix
Trunk	Circuit
VPN	Circuit LPort

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::ObjectId &*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 as a parent of multiple LPorts).

Status, an output parameter, is a reference 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 6-2 lists the valid parent and child object types you can specify with this function.

Table 6-2. Valid Parent and Child Object Types

Parent Object Type	Child Object Types
Card	CardTca PPort
Channel	LPort PerformanceMonitor
Customer	Circuit LPort

Table 6-2. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
LPort	AssignedSvcSecScn Circuit LPort (only for listing MLFR Members on an MLFR Bundle) PMPCktRoot PMPSpvcRoot SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress Spvc SvcAddress SvcConfig SvcNetworkId SvcPrefix SvcSecScnActParam SvcUserPart VpciTable
Network	Customer NetCac ServiceName SmdsCountryCode SmdsNetwideGroupAddress SvcCUG SvcCUGMbrRule SvcSecScn Switch TrafficDesc Trunk VPN
PMPCktRoot	PMPCktLeaf
PMPSpvcRoot	PMPSpvcLeaf
PPort	Aps (1-port OC-12c/STM-4 and OC-3/STM-1 cards) Channel LPort PerformanceMonitor PFdl (8-port ATM T1 card only) PPortTca TrafficShaper
ServiceName	Circuit
SmdsGroupScreen	SmdsAlienGroupAddress SmdsSwitchGroupAddress
SmdsIndividualScreen	SmdsAlienIndividualAddress SmdsLocalIndividualAddress

Table 6-2. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
SmdsNetwideGroupAddress	SmdsLocalIndividualAddress SmdsSwitchGroupAddress
SvcCUG	SvcCUGMbr
SvcCUGMbrRule	SvcCUGMbr
Switch	Card PnniNode RefTimeServer SmdsAddressPrefix SmdsAlienGroupAddress SmdsAlienIndividualAddress SmdsSwitchGroupAddress SvcNodePrefix
Trunk	Circuit
VPN	Circuit LPort

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).

listContainedObject (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 as a parent of multiple LPorts) or the parent object can be positioned higher in the containment hierarchy (such as a switch as a parent of multiple LPorts).

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

Callback is a reference 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 6-3 lists the valid parent and child object types you can specify with this function.

Table 6-3. Valid Parent and Child Object Types

Parent Object Type	Child Object Types
Card	Aps CardTca Channel Circuit LPort Performance Monitor PFdl PMPSpvcRoot PPort Spvc SmdsAlienGroupAddress SmdsAlienIndividualAddress SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress SmdsSwitchGroupAddress SvcConfig SvcNodePrefix SvcUserPart Trunk
Channel	Circuit LPort Performance Monitor Trunk
Customer	Circuit LPort
LPort	AssignedSvcSecScn Circuit PMPCktRoot PMPSpvcRoot SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress Spvc SvcAddress SvcConfig SvcNetworkId SvcPrefix SvcSecScnActParam SvcUserPart VpciTable Trunk

Table 6-3. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
Network	Circuit Customer NetCac PMPCktLeaf PMPCktRoot PMPSpvcRoot ServiceName SmdsAddressPrefix SmdsCountryCode SmdsLocalIndividualAddress SmdsNetwideGroupAddress SvcCUG SvcCUGMbrRule SvcSecScn Switch TrafficDesc Trunk VPN
PMPCktRoot	PMPCktLeaf
PMPSpvcRoot	PMPSpvcLeaf
PPort	Aps Channel Circuit LPort PerformanceMonitor PFdl (8-port ATM T1 card only)| PMPCktRoot PMPSpvcRoot PPortTca Spvc Trunk
SmdsGroupScreen	SmdsAlienGroupAddress SmdsSwitchGroupAddress
SmdsIndividualScreen	SmdsAlienIndividualAddress SmdsLocalIndividualAddress
SmdsNetwideGroupAddress	SmdsLocalIndividualAddress SmdsSwitchGroupAddress
SvcCUG	SvcCUGMbr
SvcCUGMbrRule	SvcCUGMbr

Table 6-3. Valid Parent and Child Object Types (Continued)

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

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).

listContainedObject (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 as a parent of multiple LPorts) or the parent object can be positioned higher in the containment hierarchy (such as a switch as 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 reference 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 6-4 lists the valid parent and child object types you can specify with this function.

Table 6-4. Valid Parent and Child Object Types

Parent Object Type	Child Object Types
Card	Aps CardTca Channel Circuit LPort Performance Monitor PFdl PMPSpvcRoot PPort Spvc SmdsAlienGroupAddress SmdsAlienIndividualAddress SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress SmdsSwitchGroupAddress SvcConfig SvcNodePrefix SvcUserPart Trunk
Channel	Circuit LPort Performance Monitor Trunk
Customer	Circuit LPort
LPort	AssignedSvcSecScn Circuit PMPCktRoot PMPSpvcRoot SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress Spvc SvcAddress SvcConfig SvcNetworkId SvcPrefix SvcSecScnActParam SvcUserPart VpciTable Trunk

Table 6-4. Valid Parent and Child Object Types (Continued)

Parent Object Type	Child Object Types
Network	Circuit Customer NetCac PMPCktLeaf PMPCktRoot PMPSpvcRoot ServiceName SmdsAddressPrefix SmdsCountryCode SmdsLocalIndividualAddress SmdsNetwideGroupAddress SvcCUG SvcCUGMbrRule SvcSecScn Switch TrafficDesc Trunk VPN
PMPCktRoot	PMPCktLeaf
PMPSpvcRoot	PMPSpvcLeaf
PPort	Aps Channel Circuit LPort PerformanceMonitor PFdl (8-port ATM T1 card only)| PPortTca Trunk TrafficShaper
SmdsGroupScreen	SmdsAlienGroupAddress SmdsSwitchGroupAddress
SmdsIndividualScreen	SmdsAlienIndividualAddress SmdsLocalIndividualAddress
SmdsNetwideGroupAddress	SmdsLocalIndividualAddress SmdsSwitchGroupAddress
SvcCUG	SvcCUGMbr
SvcCUGMbrRule	SvcCUGMbr

Table 6-4. Valid Parent and Child Object Types (Continued)

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

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, Args)

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 reference 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 (see 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, see Table 1-5 on page 1-10.

Use this function to create a card or PPort. The NavisCore 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, NavisCore automatically populates the card with all necessary physical ports. Use this function to change the PPort specifications. In the case of channelized DS3 cards, once the card has been configured, NavisCore 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 (asynchronous, ObjectList)

Purpose

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

Syntax

CvReqId CvClient::modifyObject(
const CvClient::CvObjectId &*ObjectId***,**
const CvClient::ObjectList &*ObjList*
const CvClient::Args &*Args***,**
CvClient::Callback **Callback* **);**

Description

modifyObject is an asynchronous function that modifies the specific attributes of an object in the database and (optionally) in the switch. Use this function to modify an object's attribute that is specified as an ObjectList.

ObjectId is a reference to the **CvClient::ObjectId** structure that contains the list of objects to be modified.

ObjList specifies the new ObjectList.

Args is the argument list associated with the container object. Each member of the list has its own CvArgs structure.

Callback is the pointer to the **CvClient::Callback** class.

Notes

Use this function only with the DefinedPath object.

Prerequisites

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

Post-processing

The application must return to the **select** loop. Invoke **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, Args)

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*,
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.

Args is the **CvClient::Args** class that specifies the attributes for the object to be modified. Specify only those attributes appropriate for the object type (see the *NavisXtend Provisioning Server Object Attribute Definitions*). You can modify any attributes except those with either the Read-Only or Create-Only access restriction.

Status, an output parameter, is a reference 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, see Table 1-5 on page 1-10.

Use this function to create a card or PPort. The NavisCore 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, NavisCore automatically populates the card with all necessary physical ports. Use this function to change the PPort specifications. In the case of channelized DS3 cards, once the card has been configured, NavisCore 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).

modifyObject (synchronous, ObjectList)

Purpose

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

Syntax

void CvClient::modifyObject(
const CvClient::CvObjectId &*ObjectId*,
const CvClient::ObjectList &*ObjList*,
const CvClient::Args &*Args*,
CvClient::ApiStatus **Status* **);**

Description

modifyObject is an asynchronous function that modifies the specific attributes of an object in the database and (optionally) in the switch. Use this function to modify an object's attribute that is specified as an ObjectList.

ObjectId is a reference to the **CvClient::ObjectId** structure that contains the list of objects to be modified.

ObjList specifies the new ObjectList.

Args is the argument list associated with the container object. Each member of the list has its own CvArgs structure.

Status, an output parameter, is a reference to the CvStatus variable that contains the status returned by the function.

Notes

Use this function only with the DefinedPath object.

Prerequisites

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

Initialize status to **cveNoError**.

Post-processing

If the status has a value other than **cveNoError**, issue a **CvClient::getresponseObjList** function to get the ObjectList value that was returned from the server. This list will be altered according to the error that occurred. Check the status of each object in the list, and each argument of each object in the list, to determine where the error occurred.

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 reference 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).

startDiagObject (asynchronous)

Purpose

Starts the diagnostics for an object in the network.

Syntax

CvReqId startDiagObject(
const CvClient::ObjectId &*Object*,
CvClient::Callback ***Callback**,
const CvClient::Args &*Args* **);**

Description

startDiagObject is an asynchronous function that starts the diagnostics on an object in the network. The function includes diagnostic parameters that are set on the switch.

Object is the **CvClient::ObjectId** class that specifies the object to be diagnosed.

Callback is a pointer to the callback function of the class **CvClient::Callback**.

Args is the **CvClient::Args** class that specifies the attributes that set up the diagnostics. The API does not modify or delete this argument. Specify only those attributes appropriate for the object type (see the *NavisXtend Provisioning Server Object Attribute Definitions*).

Notes

For a list of object types that you can specify with this function, see Table 1-5 on page 1-10.

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 omit.

Prerequisites

You must establish a session with the Provisioning Server.

Create each of the following:

- A class instance of **CvClient::ObjectId** for the object to be diagnosed.
- A class instance of **CvClient::Args** for the list of attributes with which to set up the diagnostics.

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 **startDiagObject** (synchronous).

startDiagObject (synchronous)

Purpose

Starts the diagnostics for an object in the network.

Syntax

void startDiagObject(
const CvClient::ObjectId &*Object*,
const CvClient::Args &*Args*,
CvClient::ApiStatus ***Status*);**

Description

startDiagObject is a synchronous function that starts the diagnostics on an object in the network. The function includes diagnostic parameters that are set on the switch.

Object is the **CvClient::ObjectId** class that specifies the object to be diagnosed.

Args is the **CvClient::Args** class that specifies the attributes that set up the diagnostics. The API does not modify or delete this argument. Specify only those attributes appropriate for the object type (see the *NavisXtend Provisioning Server Object Attribute Definitions*).

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, see Table 1-5 on page 1-10.

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 omit.

Prerequisites

You must establish a session with the Provisioning Server.

Create each of the following:

- A class instance of **CvClient::ObjectId** for the object to be diagnosed.
- A class instance of **CvClient::Args** for the list of attributes with which to set up the diagnostics.

Post-processing

If the response has the status **cveArgumentNotPresent**, issue **CvGetResponseArgs** to obtain the argument list returned by the function.

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 **startDiagObject** (synchronous).

stopDiagObject (asynchronous)

Purpose

Stops the diagnostics for an object in the network.

Syntax

CvReqId stopDiagObject(
const CvClient::ObjectId &*Object*,
CvClient::Callback ***_Callback_ **);

Description

stopDiagObject is an asynchronous function that stops the diagnostics on an object in the network. The function includes diagnostic parameters that are set on the switch.

Object is the **CvClient::ObjectId** class that specifies the object to be iagnosed.

Callback* is a pointer to the callback function of the class **CvClient::Callback.

Args is the **CvClient::Args** class that specifies the attributes that set up the diagnostics. The API does not modify or delete this argument. Specify only those attributes appropriate for the object type (see the *NavisXtend Provisioning Server Object Attribute Definitions*).

Notes

For a list of object types that you can specify with this function, see Table 1-5 on page 1-10.

Prerequisites

You must establish a session with the Provisioning Server.

Create a class instance of **CvClient::ObjectId** for the object to be diagnosed.

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

stopDiagObject (synchronous), **getDiagObject**, **startDiagObject**, and **updateDiagObject**.

stopDiagObject (synchronous)

Purpose

Stops the diagnostics for an object in the network.

Syntax

void stopDiagObject(
const CvClient::ObjectId &*Object*,
CvClient::ApiStatus ***Status** **);**

Description

stopDiagObject is a synchronous function that stops the diagnostics on an object in the network.

Object is the **CvClient::ObjectId** class that specifies the object to be diagnosed.

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, see Table 1-5 on page 1-10.

Prerequisites

You must establish a session with the Provisioning Server.

Create a class instance of **CvClient::ObjectId** for the object to be diagnosed.

Post-processing

None.

See Also

stopDiagObject (asynchronous), **getDiagObject**, **startDiagObject**, and **updateDiagObject**.

updateDiagObject (asynchronous)

Purpose

Modifies the diagnostic parameters for an object being diagnosed in the network.

Syntax

CvReqId updateDiagObject(
const CvClient::ObjectId &*Object*,
CvClient::Callback ***Callback**,
const CvClient::Args &*Args* **);**

Description

updateDiagObject is an asynchronous function that modifies the diagnostic parameters on the switch for an object being diagnosed in the network.

Object is a reference to the **CvClient::ObjectId** class that specifies the object to be diagnosed.

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 (see 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, see Table 1-5 on page 1-10.

The only valid argument ID for **updateDiagObject** is the **CVA_**ObjectType**TestType** for any ObjectType that supports diagnostics.

Prerequisites

You must establish a session with the Provisioning Server.

Create each of the following:

- A class instance of **CvClient::ObjectId** for the object to be diagnosed.
- A class instance of **CvClient::Args** for the list of attributes with which to set up the diagnostics.

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 **updateDiagObject** (synchronous).

updateDiagObject (synchronous)

Purpose

Modifies the diagnostic parameters for an object being diagnosed in the network.

Syntax

Args CvClient::updateDiagObject(
const CvClient::ObjectId *&Object*,
const CvClient::Args *&Args*,
CvClient::ApiStatus ****Status* **);

Description

updateDiagObject is a synchronous function that modifies the diagnostic parameters on the switch for an object being diagnosed in the network.

Object is a reference to the **CvClient::ObjectId** class that specifies the object to be diagnosed.

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 (see 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, see Table 1-5 on page 1-10.

Prerequisites

You must establish a session with the Provisioning Server.

Create each of the following:

- A class instance of **CvClient::ObjectId** for the object to be diagnosed.
- A class instance of **CvClient::Args** for the list of attributes to be retrieved.

Post-processing

None.

See Also

updateDiagObject (asynchronous).

7

CvClient::Callback

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**.

CvClient::Callback (Client Callback Handler class)

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 {
    public:
        MyHandler(/*Your context information here*/);

        void response (
            const CvClient::ObjectId &ObjId,
            CvClient::ApiStatus Status,
            CvReqId RequestId,
            const CvClient::Args *Args );
    }
        void listresponse (
            const CvClient::ObjectId &ObjId,
            CvClient::ApiStatus Status,
            CvReqId RequestId,
            const CvClient::ObjectList *ObjList,
            const CvClient::Args *Args );
    }
```

Description

CvClient::Callback is the abstract 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 for the asynchronous request.

RequestId is the request ID associated with the asynchronous request.

ObjList* is the **CvClient::ObjectList class instance.

Args is the **CvClient::Args** class instance that contains the attributes for the response.

Notes

When you issue an asynchronous request, pass in a pointer to the class object derived from **CvClient::Callback**. For asynchronous requests on non-ObjectList objects, the API calls the virtual member function response upon receiving the response. For asynchronous requests on ObjectList objects, the API calls the virtual member function **listresponse** upon receiving the 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.

getSelectInfo

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 ***Numfd*,**
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**.

processEvents

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 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**.

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**.

8

CvClient::ObjectId

CvClient::ObjectId is 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::ObjectId

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.

getCard

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.

getDlci

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.

getName

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.

getNetwork

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.

getNumRetries

Purpose

Obtains the current retry setting for card status checks preceding circuit provisioning requests.

Syntax

int CvClient::getNumRetries( )

Description

getNumRetries obtains the current retry setting for card status checks preceding circuit provisioning requests.

Notes

For details on the Provisioning Server's retry behavior to ensure reliability and accuracy of circuit provisioning, see Chapter 1 of the *NavisXtend Provisioning Server User's Guide*.

If you do not want the Provisioning Server to obtain card status prior to provisioning circuits, you can override the server's default behavior. Set the server environment variable CV_CARD_STATS to DISABLE to disable card status checking on circuit endpoints. For details, see Chapter 2 of the *NavisXtend Provisioning Server User's Guide*.

Prerequisites

None.

Post-processing

None.

See Also

setNumRetries.

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.

getPnniNodeId

Purpose

Reads the PNNI Peer Group string out of an object ID.

Syntax

const char* CvClient::ObjectId::getPnniNodeId( )

Description

getPnniNodeId reads the PNNI Peer Group 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 PNNI Peer Group string consists of a Peer Group Level and a Peer Group ID, separated by a dash (-). The Peer Group Level value is a decimal number representing the number of bits allowed in the Peer Group ID field (range 0 - 104). The Peer Group ID value is a hexidecimal number up to 28 characters (limited to the characters 0 - f).

Notes

If the type of object specified by **CvClient::ObjectId** is not CVT_PnniNode or the child of a switch, the function returns 0.

Prerequisites

Make sure that the object ID is valid.

Post-processing

None.

See Also

Other **get***ObjectType* functions.

getPPort

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.

getRefTimeServerIpAddr

Purpose

Reads the reference time server IP address out of an object ID.

Syntax

CvClient::IpAddr CvClient::ObjectId::getRefTimeServerIpAddr( ) const

Description

getRefTimeServerIpAddr reads the reference time server 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.

Notes

To read the reference time server switch ID out of an object ID, use **getSwitchId**.

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.

getSVCAddress

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 the different SVC address formats:

E.164native
DCCAESA
ICDAESA
E.164AESA
CustomAESA
DefaultRoute
UserPart
Carrier ID Code (CIC)
Data Network ID Code (DNIC)
X.121
DCCAnycastAESA
ICDAnycastAESA
E.164AnycastAESA

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, see Chapter 12, “CvSVCAddress”.

Prerequisites

Make sure that the object ID is valid.

Post-processing

None.

See Also

Other **get***ObjectType* functions.

getSwitch

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.

getTrafficShaper

See Also Other **get***ObjectType* functions

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.

getVci

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.

getVpciNumber

Purpose Reads the Management DLCI number out of an object ID.

Syntax **int CvClient::ObjectId::getVpciNumber( ) const**

Description **getVpciNumber** reads the Management 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_VpciTable or the child of a VpciTable, the function returns 0.

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.

set

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_Trunk
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.

setPnniNodeId

Purpose

Sets the necessary fields of an object ID with the values for an object's PnniNode ID.

Syntax

int CvClient::ObjectId::setPnniNodeId(
const char ****pgString*,**
int *changeType* **);**

Description

setPnniNodeId fills in an object ID with a value for an object's PnniNode ID. This function is part of the **set***ObjectType* family of functions used to create an object ID based on the containment hierarchy.

pgString is a pointer to a PNNI Peer Group string consisting of a Peer Group Level and a Peer Group ID, separated by a dash (-). The Peer Group Level value is a decimal number representing the number of bits allowed in the Peer Group ID field (range 0 - 104). The Peer Group ID value is a hexidecimal number up to 28 characters (limited to the characters 0 - f).

changeType, an optional parameter, indicates whether you want to change the object type to CVT_PnniNode. 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_PnniNode.

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 PnniNode 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.

setRefTimeServerIpAddr

Purpose

Sets the reference time server field of an object ID with the IP address for an object's reference time server.

Syntax

int CvClient::ObjectId::setRefTimeServerIpAddr(
const CvClient::IpAddr& *addr*,
int *changeType* **);**

Description

setRefTimeServerIpAddr fills in an object ID with the IP address for the reference time server. 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 **CvClient::IpAddr** class that specified the IP address of the host on which the reference time server is running. Specify the address as a string or an unsigned long.

changeType, an optional parameter, indicates whether you want to change the object type to CVT_RefTimeServer. 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_RefTimeServer.

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 IP 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.

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 different SVC address formats:

E.164native
DCCAESA
ICDAESA
E.164AESA
CustomAESA
DefaultRoute
UserPart
Carrier ID Code (CIC)
Data Network ID Code (DNIC)
X.121
DCCAnycastAESA
ICDAnycastAESA
E.164AnycastAESA

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, see Chapter 12, "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.

setType

See Also Other **set***ObjectType* functions.

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.

setVirtualUniLPort

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.

setVpciNumber

Purpose

Sets the VpciTable field of an object ID with a value for an object's Vpci number.

Syntax

int CvClient::ObjectId::setVpciNumber(
unsigned long *instance***,**
int *changeType* **);**

Description

setVpciNumber fills in an object ID with a value for an object's Vpci number. 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 Vpci number, in the range 0 - 65535.

changeType, an optional parameter, indicates whether you want to change the object type to CVT_VpciTable. 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_VpciTable.

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 Vpci 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.

type

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.

9

CvClient::Args

CvClient::Args is a class that represents the argument list and contains member functions to add values to and read values from the argument list.

CvClient::Args

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**.

errorIndex

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.

getStatus

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.

idAt

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**).

statusAt

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**.

10

CvClient::ObjectList

CvClient::ObjectList is a class that consists of pairs of object identifiers (**CvClient::ObjectId**) and argument lists (**CvClient::Args**) pairs. The class has member functions to set and read specific ObjectId-Args pairs to and from the ObjectList. The classes **CvClient::ObjectId** and **CvClient::Args** are existing classes supported in the Provisioning Server toolkit. See Chapter 8, “CvClient::ObjectId,” and Chapter 9, “CvClient::Args,” for information about these classes.

CvClient::ObjectList

Purpose

Builds a list that consists of object identifier (**CvClient::ObjectId**) and argument list (**CvClient::Args**) pairs, enabling the client to transport lists of objects to and from the Provisioning Server. This null constructor declares an empty ObjectList.

Syntax

CvClient::ObjectList ( );
CvClient::ObjectList (const ObjectList &);

Description

CvClient::ObjectList builds a list of object identifier (**CvClient::ObjectId**) argument list (**CvClient::Args**) pairs. The client can use this list to transport objects and attributes to and from the Provisioning Server.

The class has member functions to set and read specific objectId-Args pairs to and from the ObjectList. The classes **CvClient::ObjectId** and **CvClient::Args** are existing classes supported in the Provisioning Server toolkit.

The API treats instances of **CvClient::ObjectList** as single values. You can make copies of **CvClient::ObjectList** objects, create an individual **CvClient::ObjectList** and assign the object with the assignment operator. You can also create a temporary **CvClient::ObjectList** and pass it as an argument.

You can add attributes stored in **CvClient::Args** as well as objects stored in **CvClient::ObjectId** to the lists transported to and from the Provisioning Server.

For each object, there is also a status. If there is an error with one of the objects specified in the list, the server reports the error code to the client.

Notes

A copy constructor is provided: if an ObjectList already exists, the copy constructor creates a copy of it. Use this function if you have a subset of a list and want to keep the original and add members to it. After the list is copied, you use the operator [ ] to add to it. The operator [ ] indexes the object list by using the object's ObjectId, and it returns a reference to **CvClient::Args**. The operator [ ] function uses the following syntax:

CvClient::Args& CvClient::ObjectList::operator [ ] (
const CvClient::ObjectId&);

You can operate on pairs of **CvClient::ObjectList** objects with the following operators:

= Assigns an object list to another object list.

CvClient::ObjectList& CvClient::ObjectList::operator=(const ObjectList&)

== Compares two object lists for equality. Object lists are equal if their object numbers, IDs, values, and status are equal. If the object lists are equal, the operator returns a non-zero value. This operator is order-independent.

int CvClient::ObjectList::operator == (const ObjectList&)const

!= Compares two object lists for inequality. Object lists are unequal if their object numbers, IDs, values, or status differ. If the object lists are unequal, the operator returns a non-zero value. This operator is order-independent.

int CvClient::ObjectList::operator != (const ObjectList&)const

+ Adds two object lists together. The result is a new object list that combines both sets of objects. When an object exists in both object lists, the value from the second list is used. This operator is order-dependent.

CvClient::ObjectList CvClient::ObjectList::operator + (const ObjectList&)const

- Subtracts two object lists. The result is a new object list that contains the objects from the first set that are not contained in the second set. This operator is order-dependent.

CvClient::ObjectList CvClient::ObjectList::operator - (const ObjectList&)const

***** Returns the intersection of two object lists. The result is a new object list containing objects that exist in both sets of objects. Values from the first list are used. This operator is order-dependent.

CvClient::ObjectList CvClient::ObjectList::operator * (const ObjectList&)const

The operator << allows you to write the text object descriptions of an object list to an ostream. For example, an ostream could be a file or a character stream into which you are encoding objects.

ostream& CvClient::ObjectList::operator << (ostream&, const CvClient::ObjectList&)

addToObjectList

Purpose

Adds the given **CvClient::ObjectId** to the ObjectList.

Syntax

```
void CvClient::ObjectList::addToObjectList(
        const CvClient::ObjectId&,
        const CvClient::Args& );
```

Description

Adds the given **CvClient::ObjectId** to the ObjectList. Each element of an ObjectList consists of an ObjectId and Args pair. The CvClient::Args& argument can be used to specify attributes of the corresponding ObjectId. If no Args are specified, the ObjectId's corresponding argument list within the ObjectList is initialized to a null argument list.

Prerequisites

None.

Post-processing

None.

argsAt

Purpose

Retrieves the attribute list of an object located at a specified position in an ObjectList.

Syntax

const CvClient::Args& CvClient::ObjectList::argsAt(
int *Index* **) const;**

Description

argsAt retrieves the Args of an object located at a specified position in an ObjectList.

Index is the position of the object in the ObjectList.

Prerequisites

None.

Post-processing

None.

See Also

count, **valueAt,** and **idAt**.

count

Purpose Retrieves the number of objects in an ObjectList.

Syntax **int CvClient::ObjectList::count( ) const;**

Description **count** retrieves the number of objects in an ObjectList.

Prerequisites None.

Post-processing None.

See Also **idAt, valueAt,** and **argsAt**.

errorIndex

Purpose

Determines if any object in an ObjectList has an error status.

Syntax

int CvClient::ObjectList::errorIndex ( ) const;

Description

errorIndex determines if any object in an ObjectList has an error status. If an error status exists, the function returns the index to the first object with an error.

Notes

This function returns:

- Zero, if no error status exists.
- The index of the object's position plus 1 (index + 1), if an error status exists.

Prerequisites

None.

Post-processing

To return the error status for the argument, issue **CvClient::ObjectList::statusAt**.

See Also

statusAt.

exists

Purpose

Tells you whether a specified object exists in an ObjectList.

Syntax

int CvClient::ObjectList::exists (CvClient::ObjectId&) const;

Description

exists tells you whether an object already exists in an ObjectList. This function returns a non-zero value if an object with the given ObjectId exists in the ObjectList.

ObjectId& is the ObjectId for the object in the ObjectList.

Notes

This function returns:

- Zero, if the argument specified by *ObjectId&* is not in the ObjectList.
- The index of the object's position plus 1 (index + 1, if the argument specified by the ObjectId& is found in the ObjectList.

Prerequisites

None.

Post-processing

None.

See Also

idAt

Purpose

Retrieves the object ID at a specified location in an ObjectList.

Syntax

CvClient::ObjectId& CvClient::ObjectList::idAt(
int *Index***) const;**

Description

idAt retrieves the object ID at a specified index in an object list. If the index is invalid, the function returns 0.

Index is the index to the object list.

Notes

Use this function to read object IDs sequentially from a **CvClient::ObjectList** object.

An index of 0 returns a reference to the first ObjectId in the list.

Prerequisites

None.

Post-processing

None.

See Also

count, valueAt, and **argsAt**.

removeFromObjectList

Purpose Removes an object from an ObjectList.

Syntax

```
void CvClient::ObjectList::removeFromObjectList(
        const CvClient::ObjectId& );
```

Description Removes the given CvClient::ObjectId from the ObjectList. If the specified ObjectId does not exist in the list, the function does nothing.

Prerequisites None.

Post-processing None.

statusAt (CvClient::ObjectId&)

Purpose Retrieves the error status code of an ObjectList member object.

Syntax

```
CvClient::ApiStatus CvClient::ObjectList::statusAt(
    CvClient::ObjectId& ) const;
```

Description **statusAt** retrieves the error status code of the specified object.

Prerequisites None.

Post-processing None.

statusAt (int)

Purpose

Retrieves the error status code of an object located at a specified position in the ObjectList.

Syntax

CvClient::ApiStatus CvClient::ObjectList::statusAt(
int *Index* **);**

Description

statusAt retrieves the error status code of the object located at a specified position in the ObjectList.

Index is the position in the ObjectList. If *Index* is out of range, a status of -1 is returned.

Prerequisites

None.

Post-processing

None.

See Also

idAt, errorIndex.

11

CvClient::IpAddr

CvClient::IpAddr is 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::IpAddr

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.

12

CvSVCAddress

CvSVCAddress is used to create SVC addresses. The class enforces the basic syntax rules of the seven different formats of SVC addresses.

CvSVCAddress

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 different SVC address formats:

E.164native
DCCAESA
ICDAESA
E.164AESA
CustomAESA
DefaultRoute
UserPart
Carrier ID Code (CIC)
Data Network ID Code (DNIC)
X.21
DCCAnycastAESA
ICDAnycastAESA
E.164AnycastAESA

For information on the syntax rules of SVC addresses, see 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
8 — X.21
16 — Carrier ID Code (CIC)
32 — Data Network ID Code (DNIC)
100 — DCCAESA
101 — ICDAESA
102 — E.164AESA
103 — CustomAESA
104 — DefaultRoute
105 — UserPart
106 — DCCAnycastAESA
107 — ICDAnycastAESA
108 — E.164AnycastAESA

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
BD — DCCAnycastAESA
C5 — ICDAnycastAESA
C3 — E.164AnycastAESA

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 12-1.

Table 12-1. 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
Carrier ID Code (CIC)	8	string length of address prefix * 8
Data Network ID Code	9	string length of address prefix * 8
X.21	10	string length of address prefix * 8
DCCAnycastAESA	11	(integral-part-of((string length of address prefix + 1)/2)*8)
ICDAnycastAESA	12	(integral-part-of((string length of address prefix + 1)/2)*8)
E.164AnycastAESA	13	(integral-part-of((string length of address prefix + 1)/2)*8)

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.

Section 4
Appendixes

Appendix A **Sample Code**

Appendix B **Error Codes**

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.



The files necessary for developing an NavisXtend Provisioning client program are found in the directory /opt/ProvServ/include.

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_SmdsNet
wideGroupAddress,
0));
      break;
    }

  case CVT_SmdsLocalIndividualAddress:
    {
      objid = makeId(CVT_LPort,0);
      objid =
CvCreateSmdsLocalIndivAddrId(objid,makeAddrString(CVT_SmdsLocal
IndividualAddress
,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] =

“ABCDEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”;
    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.vc
i.100”);

rmObject(“switch.193.1.141.2.card.7.pport.1.startVpi.2.vpi.2.vc
i.101”);

rmObject(“switch.193.1.141.2.card.7.pport.1.startVpi.2.vpi.2.vc
```

```
i.102”);

rmObject(“switch.193.1.141.2.card.7.pport.1.lport.1.vpi.1.vci.1
00”);

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.svcpref
ix.1-111111111111111-120”);

rmObject(“switch.193.1.141.2.card.12.pport.1.startVpi.2.svcaddr
ess.1-111111111111111-120”);

rmObject(“Switch.193.1.141.2.card.7.pport.1.startVpi.2.svcuserp
art.7-aaaaaaaaaaaaa-56”);


rmObject(“Switch.193.1.141.2.card.7.pport.1.startVpi.2.assigned
SvcSecScn.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.5
5”);
    rmObject(

“Switch.193.1.141.2.card.12.pport.1.startVpi.2.Spvc.vpi.5.vci.5
6”);
    rmObject(

“Switch.193.1.141.2.card.12.pport.1.startVpi.2.Spvc.vpi.5.vci.5
7”);

    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.88”);
    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.svcpr
efix.1-111111111111111-120”,
               CVT_SvcPrefix);

addSvcAddr(“switch.193.1.141.2.card.12.pport.1.startVpi.2.svcad
dress.1-111111111111111-120”,
               CVT_SvcAddress);

addSvcAddr(“Switch.193.1.141.2.card.7.pport.1.startVpi.2.svcuse
rpart.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
.securityScreen”);


modifySvcConfig(“Switch.193.1.141.2.card.7.pport.1.startVpi.2.s
vcconfig”);


addSPVC(“Switch.193.1.141.2.card.12.pport.1.startVpi.2.Spvc.vpi
.5.vci.55”,
            “1-111111111111111-120”,
            “spvc1211-1”);

addSPVC(“Switch.193.1.141.2.card.12.pport.1.startVpi.2.Spvc.vpi
.5.vci.56”,
            “1-111111111111112-120”,
            “spvc1211-2”);

addSPVC(“Switch.193.1.141.2.card.12.pport.1.startVpi.2.Spvc.vpi
.5.vci.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.88”);

   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 codes that can be returned by the NavisXtend Provisioning Server API functions or the CLI commands. The 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 code number. Table 2-1 lists the success condition returned, and Table 2-2 lists the error codes.

Table 2-1. Success Condition

Number	Message	Explanation
0	cveNoError	The function returned without error.

Table 2-2. Error Codes

Number	Message	Explanation
1	cveAccessDenied	Access is denied.
2	cveLockRequired	A lock is required to perform this action.
3	cveNoMemory	Out of memory.
4	cveInvalidSession	Invalid session was used.
5	cveInvalidRequest	Invalid request was 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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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 NavisCore-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	cve LPortsExist	The object has logical ports defined for it.
26	cveSwitchNotManaged	This switch will not be managed by NavisCore.
27	cveInvalidSwitchName	Invalid switch name.
28	cveInvalidSwitchType	Invalid switch type.
29	cveOnlyOvDataFound	The object data has been set up by OpenView but not by NavisCore.
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.
34	cveSwitchNotFound	The switch object was not found.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
35	cveMaxNmsAccessExceeded	The maximum number of NMS entries has been exceeded.
36	cveInvalidGatewayAddr	Invalid gateway address.
37	cveNmsPathExists	A path to the specified NMS address already exists.
38	cveInvalidMgmtDlciName	Unable to obtain configuration information for the Management DLCI.
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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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	cveCard LPortsExist	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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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	cve LPortTypeInvalid	Invalid logical port type.
96	cve LPortTypeNotSupported	Logical port type not supported
97	cve LPortServiceInvalid	Invalid logical port service type.
98	cve LPortServiceNotSupported	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).

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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 (0 - 100).
114	cveErrPMinThreshInvalid	Error-per-minute threshold value is invalid.
115	cveAdminStatusInvalid	Invalid administrative status value
116	cveMax LPortsExceeded	Maximum number of logical ports exceeded.
117	cve LPortNameInUse	The specified logical port name is already in use on this switch.
118	cveInvalid LPortId	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	cve LPortIdInUse	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	cve LPortChannelsNotAvail	Some of the DS0s requested are not available.
124	cve LPortBandwidthTooHigh	More bandwidth was requested than the physical port has available.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
125	cveBitStuffForUio	Bit stuffing should not be set for this logical port type.
126	cveBitStuffInvalid	Invalid bit stuffing value.
127	cve LPortChannelsForUio	Fractional DS0s should not be set for this logical port type.
128	cve LPortChannelsNotSet	The fractional DS0s field is not set.
129	cve LPortJammedBit	The physical port's zero coding is set to 'Jammed Bit'; bit stuffing must be on.
130	cve LPortObjectNotFound	The logical port could not be found
131	cve LPortDelTrunkDefined	The logical port cannot be deleted because it is in use as a trunk endpoint.
132	cve LPortDelDlciDefined	The logical port cannot be deleted because a circuit and/or management connection is defined on it.
133	cve LPortDelFeeder	A feeder port cannot be deleted as long as one or more OPTimum \ntrunks of the same service type are defined on the same physical port.
134	cve LPortTypeChange	The service type and port type cannot be changed.
135	cve LPortBandwidthInUse	The specified bandwidth is too low for the circuits defined on the port.
136	cve LPortModTrunkDefined	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	cveTrunkOn LPorts	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 forward 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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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 (1006 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)\ncannot exceed 4263 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	cveMgmtDlciBad LPort	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.\nAlso the QoS classes (forward and reverse) of all the circuits should be the same.
163	cveMcastTooManyMembers	A multicast DLCI can have at most 32 members.
164	cveLockFailed	Lock failed, the object is locked by others.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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	Internal Error: Arguments Required are out of range.
170	cveUIRead	Error in reading user input data.
171	cveCfm_CommitPPort	Changes in physical port attributes must be applied before setting\nlogical port attributes. Press Apply to apply 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	cve LPortDelMgmtAddrDefined	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\nsoftware to the switch?
181	cveCpSlot2Error	+ Undefined Control/Switch Processor card detected in slot 2.
182	cve LPortDelDteAddrExist	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\npram file to the switch?

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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\nthe 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 applied before setting logical port attributes. Press Apply to apply 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 physical port attributes must be applied before setting channel attributes. Press Apply to apply changes.
194	cveCfm_DelSecScn	There exists logical ports to which this screen is assigned. Press OK to confirm the deletion.
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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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	cveCfm_RapidUpgradeSW	Do you really want to schedule a rapid upgrade when there is one already scheduled?
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	cveNull LPortName	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	cve LPortDelMulSsiExist	The logical port cannot be deleted because it is in use by multiplexed DXI SNI LPorts.
260	cve LPortDelIaDefined	The logical port cannot be deleted because there is an SMDS Address defined in it.
261	cveCardSmds LPortsExist	SMDS Logical ports are defined on the card.
262	cveSmdsIaObjectNotFound	The SMDS IA Object could not be found.
263	cveInvalidSsi LPortName	Invalid Ssi LPort name.
264	cveMaxAddrNameSizeExceeded	Address Name is too long.
265	cveSmdsAaObjectNotFound	The SMDS AA Object count not be found.
266	cveSmdsAa LPortTypeInvalid	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 SAA Object which is associated with other screen objects.
269	cveSmdsAddrPoolObjectNotFound	The SMDS Address Pool Object cannot be found.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
270	cveSmdsAddrPool LPortTypeInvalid	Cannot define an SMDS Address Pool Object in the current logical port type.
271	cveSmdsDteAddrNotUnique	Duplicate SMDS DTE Address.
272	cveSmdsAddrSizeInvalid	Invalid SMDS Address size, total number 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	cveSmdsIa LPortTypeInvalid	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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
289	cveSmdsMaskObjectNotFound	The SMDS Prefix Object cannot be found.
290	cveSmdsMgmtAddrObjectNotFound	The SMDS Mgmt Address Object cannot be found.
291	cveSmdsMgmtAddr LPortTypeInvalid	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	cveSmdsScn LPortTypeInvalid	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	cveAddSmds LPortInFrameRelay	Cannot add SMDS Logical ports to a card with Frame Relay Only capability.
303	cve LPortDelScnMbrExist	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 Address.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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.
310	cveSyncFileBadHeader	Unable to process the PRAM file as it contains a bad file header.
311	cveInitDlg	Error initializing the dialog box.
312	cveCfgSyncUploadFailed	Unable to upload PRAM from switch.
313	cveNMSSyncFileBadHeader	Unable to process the NMS-generated PRAM file as it contains a bad file header.
314	cveSWSyncFileBadHeader	Unable to process the uploaded PRAM file as it contains a bad file header.
315	cveNMSSyncFileOpenFail	Unable to open the NMS-generated PRAM file.
316	cveSWSyncFileOpenFail	Unable to open the uploaded PRAM file.
317	cvePramULObjConflict	The database item cannot be constructed from PRAM for deletion or modification because it conflicts with an existing NMS database item.
318	cvePramULDelObjConflict	The database item cannot be constructed from PRAM for deletion because it conflicts with an existing NMS database item.
319	cvePramULAddObjConflict	The PRAM item cannot be added because it conflicts with an existing NMS database item.
320	cvePramULModObjConflict	The database item cannot be constructed from PRAM for modification because it conflicts with an existing NMS database item.
321	cvePramULSkip	Skipping PRAM table.
322	cvePramUL LPortNoSuchPPort	Unable to construct the logical port from PRAM because its owning physical port does not exist.
323	cvePramULCktNoSuchRemoteSwitch	Unable to construct the circuit from PRAM because its remote switch does not exist.
324	cvePramULCktNoSuchRemote LPort	Unable to construct the circuit from PRAM because its remote logical port does not exist.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
325	cvePramULCktNoSuchLocal LPort	Unable to construct the circuit from PRAM because its local logical port does not exist.
326	cvePramULPathNoSuchCkt	Unable to construct the circuit path from PRAM because its circuit does not exist.
327	cvePramULPathNoSuchNode	Unable to construct the circuit path from PRAM because a node in the hop sequence does not exist
328	cvePramULPathNoSuchTrunk	Unable to construct the circuit path from PRAM because a trunk in the hop sequence does not exist
329	cvePramULTrkNoSuchRemoteSwitch	Unable to construct the trunk from PRAM because its remote switch does not exist.
330	cvePramULTrkNoSuchRemote LPort	Unable to construct the trunk from PRAM because its remote logical port does not exist.
331	cvePramULTrkNoSuchLocal LPort	Unable to construct the trunk from PRAM because its local logical port does not exist.
332	cvePramULMgmtDlciNoSuch LPort	Unable to construct the management DLCI from PRAM because its logical port does not exist.
333	cvePramULMcastDlciNoSuch LPort	Unable to construct the multicast DLCI from PRAM because its logical port does not exist.
334	cvePramULMultiMemNoSuchMcastDlci	Unable to add the circuit as a multicast member because its multicast DLCI does not exist.
335	cvePramULNmsPthNoSuch LPort	Unable to construct the NMS Path from PRAM because its logical port does not exist.
336	cvePramULNmsPthNoSuchDlci	Unable to construct the NMS Path from PRAM because its DLCI does not exist.
337	cvePramULNmsPthNoSuchMgmtDlci	Unable to construct the NMS Path from PRAM because its Management DLCI does not exist.
338	cvePramULNmsPthNoSuchSmdsMgmtAddr	Unable to construct the NMS Path from PRAM because its Smds Management Address does not exist.
339	cvePramULIaMaskInvalid	Unable to construct the SMDS IA from PRAM because its address prefix contains too few or too many digits.
340	cvePramULIaNoSuch LPort	Unable to construct the SMDS IA from PRAM because its logical port does not exist.
341	cvePramULGaNoSuchIa	Unable to add the SMDS IA as a GA member because the IA does not exist.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
342	cvePramULGaNoSuchGa	Unable to add the SMDS IA as a GA member because its GA does not exist.
343	cvePramULIScnNoSuch LPort	Unable to add the SMDS IA screen from PRAM because its logical port does not exist.
344	cvePramULIScnNoSuchIa	Unable to add the SMDS IA as an IA screen member because it does not exist.
345	cvePramULIScnNoSuchAIa	Unable to add the SMDS AIA as an IA screen member because it does not exist.
346	cvePramULGScnNoSuch LPort	Unable to add the SMDS GA screen from PRAM because its logical port does not exist.
347	cvePramULGScnNoSuchGa	Unable to add the SMDS GA as a GA screen member because it does not exist.
348	cvePramULGScnNoSuchAGa	Unable to add the SMDS AGA as a GA screen member because it does not exist.
349	cvePramULSmdsMgmAddrNoSuchNmsPath	Unable to construct the SMDS Management Address because its NMS path does not exist.
350	cvePramULSmdsMgmAddrNoSuchPPort	Unable to construct the SMDS Management Address because its physical port does not exist.
351	cvePramULSmdsMgmAddrNoSuch LPort	Unable to construct the SMDS Management Address because its logical port does not exist.
352	cvePramUL LPortNoSuchSsi LPort	Unable to construct the logical port from PRAM because its SSI logical port does not exist.
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 LPorts.
358	cveBillingAggPeriodInvalid	Billing aggregation period must be greater than 10 minutes.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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 (0-64).
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 trafic and disrupt user data.\nDo 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 or equal to PCR and greater than (PCR/63).
374	cve LPortPPortConflict	This logical port cannot be added to this physical port.
375	cve LPortDelAtmXportFeeder	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	cveCktScrNon0BtInvalid	When SCR is non-zero, the MBS must be a multiple of SCR.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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 cannot be empty.
384	cve LPortDelSNameDefined	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\nnot of type FR UNI or FR NNI.
386	cve LPortModDlciDefined	The logical port cannot be designated as a 'backup' because a circuit is defined on it.
387	cve LPortModSNameDefined	The 'backup' designation cannot be removed from the logical port because a service name is defined on it
388	cve LPortPrimaryForSName	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\nthan 32 or greater than 2016.
394	cveCktScrInvalidDs3	For circuits on DS3 ATM cards, SCR cannot be less than PCR/64\nor greater than PCR.
395	cveCktBurstTolInvalidDs3	For circuits on DS3 ATM cards, the MBS cannot be less than 1 or greater than 2016.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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	cve LPortBuNotOkForPrimary	The selected backup logical port cannot backup the defined primary logical port.
399	cve LPortNotPrimary	A backup logical port cannot be bound to a service name as a primary logical port.
400	cve LPortNotBackup	The logical port must be designated as a 'backup' before it\ncan be bound to a service name as a backup logical port.
401	cveCkt LPortIsBackup	A circuit cannot be defined on a logical port designated as a backup.
402	cveMcastDlci LPortIsBackup	A multicast DLCI cannot be defined on a logical port designated as a backup.
403	cveMgmtDlci LPortIsBackup	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	cveBu LPortOnFROnly	The logical port cannot be designated as a 'backup' because it is not of type FR UNI or FR NNI.
406	cveNoAtmFeeder LPort	Cannot add the current logical port when there is\nno ATM feeder logical port defined.
407	cveOneCktOnFrad LPort	Cannot define more than one PVC on an Encapsulation FRAD Logical port or PPP to 1490 Logical port.
408	cveE1CardExtlClockValue	The external clock rate must be a multiple of 8Kps and cannot be higher than 2.048Mbps.
409	cveInvalid LPortBw	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\nif Virtual Path ID is 0.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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	cveSyncCpFirst	CP/SP/NP is out of synchronization. Synchronize the CP/SP/NP first, then synchronize this IOP/IOM/BIO.
417	cveNoCpFound	Error finding CP/SP from the database.
418	cveQoSCbrBwPercentNeg	CBR QoS fixed bandwidth allocation percentage cannot be less than zero.
419	cveQoSVbrRTBwPercentNeg	VBR Real Time QoS fixed bandwidth allocation percentage cannot be less than zero.
420	cveQoSVbrNRTBwPercentNeg	CBR Non-Real Time QoS fixed bandwidth allocation percentage cannot be less than zero.
421	cveQoSUbrBwPercentNeg	UBR QoS fixed bandwidth allocation percentage cannot be less than zero.
422	cveQoSBwAllocExd100Percent	The total of all QoS class bandwidth allocation percentages cannot exceed 100 percent.
423	cve LPortInvalidUniType	The UNI type is invalid.
424	cve LPortTooManyVpiVciBits	The total number of valid VPI and VCI bits must be 14 or less.
425	cve LPortTooFewVpiBits	The number of valid VPI bits cannot be less than zero.
426	cveLPAtmProtocolConnTypeInv	Connection type must be \"network <-> network\" if the ATM protocol is IISP 3.1.
427	cve LPortInvalidBwPolicing	The bandwidth policing option is invalid.
428	cve LPortConnTypeInv	The connection type is invalid
429	cve LPortInvalidIlmiAdminStatus	The ILMI admin status is invalid.
430	cve LPortInvalidIlmiPollPeriod	The ILMI poll period value is invalid. Valid range is 1 - 255.
431	cve LPortInvalidIlmiLossThresh	The ILMI loss threshold value is invalid. Valid range is 1 - 255.
432	cve LPortInvalidSigAdminStatus	The signaling admin status is invalid.
433	cve LPortInvalidIlmiPfxScreen	The ILMI DTE prefix screen mode is invalid.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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	cve LPortInvalidOamAlmTimerThresh	The OAM alarm timer threshold value is invalid. Valid range is 1 - 60.
439	cve LPortInvalidSigTuningValue	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\nless than 2 to the power N where N is the number of\nVCI 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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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	cve LPortDChanIsdnPriDis	The ISDN D-Channel logical port cannot be added because its physical\nport does not have ISDN PRI enabled
461	cve LPortDChanMustBeFirst	The ISDN B-Channel logical port cannot be added because there is no ISDN D-Channel logical port defined on its physical port.
462	cve LPortTooManyIsdnDs0s	A single DS0 must be selected for an ISDN logical port.
463	cve LPortInvalidDChanDs0	An ISDN D-Channel logical port must be defined on DS0 24 only.
464	cve LPortInvalidBChanDs0	An ISDN B-Channel logical port must be defined on one of DS0 1 through 23 only.
465	cve LPortDelDChan	The ISDN D-Channel logical port cannot be deleted because at least one ISDN B-Channel\nlogical port is still defined on its physical port.
466	cve LPortNoDChanTypeChg	The logical port type of an ISDN D-Channel logical port cannot be changed.
467	cveIsdnBChan LPortNotFound	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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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\non the card is set to one.
474	cve LPortDChanNotOnPPort1	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	cve LPortLmiDelayInvalid	Lmi Update Delay value is invalid.
480	cveModNmsEntryNeedPramSync	NMS entries cannot be changed dynamically\nin the switch. They can only be maintained\nthrough PRAM synchronization or\nscript 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	cveNoSrc LPort	A source Logical Port must be selected.
486	cveNoDes LPort	A Destination Logical Port must be selected.
487	cveSameSrcDes LPorts	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\nits 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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
495	cveNULLPramFile	Pram file cannot be empty.
496	cveModifyLB LPort	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	cve LPortInvalidIlmiVpi	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	cve LPortInvalidIlmiVci	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\nVCI bits valid for this logical port.
504	cveCktCellBasedPcrTooHigh	The value (PCR * 53 * 8) must be no higher than the physical port bandwidth.
505	cve LPortInvalidUpc	The UPC function option is invalid.
506	cveInvalidAtmTrkVpiId	VPI must be greater than 0 and less than 2 to the power of N, \nwhere 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	cve LPortInvalidAtmTrkVpiVci	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	cve LPortTooFewVciBits	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 effective bandwidth below that which is required by the member logical ports.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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\nVCI bits valid for the logical port.
513	cveInvalidMgmtVpiVciName	Unable to obtain configuration information for the Management VPI/VCI.
514	cve LPortCdvTooSmall	Logical port cell delay variation cannot be less than 0.
515	cve LPortInvalidDs3TrkCdv	Invalid DS-3 logical port direct trunk cell delay variation.
516	cve LPortInvalidOc3TrkCdv	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	cve LPortPvcUsingVpiSpace	The number of valid VPI bits cannot be reduced below the value needed to support the largest PVC VPI on the logical port.
523	cve LPortPvcUsingVciSpace	The number of valid VCI bits cannot be reduced below the value needed to support the largest PVC VCI on the logical port.
524	cve LPortTrkUsingVpiSpace	The number of valid VPI bits cannot be reduced below the value needed\nto support the largest OPTimum trunk VPI in use on this logical port.
525	cve LPortTrkUsingVciSpace	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	cveTrunk LPortInvalid	The specified logical port types cannot support a trunk.
527	cveTrunk LPortTypeDiff	The specified logical ports are not of the same type.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
528	cveTrunk LPortBwDiff	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	cveVpiVciOfFeeder LPortDup	The entered VPI/VCI is being used by the feeder logical port in the physical port.
539	cveVpiOfVpcFeeder LPortDup	The entered VPI is being used by a VPC on the feeder logical port in the physical port.
540	cveVpiOfFeeder LPortDup	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	cve LPortInvalidCongThresh	Severe Congestion Thresholds specify invalid settings.
545	cveCardInvBulkStatsSettings	Invalid Bulk Statistics Settings.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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	cve LPortTooManyVpiBits	The number of valid VPI bits cannot be greater than 8.
553	cve LPortTooManyVciBits	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	cve LPortInvalidSvcRanges	The SVCC VPI/VCI Ranges entered are invalid.
556	cve LPortSvcPvcRangeError	The specified SVCC VPI/VCI Ranges are not within the VCC VPI/VCI Ranges.
557	cve LPortSvcRangeConflict1	The SVCC VPI/VCI Ranges entered are in conflict with those used by another logical port or trunk.
558	cve LPortSvcRangeConflict2	The VPI entered is in conflict with those\nused by another logical port or trunk.
559	cve LPortDelVUNIFeeder	A feeder port cannot be deleted as long as one or more Virtual UNI \nlogical ports are defined on the same physical port.
560	cveCktVpiVciConflictIlmi1	Invalid VPI/VCI due to conflict with ILMI VPI/VCI Settings\nof Logical Port Endpoint 1.
561	cveCktVpiVciConflictIlmi2	Invalid VPI/VCI due to conflict with ILMI VPI/VCI Settings\nof Logical Port Endpoint 2.
562	cve LPortVpiVciConflictIlmi	Can not enable (Signalling or ILMI) because the VPI/VCI is already in use.
563	cve LPortSvcVpcUsingVpiSpace	The number of valid VPI bits cannot be reduced below the value needed\nto support the largest SVCC VPI/VCI Range VPC VPI on this physical port.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
564	cve LPortSvcVccUsingVpiSpace	The number of valid VPI bits cannot be reduced below the value needed to support the largest SVCC VPI/VCI Range VCC VPI on this physical port.
565	cve LPortSvcVccUsingVciSpace	The number of valid VCI bits cannot be reduced below the value needed to support the largest SVCC 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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
582	cveSvcCUGMRBadE164Digit	The CUG member rule E.164/X.121 regular expression can only contain\ndecimal 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\nhexidecimal 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/X.121 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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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	cveOverMaxSvcSecScnPer LPort	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	cve LPortSecScnsBound	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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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 setting entered are invalid. At least one attribute should be enabled.
621	cveSvcAddressAttrInvalid	The SVC Address Attribute settings entered are invalid.
622	cveSvcAddressRegNotAllowed	SVC Address Registration is not allowed for this prefix configuration.
623	cve LPortSvcUsingVpcVpiSpace	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	cve LPortSvcUsingVccVpiSpace	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	cve LPortSvcUsingVccVciSpace	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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
626	cve LPortInvalidPvcRanges	The VUNI PVC Address Range entered are invalid.
627	cve LPortVuniUsingVpiSpace	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 4096 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	cve LPortDefinedAtmVpi	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.
638	cveCreateBackupVPIError	To create a backup port, the number of VPI and VCI bits on the backup must be equal to those on the primary.
639	cveCreateBackupVCIError	To create a backup port, the number of VPI and VCI bits on the backup must be equal to those on the primary.
640	cveRevertBackupVPIError	To revert a backup port, the number of VPI and VCI bits on the backup must be equal to those on the primary.
641	cveRevertBackupVCIError	To revert a backup port, the number of VPI and VCI bits on the backup must be equal to those on the primary.
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).

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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	cveSvcCUGMRE164BadLen	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 * 53 * 8) must be no higher than the physical port bandwidth.
655	cveMod LPortNegativeBandwidth	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.
656	cveSvcIMLPAlreadDefined	Only single IMLP can be enabled per switch.
657	cveSvcIMLPAddrOverMaxLen	Node Prefix address length too long\nMaximum length of node prefix with IMLP enabled is 13 byte.
658	cveSvcIMLPInvalidAddr	With IMLP enabled, Valid address are DCC AESA, ICD AESA, E.164 AESA or Custom AESA.
659	cveTrkOspfIdInSvcNodePrefix	Can not delete trunk since its area ID is referenced in SVC node prefix.
660	cveModOspfIdInSvcNodePrefix	Can not modify trunk Area ID since its area ID is referenced in SVC node prefix.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
661	cveInvalidSvcAddrScope	Invalid Svc Address organizational scope.
662	cvxSvcSameE164X121Address	Same E.164 and X.121 address defined under same Node or same port.
663	cveSvcSecScnX121BadLen	The security screen address X.121 regular expression cannot contain more than 14 characters.
664	cveSvcCUGMRX121BadLen	The CUG member rule X.121 regular expression cannot contain more than 14 characters.
665	cveSvcOSPFRouteDetDisabled	Node Prefix OSPF attrbite only can be enabled if route determination attribute is also enabled.
666	cveSvcOSPFIdNotFound	Invalid OSPF Area Id, Select the right OSPF Area Id or disable OSPF attribute.
667	cveCktVpiConflictIlmi1	Invalid VPI due to conflict with ILMI or Signalling VPI Settings of Logical Port Endpoint 1.
668	cveCktVpiConflictIlmi2	Invalid VPI due to conflict with ILMI or Signalling VPI Settings of Logical Port Endpoint 2.
669	cve LPortIlmiVpiVciConflict	The ILMI VPI and/or VCI conflict with another enabled ILMI VPI/VCI of the logical port on the same physical port. You may either disable ILMI admin status or change VPI or VCI of either logical 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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
711	cvePPortShaperPcrExceedClockSpeed	The peak cell rate exceeds the port's data rate.
712	cvePPortShaperScrHigherThanPcr	The sustainable cell rate is higher\nthan the corresponding peak cell rate.
713	cveNewCardTypeNotSupported	The specified card type is not supported by this switch version.
714	cveOneCktOnAAL1 LPort	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 (an ATM UNI port residing on an ATM IWU card) can only be connected to another AAL1 User Port if the connected cards are of the same type.
716	cve LPortBwExceedPPortDataRateShaperChange	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	cveVpiBitsChangeWith LPortsDefined	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 X.121 address must be 1.14 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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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.
735	cveInvSvcFmtFor LPort	Invalid address format for this LPort type.
750	cveInvalidTrkAdminCost	Trunk admin cost must be within the range of 1 to 65534.
751	cveXfaceMismatch	+ Interface Mismatch.
760	cveInvalid LPortOversub	QoS oversubscription factors must be between 100% and 10,000%.
761	cveInvalid LPortOversubOc12	QoS oversubscription factors must be between 100% and 2,700%.
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	cveOne LPortInDs0Lb	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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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 or Country Code 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	cveDcePVTLessThanDtePI	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	cveMax LPortsPerChDs3Exceeded	Maximum number of logical ports exceeded per ChDs3 channel.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
802	cve LPortIdChanInUse	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 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	cve LPortDelFeederChan	A feeder port cannot be deleted as long as one or more OPTimum trunks \nof the same service type are defined on the same channel.
810	cve LPortBandwidthTooHighChan	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 Closed 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	cveAddrInOwn LPort	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 Management Path.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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 Closed 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 charracters.
836	cveInvalidTimedOutValue	Timeout value must be between 1 to 10 seconds.
837	cveIsdnAuthenNoAuthenDomain	The modified Authentication Domain does not exist.
838	cve LPortInvalidDChanTS16	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 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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
843	cvePramULIsdnAuthNoSuchPPort	Unable to construct the ISDN AuthenDomain Option from PRAM because its owning physical port does not exist.
844	cvePramULIsdnAuthNoSuch LPort	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	cveMgmtDlciDefinedOn LPort	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\nmust 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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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	cveNoMemorySerialNoSwitchSwRev No	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	cveSpvcDeleteConfirm	About to DELETE the selected Soft PVC. Are you sure?
873	cveSpvcLeafDeleteConfirm	About to DELETE the selected Soft PVC Leaf. Are you sure?
874	cveConnIdInUse	The entered Connection ID is being used by a PVC, SPVC or another SVC Address..
875	cve LPortDelSvcAddress	The logical port cannot be deleted as long as an SVC Address is associated with it.
876	cve LPortDelSvcPrefix	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	cve LPortDelSvcUserPart	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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
883	cveSvcHasCleared	The SVC has cleared.
884	cveSharedThreadLPBwExceeded	The IOM upon which this logical port is being configured shares a common switching\nfabric 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,\nand 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\nthread 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	cve LPortPvcRangeVPIError	The specified VPI Stop is not within the VCC VPI 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 -1 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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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	cveCkt LPortDirectUniVpiError	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	cveCkt LPortVirtualUniVpiError	The specified VPI of a Virtual Uni Endpoint is not within its VPI Start/Stop Range.
902	cve LPortCktVUniDirectVpiError	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	cve LPortCktVUniSelfVpiError	The specified VPI Start/Stop Range is invalid. The Virtual Uni is used as a circuit endpoint\nusing a VPI within the old VPI Start/Stop Range.
904	cveCfm_CktEPDOnlyAAL5Traffic	A Frame Discard Enabled or FCP Discard = EPD circuit must carry only AAL5 traffic. Otherwise, non-AAL5 traffic that was not intended for discard may be discarded. Do you wish to continue?
905	cveNoSmdsCountryCodeDefined	No Country code has been defined yet.
906	cveMvCktNoCircuitsOnSrc LPort	No Circuits defined on the selected source logical port.
907	cveInvalidIsdnNegotiationTime	Invalid Negotiation time specified. Valid range for the value is 0-99999.
1000	cveSubCardAdminTypeInv	The subcard type is invalid.
1002	cveSubCardAdminStatusInv	The subcard admin status is invalid.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
1003	cveSubCardCardMismatch	The selected subcard type is not supported on this type of switch and/or card.
1004	cveSubCardOc48NotSubSlot1	Subcard types capable of OC-48 date rates can only be configured in subslot A.
1005	cveRestSubSlot4NotEmpty	A subcard cannot be configured in subslot D of this slot.
1006	cveSubCardOc48Rest	Subcard types capable of OC-48 date rates cannot be configured in this slot.
1007	cveSubCardNotFound	The subcard does not exist.
1008	cveSubCard LPortsExist	A subcard's type cannot be changed when there are logical ports defined on the subcard.
1009	cveSubCardTypeChgToNonEmpty	Changing directly from one subcard type to another is not allowed. The subcard type must first be set to "Empty."
1010	cveSubCardOc48SubSlot1Only	If subslot A contains a subcard capable of OC-48 data rates, the remaining subcards on the card must be set to "Empty."
1011	cveInvBioSubCardCount	A BIO card must contain four subcards.
1012	cveInvNpSubCardCount	An NP card cannot contain any subcards.
1013	cveInvSysClkPortRef1	System clock port reference one refers to a non-existant physical port on the subcard.
1014	cveInvSysClkPortRef2	System clock port reference two refers to a non-existant physical port on the subcard.
1015	cveInvSfRedSetting	The SF redundancy setting is invalid.
1016	cveInvTmRedSetting	The TM redundancy setting is invalid.
1017	cveSwSoftwareDownloadFailed	The download of the switch software file has failed.
1018	cveSubCardHasAnyApsEnabled	The card type cannot be changed because one or more physical ports are part of an APS pair.
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.
1100	cveCfm_ChgAtmSvcVpiVci	Do you wish to reset all the SVC VPI/VCI ranges to the specific default values?

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
1101	cveCfm_CommitSubPort	Changes to subport attributes must be applied before setting logical port attributes. Press Apply to apply changes.
1987	cveDb_PERM_DENY	Permission denied; The network resource that you are trying to access does not belong to one of your groups.
1988	cveDb_TRAN_LOG_FULL	NavisCore 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.
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 \nbe 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	cveCfm_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?

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
2036	cvePPortLowMaxPCRWhenPLCP	The Peak Cell Rate values used by one of the configured logical port(s) \nis 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	cveDLCIOfFeeder LPortDup	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 NavisCore.
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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
2056	cveIsdnAaMatchBChan	ISDN B-Channel E.164 address should be different from any Authentication address.
2057	cveDeleteFarEndLB LPort	Cannot delete a logical port while the logical port is in loopback state.
2058	cve LPortDChan1MustBeFirst	The ISDN B-Channel logical port cannot be added because there is no ISDN D-Channel logical port defined on physical port one.
2059	cveTrunk LPort1InUse	The logical port 1 is already in use.
2060	cveTrunk LPort2InUse	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\non 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 4263 for a switch.
2072	cvePPortUIONo4MgAdjaUsed	Current port cannot be configured at 4Mbps or greater since the adjacent port is used.
2073	cveTrunkOutOfRangeKAError Threshold	Keep alive error threshold is out of range. Valid range is 3-255.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
2074	cve LPortCongCtrlInvalid	Congestion control values must be in the range of (mild < severe < absolute).
2075	cve LPortCongCtrlMaxExceeded	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	cveCktRRedFramePercentShould BeZero	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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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\ncannot 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 seconds.
2108	cveInvalidTrkRestThreshld	Trunk restoration threshold cannot be less than 2 seconds.
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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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	cveDelCustomerInUseBy LPort	Cannot delete customer. It is still referenced by an LPort.
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	cveDelVPNInUseBy LPort	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 between Switch Processor 10 and Switch Processor 20/30/40 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?

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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	cveCktInvalidBwPrio16	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	cveAtmCs LPortBandwidthExceeds LPortScr	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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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 8000 for a card on a 500 switch and 16000 for a card on a 550 switch.
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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
2171	cvePPortInvalidCircuitType	Invalid circuit type for this port type.
2172	cvePPortInvalidInBandLineLoopback Code	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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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	cveSESThresholdSetting Modify_Warning	Changing the SES Threshold Setting will affect how the SES counts\nare calculated and it will invalidate interval and total data\ncollected to this point. Are you sure?
2198	cve LPortInvalidOc12TrkCdv	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	cve LPortMildThrhldEqZero	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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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	cve LPortZeroCodingNx56	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	cveInvalidSwitchNameBadChars	Switch name can ONLY contain alphanumeric characters and underscores.
2216	cveSmdsAddr LPortError	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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
2226	cveNoCustForTrunk	Trunk does not contain any selected customer name.
2227	cve LPortTooManyVpiVciBitsCsIwu	The total number of valid VPI and VCI bits must be 12 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	cveRipSendHostRoutesOnStateIpAddr Missing	Cannot Set Send Host Routes on. You must first enter an Ethernet IP Address\nto 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	cve LPortInvalidIlmiVci0_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	cveMgmtVpiVciDefinedOn LPort	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. Use "OK" to delete the card if it is in-sync. Use "Cancel" otherwise.
2240	cveCascadeViewAttrs	Please set NavisCore Object Attributes.
2241	cveMoveLoopbackCkt	This circuit is set to loopback, cannot be removed.
2242	cveHpMapObjInvalid	A database inconsistency has been detected in the HP OpenView database. Cannot locate Ascend Network Information in the HP Openview database. Contact the NMS administrator to help isolate the source of corruption.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
2243	cveCktInvalidBwPrio4	Invalid circuit bandwidth priority value (0 - 3).
2244	cveSecondIPAddrInUse	Cannot modify second network number when it is currently in use by a switch.
2245	cveSecondIPAddrConflict	Cannot accept new secondary network number. There exists a network in the database which already uses the same value.
2246	cveClusterNameIsEmpty	Cluster name can not be empty.
2247	cveT1E1FifteenMinThreshUaspInvalid	Fifteen Minute UAS-P threshold value must be within the range of 1 to 63.
2248	cveClusterIdIsEmpty	Cluster ID can not be empty.
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 milliseconds.
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	cveModCcrmProtocolIDNeedPram Sync	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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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	cveTotal LPortBuffersExceeded	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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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	cveInvalid LPortClp01Threshold	The Clp0+1 value must be less than the Port Buffers configured for this logical port.
2281	cveInvalid LPortDiscardThreshold	The Discard value must be less than the CLP0+1 value configured for this logical port.
2282	cveInvalid LPortEfciThreshold	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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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.
2302	cveTrunk LPortServiceClassDiff	The specified logical ports does not belong to same QoS service class.
2303	cveModTrunk LPortServiceClassDiff	The modification of Trunk configuration is not allowed unless both endpoint logical ports belong to the same QoS service class.
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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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\nin the switch. They can only be maintained\nthrough PRAM synchronization.
3015	cveInvalidRedundancyCombination	APS 1+1 Redundancy is not available for this card type.
3016	cve LPortsExistOnProtectionPort	Redundancy cannot be changed 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\nin the switch. It can only be maintained\nthrough PRAM synchronization.
3022	cveModNrtsManageVBRnrtNeedPram Sync	Manage VBRnrt Traffic cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization.
3025	cveLossOfQoSGuarantee_warning	Warning: Turning off packet segmentation will cause loss of Qos guarantee.
3026	cveNotEnoughDS0sForServiceClasses	Not enough DS0s are selected for supporting all the service classes. At least 4 DS0s are required.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
3027	cveCircuitDefinedForServiceClass	There is a circuit defined for one of the service classes being removed. Delete the circuit first.
3028	cveCktServiceClassNotSupported	The end point logical port does not support the specified service class.
3029	cveCktInvalidAvgFrameSize	The specified average frame size (fwd/rev) is out of range. Valid range is 3 - 8192.
3030	cvePPortInvalidFeacLoopback	Invalid FEAC Loopback setting for this port type.
3031	cveAlarmFailureOutOfRange	The Alarm Failure setting is out of range. Valid range is 0 - 65535.
3032	cveAlarmClearOutOfRange	The Alarm Clear setting is out of range. Valid range is 0 - 65535.
3033	cveSpvcRetryLimitInvalid	The Retry Limit value must be within the range of 0 to 65535.
3034	cveSpvcRetryIntervalInvalid	The Retry Interval value must be within the range of 0 to 3600.
3035	cveModMulticastRateNeedPramSync	The Multicast Rate cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization.
3036	cveInvalidNrtsParent	The Atm FCP cannot be enabled on this card.
3037	cveQosDelayIntPrd	The interval period must be within the range of 4 - 1440.
3038	cveMLFRBindingExists	This ML Member LPort is already bound to a MLFR Trunk Bundle LPort.
3039	cveMLFRExceededMaxBindings	A MLFR Trunk Bundle LPort can not have more than 32 ML Members bound to it.
3040	cveMLFRBindingNotFound	The MLFR Member LPort binding does not exist.
3041	cveMLFRCantDeleteBundle	This MLFR Trunk Bundle LPort can not be deleted because it has bound ML Members.
3042	cveMLFRCantDeleteMember	This ML Member LPort can not be deleted because it is bound to a MLFR Trunk Bundle LPort.
3043	cveMLFRCantUnbindMember	This ML Member LPort can not be unbound because it is the only bound LPort of a MLFR Trunk.
3044	cveMLFRUnbindMemberWarning	The unbinding of this ML Member results in an unequal number of bindings on each end of A MLFR trunk. Do you wish to continue ?

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
3045	cveMLFRTrunkEndpointError	A MLFR Trunk Bundle LPort must contain at least one bound ML Member LPort.
3046	cveMLFRUnevenTrunkEndpoints	The MLFR Trunk Bundle endpoints contain a different number of bound ML Members. Do you wish to continue?
3047	cveMLFRBindMemberWarning	The binding of this ML Member results in an unequal number of bindings on each end of A MLFR trunk. Do you wish to continue?
3048	cveInvalidCollectorname	Collector Name cannot be NULL or more than 65 characters in length.
3049	cveBulkStatsCollectorDupIp	A Collector with same IP address already exists.
3050	cveBulkStatsCollectorDupName	A collector with same name already exists.
3051	cveBulkStatsCollectorNoAddress Change	The Collector Address cannot be modified.
3052	cveCktInvalidAdminCost	The admin cost of the circuit must be within the range of 1 to 4294967295.
3053	cveApsNotEnabled	APS must be Enabled on the PPort screen before changing these parameters.
3054	cveCfm_CommitISDNswtype	The change in ISDN CO Switch Type will cause the card to be rebooted. Are you sure?
3055	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.
3056	cve LPortNotSelected	A logical port must be selected in order to do this operation.
3057	cveIwuCs LPortTooManyVpiVciBits	The total number of VPI and VCI bits must not exceed 12.
3058	cveIwuCs LPortTooFewVciBits	The number of VCI bits has to be a minimum of 6 and cannot exceed 12.
3059	cveIwuCs LPortTooManyVciBits	The number of VCI bits has to be a minimum of 6 and cannot exceed 12.
3060	cveIwuCs LPortTooManyVpiBits	The number of VPI bits can vary from 0 through 5.
3061	cveIwuCs LPortTooFewVpiBits	The number of VPI bits can vary from 0 through 5.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
3062	cveModDisableFcpStatusNeedPram Sync	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 ?
3064	cveBadCllmThrhldNone	Invalid CLLM ThresholdNone value. Valid range is 1-100.
3065	cveBadCllmThrhldMild	Invalid CLLM ThresholdMild value. Valid range is 1-100 and greater than CLLM ThresholdNone value.
3066	cveInvalidNodePoll	Invalid Node Poll.
3067	cveEthernetAddressInUse	Ethernet address defined matches an IP\naddress already in use in the switch.
3068	cveSvcNetworkIdNot	Network IDs cannot be created because the maximum configurable addresses for this card has been reached.
3069	cveSvcNetworkIdInvalidCIC	The Carrier Id Code is out of range. Valid values are 1 - 8 digit integers.
3070	cveSvcNetworkIdInvalidDNIC	The Data Network Id Code is out of range. Valid values are 4 digit integers.
3071	cveSvcNetworkIdInvalidAdminCost	The Admin Cost is out of range. Valid values are 0 - 65,535.
3072	cveSvcNetworkIdNotUnique	The Network ID objects already exists for this LPort.
3073	cveSvcNetworkIdRouteDetAlreadyExis ts	A Network ID with Route Determination enabled can not be created for this. LPort because an Svc Address or Prefix with Route Determination enabled already exists.
3074	cveSvcNetworkId LPortNIANotDelete	The logical port cannot be deleted because there is a Network ID defined on it.
3075	cveSvcNetworkIdNotFound	The SVC Network ID object was not found.
3076	cveSvcNetworkIdAttrInvalid	The SVC Network ID Attribute settings entered are invalid.
3077	cveSvcNetworkIdRouteDefExists	This Network ID cannot be set as the adjacent network because adjacent network has already been set.
3078	cveSvcNetworkIdSourceDefExists	This Network ID cannot be set as the source default because source default has already been set.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
3079	cve LPortsExistOnPrimaryPort	Redundancy cannot be changed because the primary port has logical ports defined on it.
3080	cveHoldOverInvalid	Holdover Mode must be in the range of 1- 2.
3081	cveCktFrameDiscardInvalid	Frame Discard Status is invalid.
3082	cveSpvcInvalidVpi	The target VPI is out of range. Valid values are 0 - 4095.
3083	cveSpvcInvalidVci	The target VCI is out of range. Valid values are 0 - 65535.
3084	cveInvalidMacAddr	Invalid MAC address.
3085	cve LPortCelTrkUsingVpiSpace	The number of valid VPI bits cannot be reduced to support the largest cell optimum trunk VPI Range on this physical port.
3086	cveInvalidCelTrkVpiVal	Invalid cell trunk LPort VPI value (VPI should be greater than or equal to zero and VPI Start should be equal or less than VPI Stop).
3087	cveInvalidCelTrkRmtVpiVal	Invalid cell trunk LPort MPT VPI value (MPT VPI should be 0 or between VPI Start+1 and VPI Stop).
3088	cveCellOptTrkVpiUsed	Cell trunk LPort VPI overlap over the same pport.
3089	cveCfm_Delete	About to DELETE the selected entry. Continue?
3090	cveInHECErrorUTimInvalid	TCA: Monitor Period - Ingress HEC Error Uncorrectable is out of range. Valid range is 1-30.
3091	cveInHECErrorUThrInvalid	TCA: Threshold Value - Ingress HEC Error Uncorrectable is out of range. Valid range is 1-10.
3092	cveEBufOverflowCBRTimInvalid	TCA: Monitor Period - Egress Buffer Overflow CBR is out of range. Valid range is 1-30.
3093	cveEBufOverflowCBRThrInvalid	TCA: Threshold Value - Egress Buffer Overflow CBR is out of range. Valid range is 1-10.
3094	cveEBufOverflowABRTimInvalid	TCA: Monitor Period - Egress Buffer Overflow ABR is out of range. Valid range is 1-30.
3095	cveEBufOverflowABRThrInvalid	TCA: Threshold Value - Egress Buffer Overflow ABR is out of range. Valid range is 1-10.
3096	cveEBufOverflowVBR1TimInvalid	TCA: Monitor Period - Egress Buffer Overflow VBR-RT is out of range. Valid range is 1-30.
3097	cveEBufOverflowVBR1ThrInvalid	TCA: Threshold Value - Egress Buffer Overflow VBR-RT is out of range. Valid range is 1-10.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
3098	cveEBufOverflowVBR2TimInvalid	TCA: Monitor Period - Egress Buffer Overflow VBR-NRT is out of range. Valid range is 1-30.
3099	cveEBufOverflowVBR2ThrInvalid	TCA: Threshold Value - Egress Buffer Overflow VBR-NRT is out of range. Valid range is 1-10.
3100	cveIomTcaIngBufOflwTimInvalid	TCA: Monitor Period - Ingress Buffer Overflow is out of range. Valid range is 1-120.
3101	cveIomTcaIngBufOflwThrInvalid	TCA: Threshold Value - Ingress Buffer Overflow is out of range. Valid range is 1-65535.
3102	cveIomTcaIngInvVpiVciTimInvalid	TCA: Monitor Period - Ingress Invalid Vpi/Vci is out of range. Valid range is 1-120.
3103	cveIomTcaIngInvVpiVciThrInvalid	TCA: Threshold Value - Ingress Invalid Vpi/Vci is out of range. Valid range is 1-65535.
3104	cveNoModClassOfSrvMcastCkt	Cannot modify the Class Of Service of a circuit which is associated with a multicast DLCI
3105	cveMaxBufferSizeRangeInvalid	Maximum buffer size value must be between 64 and 8192 inclusive.
3106	cveInvalidCllmInterval	Invalid CLLM Interval value. Valid range is 5-30.
3107	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?
3108	cveLANIdleTimeoutOutOfRange	The LAN Idle Timeout value must be between 0 - 900.
3109	cveSvcFrameDiscardInvalid	The Frame Discard value is invalid.
3110	cve LPortInvalidCtrlUpc	The Control UPC Function option is invalid.
3111	cveCfm_CommitSwitchReboot	Performing this action will cause the switch to be rebooted. Are you sure?
3112	cveSwitchObjNameIsNull	Switch Object Name is NULL in MaintenanceDlgData.
3113	cveCvListReadFailed	cvList Read is Failed Due to NameSList is NULL.
3114	cveMustAssignFwdAndRevTfds	A Forward and a Reverse Traffic Descriptor must be assigned.
3115	cveSvcSecScnCallingAddrTypeInvalid	The Calling Address Type of the security screen is invalid.
3116	cveSvcSecScnCalledAddrTypeInvalid	The Called Address Type of the security screen is invalid.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
3117	cveZeroCirQoSClassRestriction	A Zero CIR enabled circuit must have UBR QoS service class in the corresponding direction.
3118	cveCardTypeDoesNotSupportIntra CardApsRedundancy	Intra-Card APS 1+1 is not supported on this card type.
3119	cveCardTypeDoesNotSupportInter CardApsResUniRedundancy	Inter-Card APS Resilient UNI is not supported on this card type.
3120	cveInvalidOcApsOnePlusOnePrimary Port	Intra-Card APS 1+1 cannot be enabled on this port.
3121	cveRedundancyChgToDisabled	Changing directly from one protection port to another is not allowed. Redundancy must first be set to "None" to disable it.
3122	cveInvalidSlotPortId	Invalid protection port.
3123	cveProtectionPortOnSameCard	The protection port must be on a different card of the same type for Inter-Card APS Resilient UNI.
3124	cveInvalidOc12InterCardApsResUni PortId	Invalid protection port. Oc-12 Port 2 cannot be a primary or backup port\nfor Inter-Card APS Resilient UNI or Inter-Card APS with Trunk Backup.
3125	cveMustDisableApsBeforeChange	Changing directly from one type of Redundancy to another is not allowed. Redundancy must first be set to "None" to disable it.
3126	cveProtectionPortDoesNotExist	Invalid Protection Port. The Protection Port must be a valid port on another card of the same type.
3127	cveProtectionPortApsOnePlusOneDoes NotExist	Invalid Protection Port. The Protection Port must be a valid port on the same card.
3128	cveProtectionTypeMismatch	Invalid Protection Port. The Protection Port must be the same type of port as the Primary Port, i.e., an OC3 port can be a backup for another OC3 port.
3129	cveInvalidApsOnePlusOnePrimaryPort	Invalid Primary Port.
3130	cveCardHasRedundancyEnabledOnPort	The card type cannot be changed because one or more physical ports are part of an Inter-Card APS Resilient UNI pair.
3131	cveProtectionPortAlreadyRedundant	Invalid Protection Port. The protection port is already part of an APS-pair.
3132	cveInvalidApsOnePlusOneDisabled Port	Intra-Card APS 1+1 can only be Disabled from the Primary Port.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
3133	cveInvalidInterCardApsResUniDisabledPort	Inter-Card APS Resilient UNI can only be Disabled from the Primary Port.
3134	cveInvalidDirection	Invalid Direction. Unidirectional mode is only valid for Intra-Card APS 1+1.
3135	cveLimitVirtualLinkCard	The Virtual Link limit per card has been reached. Additions may not be made.
3136	cveWillExceedLimitVirtualLinkCard	The Virtual Link limit per card will be exceeded. Reduce the number of Traffic Descriptors defined on the Control Channels (Ilmi, Signaling).
3137	cvNoMoreObjects	No more objects in the database.
3138	cveoverflowCbrPeriodInvalid	TCA: Monitor Period - Overflow - CBR is out of range. Valid range is 1 - 120.
3139	cveoverflowCbrThreshInvalid	TCA: Threshold Value - Overflow - CBR is out of range. Valid range is 1 - 65535.
3140	cveoverflowAbrPeriodInvalid	TCA: Monitor Period - Overflow - ABR is out of range. Valid range is 1 - 120.
3141	cveoverflowAbrThreshInvalid	TCA: Threshold Value - Overflow - ABR is out of range. Valid range is 1 - 65535.
3142	cveoverflowVbr1PeriodInvalid	TCA: Monitor Period - Overflow - VBR1 is out of range. Valid range is 1 - 120.
3143	cveoverflowVbr1ThreshInvalid	TCA: Threshold Value - Overflow - VBR1 is out of range. Valid range is 1 - 65535.
3144	cveoverflowVbr2PeriodInvalid	TCA: Monitor Period - Overflow - VBR2 is out of range. Valid range is 1 - 120.
3145	cveoverflowVbr2ThreshInvalid	TCA: Threshold Value - Overflow - VBR2 is out of range. Valid range is 1 - 65535.
3146	cvecongestCbrPeriodInvalid	TCA: Monitor Period - Congestion - CBR is out of range. Valid range is 1 - 120.
3147	cvecongestCbrThreshInvalid	TCA: Threshold Value - Congestion - CBR is out of range. Valid range is 1 - 65535.
3148	cvecongestAbrPeriodInvalid	TCA: Monitor Period - Congestion - ABR is out of range. Valid range is 1 - 120.
3149	cvecongestAbrThreshInvalid	TCA: Threshold Value - Congestion - ABR is out of range. Valid range is 1 - 65535.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
3150	cvecongestVbr1PeriodInvalid	TCA: Monitor Period - Congestion - VBR1 is out of range. Valid range is 1 - 120.
3151	cvecongestVbr1ThreshInvalid	TCA: Threshold Value - Congestion - VBR1 is out of range. Valid range is 1 - 65535.
3152	cvecongestVbr2PeriodInvalid	TCA: Monitor Period - Congestion - VBR2 is out of range. Valid range is 1 - 120.
3153	cvecongestVbr2ThreshInvalid	TCA: Threshold Value - Congestion - VBR2 is out of range. Valid range is 1 - 65535.
3154	cveInvalidPrefix	The selected Prefix format is invalid for a SPVCs.
3155	cve LPortInvalidSvcBwPrio16	Invalid SVC bandwidth priority value (0 - 15).
3156	cve LPortInvalidSvcBwPrio4	Invalid SVC bandwidth priority value (0 - 3).
3157	cve LPortInvalidSvcBumpPrio	Invalid SVC bumping priority value (0 - 7).
3158	cve LPortInvalidSvcQoS	Invalid QoS class specified.
3159	cve LPortInvalidSvcQoS1	The CBR QoS is currently not supported.
3161	cveSelectCluster	Please select a cluster.
3162	cveCreateCluster	Please create cluster first before adding switch to a cluster subnet.
3163	cveVerificationComplete	Verification has completed.
3164	cvePcrScrTooLow	The calculated minimum gap or the maximum number of credits is not within the range of 0 to 255. The Sustainable Cell Rate or Peak Cell Rate specified may be too low.
3165	cveInvalidDefinedXface	The defined interface is invalid for this card type.
3166	cveInvalidClockSpeed	The clock speed is invalid for this physical port type.
3167	cveIfNoAllocFailure	Can not allocate IfIndex for the LPort, contention with another process. Wait for 30 seconds and try again.
3168	cveInvE164Addr	E.164 terminating address must not be all zeroes.
3169	cve_tmCDV_outOfRange	Traffic Management: CDV is out of range.
3170	cve_tmRCDV_outOfRange	Traffic Management: Reverse CDV is out of range.
3171	cve_tmCTD_outOfRange	Traffic Management: End to End Delay Threshold is out of range.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
3172	cve_tmCLR_outOfRange	Traffic Management: CLR is out of range.
3173	cve_tmRCLR_outOfRange	Traffic Management: Reverse CLR is out of range.
3174	cveCfm_ChgSVCAddrRange	Do you wish to reset the SVC VPI/VCI Range attributes based on these changes?
3175	cveInvalidChargePartyId	Invalid chargeable party ID
3176	cve LPortSvpcVpiRangeError	The specified SVPC VPI Min/Max Ranges are not within the VPC VPI Min/Max Ranges.
3177	cveRedundantNotAvailable	The redundant unit's status is down.
3178	cveSnmpDeleteError	The move circuit (Delete) failed on the selected object.
3179	cveSnmpAddError	The move circuit (Add) failed on the selected object.
3180	cveMvCktSnmpNoSuchName	The variable does not exist or is read-only.
3181	cveMvCktSnmpBadValue	The value given has incorrect type or length.
3182	cveMvCktNetDistAllNodeFailure	Network-wide distribution failure encountered at every node in the distribution list.
3183	cveNotAllSecondIpConfigured	You cannot swap the IP address until all the switches \nin the same network have their secondary IP addresses configured.
3184	cveCardTypeDoesNotSupportAps WithTrunkBackup	APS with Trunk Backup is not supported on this card type.
3185	cveInvalidApsWithTrunkBackup DisabledPort	APS with Trunk Backup can only be Disabled from the Primary Port.
3186	cveCardHasApsWithTrunkBackup EnabledOnPort	The card type cannot be changed because one or more physical ports\nare part of an Inter-Card APS with Trunk Backup pair.
3187	cveProtectionPortForApsTrkBackup OnSameCard	The protection port must be on a different card of the same type for Inter-Card APS with Trunk Backup.
3188	cveCardVpShapingCBR0Less1	Card CBR CLP0+1 value should be greater than EPD/CLP1 Discard value.
3189	cveCardVpShapingVBR0Less1	Card VBR Real Time CLP0+1 value should be greater than EPD/CLP1 Discard value.
3190	cveCardVpShapingVBRNRL0Less1	Card VBR Non Real Time CLP0+1 value should be greater than EPD/CLP1 Discard value.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
3191	cveCardVpShapingABR0Less1	Card ABR/UBR CLP0+1 value should be greater than EPD/CLP1 Discard value.
3192	cveCardVpThresholdsiTooBig	Sum of VP shaping CLP0+1 thresholds multiplied by the number of optimum trunks and VUNI LPorts must be less or equal to the number of buffers of its feeder LPort.
3193	cveInvalidIwu LPortVCShapingCfg	If LPort configured as a VC Shaped LPort then at least one VC shaper must exist\n in the shapers range 1-5, at least one in range 1-10 and at least one in the range 11-15.
3194	cveInvalidIwu LPortVPShapingCfg	If LPort configured as a VP Shaped LPort then at least one VP shaper must exist.
3195	cve LPortUsingVPShaping	Cannot modify shaper from VP to VC since it is used by LPort.
3196	cve LPort550VpiVciBitsNoModify	The number of valid VPI/VCI bits may not be modified while any Circuit is provisioned on the respective LPort.
3197	cve LPortTooManyVciBitsTrk8kBio	The number of valid VCI bits cannot be greater than 13.
3198	cve LPortTooManyVciBitsTrk16kBio	The number of valid VCI bits cannot be greater than 14.
3199	cveSwDownloadNoFile	There is no valid switch software file that can download.
3200	cveCfm_CommitSP8SP30SP40Card TypeChange	The change in Card Type between Switch Processor 20/30/40 may affect user data. Use "OK" to proceed. Use "Cancel" otherwise.
3201	cveCktFail_AdminDown	Circuit admin status is down.
3202	cveCktFail_NoVcbuff	Internal error: No VC buffer
3203	cveCktFail_NoBandwidth	Not enough bandwidth.
3204	cveCktFail_NoRoute	Destination node is unreachable.
3205	cveCktFail_Timeout	PVC segment call timed-out.
3206	cveCktFail_NoPduBuff	Internal Error: No PVC PDU buffer.
3207	cveCktFail_NoDest	Destination DLCI is not configured.
3208	cveCktFail_TrunkRnr	Optimum path trunk data flow is blocked.
3209	cveCktFail_TrunkDown	Trunk is down.
33210	cveCktFail_BalanceReroute	Load balance reroute.
3211	cveCktFail_Dead	No response from the remote switch beyond.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
3212	cveCktFail_DefPathReroute	Attempting to reroute through the defined path.
3213	cveCktFail_N1Down	UNI/NNI is down.
3214	cveCktFail_OtherPvcSecDown	PVC segments are inactive beyond.
3215	cveCktFail_OtherPvcSegRnr	PVC segments are not ready to receive data beyond.
3216	cveCktFail_UsingAltPath	Defined path is NOT available. The alternate path is being used.
3217	cveCktFail_IopDown	IOP/IOM is down.
3218	cveCktFail_NoPduMsg	No PVC Manager PDU msg buffer.
3219	cveCktFail_NoPort	Port is not configured.
3220	cveCktFail_Misconfig	Mis-configuration.
3221	cveCktFail_SVCSetupFail	SVC setup failed.
3222	cveCktFail_SrcBackedup	Source is in a 'backup' condition.
3223	cveCktFail_SrcUnknown	Source is unknown.
3224	cveCktFail_DestUnknown	Destination is unknown.
3225	cveCktFail_BuDLCICollision	DLCI collision on backup.
3226	cveCktFail_OldVerInPath	Node running incompatible version of switch software exists in circuit path.
3227	cveCktFail_SmdsMgmtTrunk	SMDS management trunk.
3228	cveCktFail_NeverCalled	Endpoint never called.
3229	cveCktFail_BothEndPtBackup	Both endpoints in 'backup'.
3230	cveCktFail_PvcRouteMgtTrunk	Attempting to route through management trunk.
3231	cveCktFail_NoMultiPntParent	Multipoint parent not found.
3232	cveCktFail_PvcRouteFail	Route changed during setup.
3233	cveCktFail_NoVpiVci	No VPI or VCI available.
3234	cveCktFail_SvcUserClear	SVC cleared by user.
3235	cveCktFail_PathRegister	Circuit path registration failed.
3236	cveCktFail_AtmChannel	Selected channel cannot be allocated.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
3237	cveCktFail_NoReverseBw	No available bandwidth in reverse direction.
3238	cveCktFail_InternalReset	PVC reset internally - reuse path.
3239	cveCktFail_PriRouteDisrupt	Disrupted due to priority routing.
3240	cveCktFail_NoNegPriBw	Could not allocate negative priority bandwidth.
3241	cveCktFail_NoDestVpiVci	Destination VPI/VCI is not configured.
3242	cveCktFail_BkpVpiVciColl	Backup call fails because VPI/VCI in use.
3243	cveCktFail_SmdsFrameOnly	If SMDS VP connection try to go over Frame-Only card.
3244	cveCktFail_CacNotInit	CAC not initialized.
3245	cveCktFail_CacSigError	Signalling error: SCR maybe greater than the PCR or PCR equal to zero.
3246	cveCktFail_CacLargeMbs	MBS is larger than the maximum configured MBS for the customized CAC.
3247	cveCktFail_CacCdvNotAvail	Configured CDV objectives can not be satisified for the CBR QoS Class.
3248	cveCktFail_QOSUnavail	Requested QOS is unavailable.
3249	cveCktFail_DelayUnavail	Requested Delay is unavailable
3250	cveCktFail_CLRUnavail	Requested CLR is unavailable.
3251	cveCktFail_OAMRecevied	ATM circuit down because OAM cells were received.
3252	cveCktFail_UnknownNetwork	Call to an unknown transit network.
3253	cveCktFail_InvalidNetwork	Call to an invalid transit network.
3254	cveCktFail_AlreadyAct_Inact	Caller vc already active/inactive when confirm received.
3255	cveCktFail_FrQosNoGuaranteed	FR-QoS not guaranteed from end to end for this circuit.
3256	cveCktFail_InsufBW_CLR	Insufficient bandwidth to satisfy CLR.
3257	cveCktFail_InsufBW_CDV	Insufficient bandwidth to satisfy CDV.
3258	cveCktFail_ReqAdmCstUnavail	The requested admin cost is not available.
3259	cveCktFail_ReqCdvUnavail	End-to-end CDV is not available.
3260	cveCktFail_WarmStartInit	Warm start is initiated on circuit.
3261	cveCktFail_NoWarmResources	No warm resources for circuit.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
3262	cveCktFail_WarmCircuits	Warm circuits.
3301	cveVpciNumberInvalid	The VPCI should be within the range of 0 - 65534.
3302	cveVpiNumberInvalid	The VPI should be within the range of 0 - 4095.
3303	cveVPCINumberDup	The VPCI number is duplicated either on local or target logical port.
3304	cveVpciExists	The logical port cannot be deleted because there are VPCI rows related to it.
3305	cvePnniNodeLevelInvalid	The valid range for PNNI Node Level is between 0 and 104.
3306	cvePnniNodePeerGroupInvalid	The maximum number of hex digits you can enter for PNNI Peer Group Identifier cannot exceed PNNI Node Level divided by 4.
3307	cvePnniNodeObjectNotFound	The PNNI node instance could not be found in the database.
3308	cvePnniIfAwInvalid	The valid range for PNNI Administrative Weight is between 1 and 16777215.
3309	cveInvalidMgmtSpvcName	Unable to obtain configuration information for the Management SPVC.
3310	cveDelMgmtSpvcWithNmsPath	Cannot delete the SPVC entry while it is used in a Management Path.
3311	cveLimitMspvcCard	MSPVC limit per card of 64 has been reached. Addition may not be made.
3312	cveNoAgent LPortSetforClient LPort	The agent LPort is not defined for the client LPort.
3313	cveNo LPortDefinedForVPCI	The LPort has to be created before setting VPCI table.
3314	cvePrimaryTrunkObjNotFound	The primary trunk does not exist.
3315	cveVCTrunkAdminUp	In order to change shaper from VC to VP, it is necessary to admin down the VC trunks before this operation can proceed. This is necessary because on VC Shaped trunks circuits are dynamically assigned to shapers; so, if you change a shaper that is currently in use as a VC shaper on a VC trunk to be a VP shaper, the circuits currently using that shaper will no longer have their data shaped as a VC.
3316	cveSecExtClkInvalidForSp30Sp40	The External Clock Out cannot be set to 'Secondary' for Sp30/Sp40. Please modify the System Timing Attributes.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
3317	cveSecExtClkInvalidForNp	The External Clock Out cannot be set to 'Secondary' for Np. Please modify the System Timing Attributes.
3318	cveSecPrefSysTimingClkInvalidFor Sp30Sp40	The Preferred System Timing Clock cannot be set to 'Secondary' for Sp30/Sp40. Please modify the System Timing Attributes.
3319	cveSecPrefSysTimingClkInvalidForNp	The Preferred System Timing Clock cannot be set to 'Secondary' for Np. Please modify the System Timing Attributes.
3320	cveModVPThreshCBR0NeedPramSync	The CBR CLP0+1 VP Shaping Threshold cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization.
3321	cveModVPThreshCBR1NeedPram Sync	The CBR EPD/CLP1 VP Shaping Threshold cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization.
3322	cveModVPThreshVBR0NeedPram Sync	The VBR CLP0+1 VP Shaping Threshold cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization.
3323	cveModVPThreshVBR1NeedPram Sync	The VBR EPD/CLP1 VP Shaping Threshold cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization.
3324	cveModVPThreshVBRNR0NeedPram Sync	The VBR Non-Real Time CLP0+1 VP Shaping Threshold cannot be changed dynamically in the switch. It can only be maintained\nthrough PRAM synchronization.
3325	cveModVPThreshVBRNR1NeedPram Sync	The VBR Non-Real Time EPD/CLP1 VP Shaping Threshold cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization.
3326	cveModVPThreshABR0NeedPram Sync	The ABR/UBR CLP0+1 VP Shaping Threshold cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization.
3327	cveModVPThreshABR1NeedPram Sync	The ABR/UBR EPD/CLP1 VP Shaping Threshold cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization.
3328	cve LPortTooManyVpiBitsNni550	The number of valid VPI bits cannot be greater than 12.
3329	cveSnmpTimedOutEndpoint1	The SNMP request timed out.
3330	cveSnmpNoSuchNameEndpoint1	The variable does not exist or is read only.
3331	cveSnmpBadValueEndpoint1	The value given has incorrect type or length.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
3332	cveSnmpTimedOutEndpoint2	The SNMP request timed out.
3333	cveSnmpNoSuchNameEndpoint2	The variable does not exist or is read only.
3334	cveSnmpBadValueEndpoint2	The value given has incorrect type or length.
3335	cveCBXShapingRateTooSmall	IOM card LPort vp shaping rate too small (should be in the range of 100 to bandwidth).
3336	cveInvalidRevertive	Revertive mode is only valid for Intra-Card APS 1+1.
3337	cveVpciNumberChanged	The VPCI should not be changed.
3338	cveCktInvalidDlciIdCllm1	End point one: DLCI ID must not be 1007 if CLLM of LPort is enabled.
3339	cveCktInvalidDlciIdCllm2	End point two: DLCI ID must not be 1007 if CLLM of LPort is enabled.
3340	cveIlmiSigInvalid	ILMI or Signalling can not be enabled because an Optimum cell trunk \nwith VPI Start or VPI Stop of zero has already been configured.
3341	cveVpiStartStopConflictWithPnni	Cannot have an Optimum cell trunk with VPI Start or VPI Stop of zero when an ATM PNNI port already exists.
3342	cveVpiStartStopConflictWithFeeder	Cannot have an Optimum cell trunk with VPI Start or VPI Stop of zero when the feeder has ILMI or Signalling enabled.
3343	cvePnniConflictWithVpiStartStop	An Atm PNNI port cannot be created because an Optimum cell trunk with VPI Start or VPI Stop of zero has already been configured..
3344	cveIlmiConflictWithVpiStartStop	The ILMI VPI is in use by an Optimum cell trunk's VPI Start Stop range or by a virtual UNI port's VPI Start Stop range.
3345	cveMPVCNotSupportedOnEndpt1	The switch software on endpoint 1 does not support MPVCs.
3346	cveMPVCNotSupportedOnEndpt2	The switch software on endpoint 2 does not support MPVCs.
3347	cve LPortInvalidAtmProtocol	The version of this switch software does not support Atm Protocol as PNNI 1.0.
3348	cveCardVpShapingLessZero	Card VP shaping threshold value must be greater than zero.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
3349	cveCardVpThresholdsiTooSmall	Can not create LPort since sum of VP shaping CLP0+1 thresholds multiplied by the number of optimum trunks and VUNI LPorts must be less or equal to the number of buffers of its feeder LPort.
3350	cveCanNotEnableIomVPShaing	Can not enable vp shaping since sum of VP shaping CLP0+1 thresholds multiplied by the number of optimum trunks and VUNI LPorts must be less or equal to the number of buffers of its feeder LPort.
3351	cveTfdMcrTooBig	MCR must be less than or equal to PCR0+1.
3352	cveCardSysClockTypeNotSupported	The selected system clock source is invalid on this card.
3353	cveInvalidEthernetPortName	Unable to obtain configuration information for Ethernet Port.
3354	cveDelMgmtEthnet LPortWithNmsPath	Cannot delete the LPort entry while it is used in a Management Path.
3355	cveSwitchNameTooLong	Switch name cannot exceed 31 characters.
3356	cveInvalidSwImageInList	List contains invalid switch software image file. Operation aborted.
3357	cveCktCirExceed LPortOverSub	Can not create/modify circuit since sum of CIR must not exceed the oversubscribed bandwidth of its logical port.
3358	cveSvcNodePrefixAdminCostInvalid	The valid range for Node Prefix Admin Cost is between 0 and 65535.
3359	cveTrkStaticDelayInvalid	The valid range for Static Delay is between 0 and 16777215.
3360	cveCktInvalidPrioRev	Invalid reverse circuit priority value.
3361	cveCktTrafficDescTypeInvalid	Traffic Descriptor Type is invalid.
3362	cveMapNotManaged	This map will not be managed by NavisCore.
3363	cveSvcPrefixAdminCostInvalid	The valid range for SVC Port Prefix Admin Cost is between 0 and 65535.
3364	cveVci0and4Used	VCI = 0 is reserved and VCI = 4 is also reserved for F4 OAM circuit.
3365	cveVci5Used	Cannot set VCI to 5 since enabling signaling is functionally equivalent to using VCI= 5.
3366	cveVci18Used	Cannot set VCI to 18 since ATM protocol set to PNNI is functionally equivalent to using VCI= 18.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
3367	cveFastLoadNotSupported	The switch software does not support Rapid Upgrade.
3368	cvNoFastLoadNotScheduled	No Rapid Upgrade was Scheduled.
3369	cveCanNotStopFastLoad	It is too late to stop scheduled rapid upgrade.
3370	cveModVPShapingStateNeedPramSync	The Vp Shaping State cannot be changed dynamically in the switch. It can only be maintained through PRAM synchronization. Note that if the Vp Shaping State is being set to disabled, any LPorts on this card that have VP Shaping enabled should also be set to disabled.
3371	cveCktCirExceed LPortOverSub2	Cannot enable oversubscription of bandwidth at this percentage due to the fact that there are circuits defined on this LPort that are already over this amount.
3372	cve LPortInvalidOamAlmTimerThresh Orig	The OAM alarm timer threshold value is invalid. Valid range is 1 - 9.
3373	cveCfm_DS0LoopbackWarn	Performing an Internal Loopback for this logical port on this card type may disrupt service on the whole physical port! Do you really want to continue?
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 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 to the server timed out.
4110	cveEndpoint2Needed	The second endpoint must be specified in argument 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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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.
4126	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 configured as a redundant 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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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	Error in invoking SNMP API.
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 LPort.
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 configured as a redundant SP or be empty -- for 500 switch.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
4163	cveInvalidShaperId	The Traffic Shaper ID must be in (1, 15).
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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
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	cveBackup LPortNotValid	Backup LPort 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 Backup LPort 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	cvePrimary LPortMissing	Server agent : Primary LPort needs to be specified for addition.
4196	cvePsrvSnmpAgentInvalidPdu	Server agent: invalid PDU.
4197	cvePsrvSnmpAgentTooManyRow StatusVbs	Server agent: Too many row status varbinds.
4198	cvePsrvSnmpAgentTooFewRow StatusVbs	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	cvePsrvSnmpAgentRowCreation Failure	Server agent: Object instance does not exist or row creation failure 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.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
4209	cveNtmAttrDoesNotApply	Ntm attributes do apply for this type of LPort.
4210	cveInvSwForSvcNodePrefix	Invalid switch, svcnodeprefix can be configured only on CBX, GX or B-STDX switches.
4211	cveInvLPForSvcAddress	Invalid LPort, svcaddress is valid only on ATM UNI/NNI Cell endpoints on CBX and GX switches as well as FR UNI endpoints on B-STDX switches.
4212	cveInvLPForSvcPrefix	Invalid LPort, svcprefix is valid only on ATM UNI/NNI Cell endpoints on CBX and GX switches as well as FR UNI endpoints on B-STDX switches.
4213	cveInvLPForSvcConfig	Invalid LPort, svcconfig is valid only on ATM UNI/NNI Cell endpoints on CBX and GX switches as well as FR UNI endpoints on B-STDX switches.
4214	cveInvLPForUP	Invalid LPort, svcuserpart can be can be configured only ona UNI DTE LPort on 500 switches.
4215	cveInvSvcAddrString	Could not 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 in 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	cvePsrvSnmpAgentOnePDUCreateOr Modify	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.
4225	cveCanNotAdd LPortToRedundantPPort	Cannot add LPorts to redundant PPorts.
4226	cveCustomerNameMissing	The customer name is missing.
4227	cveInvalidVpnName	Public is not a valid vpn name.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
4228	cveInvalidCustomerName	Public is not a valid customer name
4229	cveTrunk LPortInvalid	The logical port specified for the trunk endpoint could not be found.
4230	cveTrunk LPortNeeded	Trunk LPort not specified.
4231	cveTrunkTypeInvalid	Trunk Type cannot be modified.
4232	cveDuplicateRefTimeServer	A ReferenceTimeServer with the same IP address already exists.
4233	cveMaxThreeTimeServer	Maximum of three ReferenceTimeServers can be added for a switch.
4234	cveNoSuchBindingObject	No such MLFRBinding object exists.
4235	cveBadMLFRContainerType	The container object's type for MLFRBinding should be MLFRTrunkBundle LPort.
4236	cveTrunk LPortsOnSameSwitch	A trunk cannot be created when the LPorts are on the same switch.
4237	cveNumRetriesOutOfRange	Number of retries out of range; valid range (0.5).
4238	cveSnmpResourceUnavailable	Object instance is presently unavailable, try later.
4239	cveSharedCircuitEpt	One of the circuit enpoint is being used.
4240	cveNotaValidPeer	The peer LPort of a Vpci object should be either the parent LPort or the client LPort that have selected parent LPort as agent.
4241	cveCanNotSetVpciOn LPort	A Vpci object can be configured on an LPort only if that LPort's vpiToVpciMappingType is Table.
4242	cveProxyAgentInvalid	Only the LPorts with ProxyAdminStatus agent can be a ProxyAgent.
4243	cveWrongProxyStatusValue	This value for ProxySigAdminStatus is applicable only for Atm unidce LPorts.
4244	cveVpciObjectExists	The value of VpiToVpciMappingType or ProxySigAdminStatus can not be changed since VpciTable is not empty.
4245	cvePeerNotLocal	Can't change the Proxy Signaling Admin Status to disabled since VpciTable on this LPort has a peer PSC LPort.
4246	cvePeerNotExists	The specified Peer LPort does not exists.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
4247	cvePeer LPortNeeded	The value for Peer LPort is not given.
4248	cveArgDoesNotApplyToProtectionPort	This argument is not applicable to protection PPorts.
4249	cveArgApplyIfInterCardAps	This argument is applicable only if the PPort's redundancy is APS resilient Uni or Aps with trunk backup.
4250	cveArgApplyIfApsSet	This argument applies only if the PPorts have an Aps pair.
4251	cveUioBandwidthConflict	Bandwidth and ClockSpeed do not match; using either one of them is enough. (ClockSpeed will be obselete).
4252	cveInvalidValue	Invalid value for this attribute.
4253	cveInvalidAtmProtocol	The specified protocol is not applicable for this LPort type.
4254	cveSwitchTypeNotSupported	The specified Switch Does not support this feature.
4255	cveRequiresCdv	CDV value must be specified using the apropriate attribute.
4256	cveRequiresCtd	CTD value must be specified using the apropriate attribute.
4257	cveRequiresClr	CLR value must be specified using the apropriate attribute.
4258	cvePsVersionMismatch	Client/Server Version Mismatch.
4259	cvePrimaryBackupMismatch	The Primary and the Backup Trunk's endpoints must be between the same switches.
4260	cveShaperTypeShouldBeVp	This attribute applies to this type of LPort only if ShaperType is Vp.
4261	cveInvalidUnsignedInt	The value is out of range of 1 - 4294967295 (a full 32-bit integer).
4262	cveClientAsscociated	The Proxy Signalling Admin Status cannot be changed from Agent when one or more clients have selected this LPort as their agent.
4263	cveAllPeerNotLocal	The Proxy Signalling Admin Status can be changed from Agent to Disabled only when all the peers of the LPort are local.
4264	cveCannotChangeTheAgent	The Proxy Signalling agent of the LPort cannot be changed since VpciRows is associated with it.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
4265	cveOnlyVpShapersCanbeAssigned	Only Vp TrafficShapers can be assigned for OPTimum Cell Trunk and Direct Cell Trunk LPort types.
4266	cveTrafficShaperShouldBeVp	The specified trafficShaper should be a Vp trafficshaper.
4267	cveIncorrectPrimarySpecified	Only a Primary Trunk can be specified as a TrunkForBackup.
4268	cveInvLPForSvcNetworkId	SvcNetworkId can be created only on FR UNI LPorts on a 9000 switch, and ATM UNI/NNI LPorts on a 500 switch. It is not supported on the 6PortDS3T3 card on the 500 switch.
4269	cveSDErrorSVDisable	Source Default can be enabled only if Source Validation is enabled.
4270	cveANErrorRDDisable	Adjacent Network applies only if SourceValidation is disabled.
4271	cveRDAlwaysDisableForAtmNniLP	Route Determination is always disabled on an ATM NNI LPort, so this is a read-only attribute for an ATM NNI LPort.
4272	cveCfm_CommitMptChange	The change in Mpt State will cause the node to be warm booted. Are you sure?
4273	cveCktAliasInUse	The specified circuit alias name is not unique.
4274	cveSnmpNoAccess	Return noAccess if OID prefix is incorrect.
4275	cveSnmpNoCreation	Return noCreation if the length of the index of the given OID and that specified by the table do not match.
4276	cveInvalidHoldoffTime	Holdoff Time value must be between 0 and 255 seconds.
4277	cveTrafficShaperNotChangedOnPort1	The traffic shapers should not be modified on the pport except the first pport on this card.
4278	cveTrafficShaperFirstShaperChanged	The parameters of the first traffic shaper should not be modified on this card.
4279	cveInvalidBulk LPortTemplate	Invalid LPort template for bulk LPort creation.
4280	cveInvalidBulk LPortBaseInput	Invalid Bulk LPort Input.
4281	cveInvalidBulk LPortDS0Input	Invalid Bulk LPort DS0 Input
4282	cveBulk LPortNameInUse	The sepcified LPort name is already used by other bulk LPort.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
4283	cveBulk LPortIdInUse	The specified LPort ID is already used by another bulk LPort.
4284	cveBulk LPortDS0sInUse	The sepcified LPort DS0 is already used by another bulk LPort.
4285	cveVpiBitsInvForExistTrunk	There exists a Cell Trunk whose Vpi Start/Vpi Stop bits are out of range due to the change of the number of valid Vpi bits. Either change this feeder ports number of valid Vpi bits or the Cell Trunks Start/Stop bits.
4286	cveBulk LPortNoBillObj	Cannot add an LPort because the template does not have a billing object.
4287	cveWrongVpiToVpciMappingValue	This value for VpiToVpciMappingType is not applicable for AdminStatus value of Client.
4332	cveNoSuchTD	The specified Traffic Descriptor does not exist.
4333	cveInvalidCktPriority	The range for the priority value is 1 to 3.
4334	cveScrMbsPcr1AbrInvalid	Scr1, Mbs1, or Pcr1 attributes do not apply when specifying ABR as a Qos class. Use FwdParam1, and/or FwdParam2 attributes.
4335	cveScrMbsPcr2AbrInvalid	Scr2, Mbs2, or Pcr2 attributes do not apply when specifying ABR as a Qos class. Use RevParam1, and/or RevParam2 attributes.
4337	cveScrMbsPcr1TdInvalid	Scr1, Mbs1, or Pcr1 attributes do not apply when specifying the FwdTrafficDescType attribute. Use FwdParam1, FwdParam2 and/or FwdParam3 attributes.
4338	cveScrMbsPcr2TdInvalid	Scr2, Mbs2, or Pcr2 attributes do not apply when specifying the RevTrafficDescType attribute. Use RevParam1, RevParam2 and/or RevParam3 attributes.
4339	cveScrMbsPcr1Fp	Pcr1, Scr1 or Mbs1 attributes do not apply when using FwdParam1, FwdParam2 and/or FwdParam3 attributes.
4340	cveScrMbsPcr2Fp	Pcr1, Scr1 or Mbs1 attributes do not apply when using FwdParam1, FwdParam2 and/or FwdParam3 attributes.
4341	cveCDVInvalidQos	Extended Qos CDV is only supported when the Qos is CBR or VBR-RT.
4342	cveCLRInvalidQos	Extended Qos CLR is only supported when the Qos is CBR, VBR-RT or VBR-NRT.
4400	cvePSGetOperInfo	Provisioning Server Get OperInfo.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
5501	cve LPortDelIpInterfaceDefined	Unable to delete because IP parameters exist on this LPort.
	cveUnnumberedIfIpAddressAlreadyDefined	Can not define more than 1 IP address for unnumbered IP interface.
5502	cveCfm_IpIfIpAddress	Changes to IP Parameters must be committed before setting IP interface attributes. Click OK to commit changes.
5503	cveIpAddrExistForChangeToUnnumberedIf	Can not change to unnumbered interface since none or more IP addresses are already defined.
5504	cveOnlyOneIPAddrForUnnumberedIf	Only one IP address is allowed for an unnumbered interface.
5505	cveIpAddressInvalidUcastAddr	Invalid unicast IP address entered.
5506	cveIpAddressInvalidBcastAddr	Invalid broadcast IP address entered.
5507	cveIpAddressInvalidIpMask	Invalid IP mask entered.
5508	cveIpAddressEmptyUcastAddr	Must enter a unicast address.
5509	cveIpAddressEmptyNetworkMask	Must enter a network mask.
5510	cveIpAddressCktDefinedForPPP LPort	IP switching is not allowed on this PPP logical port because a circuit is already defined using this PPP logical port.
5511	cvePPPIpAddressAlreadyDefined	Circuit is not allowed on this PPP logical port because IP switching is enabled on this PPP logical port.
5512	cveIpIfDelFilterAssocDefined	Cannot delete IP LPort. IP filter association still exists.
5513	cveIpIfDelIpIfAddrDefined	Cannot delete IP LPort. IP address still exists.
5514	cveIpIfAddrDelFilterAssocDefined	Cannot delete IP address. IP filter association still exists.
5515	cveIpIfDelIpDlidDefined	Cannot delete IP LPort with protocol connection defined.
5516	cveIpAddressInUseInSwitch	IP address already defined on this switch.
5517	cve LPortDelIpStaticRouteInUse	Cannot delete IP LPort. A static route is referencing the unnumbered interface.
5518	cveIpStaticRouteReferenceUnnumberedIf	Cannot disable this unnumbered interface. A static route is referencing it.
5519	cveIpIfDelQosProfileAssocDefined	Cannot delete IP LPort. QoS profile association still exists.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
5520	cveCfm_IpIfDelete	Deleting IP LPort. Click OK to continue.
5521	cveIpStaticRouteExists	Duplicate static route entry defined.
5522	cveIpAddressInvalidUcastMTU	Invalid Unicast MTU entered. Valid range for the value is: 128-8192 for Frame and PPP LPorts, 128-9180 for ATM LPorts, and 46-1500 for Ethernet LPorts.
5523	cveIpAddressInvalidBcastMTU	Invalid Broadcast MTU entered. Valid range for the value is: 128-8192 for Frame and PPP LPorts, 128-9180 for ATM LPorts, and 46-1500 for Ethernet LPorts.
5524	cveIpAddressCannotChangeUCastAddr	Cannot modify unicast address.
5525	cveIpAddressCannotChangeNetwork Mask	Cannot modify network mask.
5526	cveIpAddressInvalidNetworkAddress	Invalid network address entered.
5527	cveIpStaticRouteMaxLimitReached	Cannot add new static Route. Maximum number of Static routes allowed per switch is 2000.
5528	cveIpAddressMaxLimitPerCard Reached	Cannot add a new IP interface. Maximum numbe of IP Interfaces allowed per card is 200.
5529	cveIp LPortMaxAllowedPerCard Reached	Cannot add a new IP LPort. Maximum number of IP LPorts allowed per card is 120.
5530	cveQosFilterNameMissing	Missing QoS Profile Name.
5531	cveQosFilterSrcAndDestAddrMissing	Missing both source and destination address.
5532	cveQosFilterSrcMaskMissing	Missing source mask.
5533	cveQosFilterDestMaskMissing	Missing destination mask.
5534	cvQosFilterInvalidSrcMask	Invalid source network mask.
5535	cvQosFilterInvalidDestMask	Invalid destination network mask.
5536	cvQosFilterInvalidSrcAddr	Invalid source address.
5537	cvQosFilterInvalidDestAddr	Invalid destination address.
5538	cvQosPVCReferenceByQosProfile	Can not delete circuit. It is referenced by a QoS Flow Profile.
5539	cvQosFilterNameInUse	QoS Profile Name already in use.
5540	cvQosMaxNumProfilePerSwitch Reached	Maximum number of QoS Profiles allowed per switch is 100.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
5541	cvQosMaxNumAssoc LPortReached	Maximum number of QoS profiles allowed to assign to an IP LPort is only 50.
5542	cveQosFilterSrcAddrMissing	Missing source address.
5543	cveQosFilterDestAddrMissing	Missing destination address.
5544	cveQosFilterMax LPortsPerSwitch	Associating QoS Flow profiles to more than 100 IP LPorts per switch is not allowed.
5545	cvFilterSrcAddrMissing	Missing source address.
5546	cvFilterDestAddrMissing	Missing destination address.
5547	cvFilterSrcMaskMissing	Missing source network mask.
5548	cvFilterDestMaskMissing	Missing destination network mask.
5549	cvFilterSrcLowGTSrcHighAddr	Invalid source address range. Low source address is higher than high source address.
5550	cvFilterDestLowGTDestHighAddr	Invalid destination address range. Low destination address is higher than the high destination address.
5551	cvFilterInvalidSrcMask	Invalid source network mask.
5552	cvFilterInvalidDestMask	Invalid destination network mask.
5553	cvFilterFilterNameMissing	Missing filter name.
5554	cvFilterLowProtoIDOutOfRange	Out of range specified for low protocol ID (0 - 254).
5555	cvFilterHighProtoIDOutOfRange	Out of range specified for high protocol ID (0 - 254).
5556	cvFilterTOSOutOfRange	Out of range specified for type of service (0 - 254).
5557	cvFilterLowSrcPortOutOfRange	Out of range specified for low source port (0 - 65535).
5558	cvFilterHighSrcPortOutOfRange	Out of range specified for high source port (0 - 65535).
5559	cvFilterLowDestPortOutOfRange	Out of range specified for low destination port (0 - 65535).
5560	cvFilterHighDestPortOutOfRange	Out of range specified for high destination port (0 - 65535).
5561	cvFilterOffsetOutOfRange	Out of range specified for offset.
5562	cvFilterLengthOutOfRange	Out of range specified for length.
5563	cvFilterValueOutOfRange	Out of range specified for value.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
5564	cvFilterInvalidLowSrcAddr	Invalid low source address.
5565	cvFilterInvalidHighSrcAddr	Invalid high source address.
5566	cvFilterInvalidLowDestAddr	Invalid low destination address
5567	cvFilterInvalidHighDestAddr	Invalid high destination address.
5568	cvDelFilterWith LPortAssoc	Can not delete filter. An association of\nthis filter to an IP LPort still exists.
5569	cvDelFilterWithIpAddrAssoc	Can not delete filter. An association of\nthis filter to an IP address still exists.
5570	cvFilterNameInUse	Filter name already in use.
5571	cvFilterMaxNumAssoc LPortReached	Associating more than 32 inbound or outbound Filters per IP LPort is not allowed.
5572	cvFilterMaxNumAssocProtocol Reached	Associating more than 32 protocol Filters per switch is not allowed.
5573	cvDelFilterWithProtocolAssoc	Can not delete filter. A host filter association still exists.
5574	cveCfm_FilterExitWithConfigChanges	Configuration changed. Exit anyway ?
5575	cveIpFilterMaxLimitReached	Can not add new Filter. The maximum number of filters allowed per switch is 1024.
5576	cveIpFilterSrcNetMaskInsufficient	The source network mask does not sufficiently mask the source low or high address. Example: Address (152.148.10.0), Mask (255.255.255.0).
5577	cveIpFilterDestNetMaskInsufficient	The destination network mask does not sufficiently mask the destination low or high address. Example: Address (152.148.10.0), Mask (255.255.255.0).
5578	cvFilterMaxNumAssocCardReached	Associating IP Packet filters to more than 128 IP LPorts and IP Circuits per card is not allowed
5579	cvFilterMaxNumAssocCktReached	Associating more than 32 inbound or outbound Filters to an IP Circuit/Protocol Connection ID is not allowed.
5580	cveIpDLIDNumberInUse	Defined protocol connection ID already in use.
5581	cveIpDLIDMaxCardEntriesReached	Maximum protocol connection ID per card reached.
5582	cveIpDLIDInvalidDLCI	Invalid DLCI entered. The valid range for the DLCI is 16-991.
5583	cveIpDLIDInvalidVpiVciInput	Invalid VPI/VCI entered.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
5584	cveIpDLIDInvalidVpi	Invalid VPI Entered.
5585	cveIpDLIDInvalidVci	Invalid VCI Entered.
5586	cveDelIpDLIDWithFilterAssoc	Cannot delete protocol connection ID. One or more IP Packet filters are associated it.
5590	cveArpInvalidNumDigitForMac Address	Invalid number of digit for MAC address in ARP entry. Each address byte must be separated by a colon(:).
5591	cveArpInvalidSyntaxForMacAddress	Invalid MAC address syntax in ARP entry.
5592	cveArpIpAddressRequired	Missing IP address in ARP entry.
5593	cveArpInvalidIpAddressSyntax	Invalid IP address syntax in ARP entry.
5594	cveArpDlciOutOfRange	Value for DLCI is out of range.
5595	cveArpVpiOutOfRange	Value for VPI is out of range.
5596	cveArpVciOutOfRange	Value for VCI is out of range.
5597	cveArpDuplicateEntry	Duplicate ARP entry defined.
5600	cveSRouteIpAddressMissing	Missing IP address in static route entry.
5601	cveSRouteIpAddressInvalid	Invalid IP address in static route entry.
5602	cveSRouteNetworkMaskMissing	Missing network mask in static route entry.
5603	cveSRouteNetworkMaskInvalid	Invalid network mask in static route entry.
5604	cveSRouteNextHopMissing	Missing next hop address in static route entry.
5605	cveSRouteNextHopInvalid	Invalid next hop address in static route entry.
5606	cveSRoutePriorityOutOfRange	Out of range specified for priority value.
5607	cveSRouteTagOutOfRange	Out of range specified for tag value.
5608	cveSRouteNextHopShouldBeZero	The next hop address should be 0.0.0.0 if the unnumbered interface is selected or if the Null route is enabled.
5609	cveSRouteIpAddrAndNextHopCannot BeSame	The Destination Network IP Address and the Next Hop Address can not be the same.
5700	cveInvalidOspfNbrPrio	Priority value must be between 0 and 255.
5701	cveOspfNbrObjectNotFound	The OSPF Neighbor object could not be found in the database.
5702	cveInvalidOspfTransDelay	Transit Delay value must be between 0 and 3600.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
5703	cveInvalidOspfRoutePrio	Route Priority value must be between 0 and 255.
5704	cveInvalidOspfTransInt	Transit Interval value must be between 0 and 3600.
5705	cveInvalidOspfHelloInt	Hello Interval value must be between 1 and 65535.
5706	cveOspfObjectNotFound	The OSPF object could not be found in the database.
5707	cveInvalidOspfTos0Metric	TOS 0 Metric value must be between 0 and 65535.
5708	cveOspfAreaAggrNotFound	The OSPF Area Aggregate object could not be found in the database.
5709	cveOspfVLinkNotFound	The OSPF Virtual Link object could not be found in the database.
5710	cveOspfNbrAddrInUse	The specified neighbor address is already in use on this switch.
5711	cveCfm_DeleteOspf	Do you really want to delete this OSPF Interface?
5712	cveInvalidOspfDeadInt	Router Dead Interval must be a multiple of Hello Interval.
5713	cveInvalidNoneKey	Authentication Key must be empty if the Authentication Type is None.
5714	cveInvalidSimpleKey	Authentication Key length can not more than 8 octets if the Authentication Type is Simple Password.
5715	cveInvalidMd5Key	Authentication Key length can not more than 16 octets if the Authentication Type is MD5.
5716	cveCfm_OspfArea1	Configuring OSPF over logical ports in the area 1 will not work unless the other OSPF router(s) are also Ascend equipment. Do you want to continue ?
5717	cveOspfNbrMaxLimitReached	Error adding OSPF Neighbor. Maximum number of OSPF neighbors allowed to add per switch is 72.
5718	cveOspfVLinkDuplicate	An OSPF Virtual link with the same area ID and neighbor already exist.
5719	cveOspfAreaAggrDuplicate	An entry with same area ID, LSDB type, network address, and mask combination already exists.
5720	cveCfm_CommitArea1Comapt	The change in OSPF Area 1 Backward Compatible will cause the switch to be rebooted. Are you sure?
5721	cveCfm_ModifyOspfInterfaceType	Changes to the OSPF Interface type will take effect and the full adjacencies will be established only after the card is warm booted.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
5722	cveOspfIfMaxLimitReached	Can not add new OSPF Interface. Maximum number of OSPF interfaces allowed per switch are is 100.
5800	cveRip2ObjectNotFound	The RIP2 object could not be found in the database.
5801	cveInvalidRip2DefMetric	Default Metric value must be between 0 and 15.
5802	cveCfm_DeleteRip2	Do you really want to delete this RIP Interface?
5803	cveAuthTypeWithoutKeyIsInvalid	Authentication key must be specified if the authentication type is not none.
5804	cveRip2MaxLimitReached	Error adding new RIP Interface. Maximum number of RIP interfaces allowed per switch is 100.
5805	cveRip2PolAsccMaxLimitReached	Error adding/Modifying RIP Interface. This is because, with the newly assigned route maps, the number of RIP Interface and Route Map associations is exceeding the allowed limit of 50.
5806	cveInvalidRip2SendHost	Send Host Routes field should always be set to Disable for B-STDX Switches Rev 5.0.0 and higher.
5830	cveRtfGenDuplicateListIndex	An entry with this List Number exists in the database.
5831	cveRtfDuplicateFilter	Duplicate Filter defined.
5832	cveRtfFilterDeleteInvalid	Cannot delete this filter; the filter belongs to an access list.
5833	cveRtfInvalidIndexValue	Invalid index value.
5834	cveRtfInvalidNameValue	Input a valid name.
5835	cveRtfDuplicateName	Duplicate Name defined.
5836	cveRtfMinNetPrefixLen	Min Net Prefix Len is out of range (0 - 32).
5837	cveRtfDuplicateFilterIndex	Duplicate Filter Index.
5838	cveRtfPolicyRipMetric	Rip Metric should be between 0-16.
5839	cveRtfAccessListDeleteInvalid	Cannot delete this access list. The access list is in use.
5840	cveRtfNullTagAccessValue	Tag access value can not be empty.
5841	cveRtfNullNetworkAddress	Network address cannot be empty.
5842	cveRtfNullNetworkMask	Network mask can not be empty.
5843	cveRtfNullOriginAS	Origin AS cannot be empty.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
5844	cveRtfNullTransitAS	Transit AS can not be empty.
5845	cveRtfNullLastAS	Last AS can not be empty.
5846	cveRtfNullCommValue	Community value can not be empty.
5847	cveRtfPolicyDeleteInvalid	Cannot delete this route map. The route map is currently in use.
5848	cveRtfDuplicatePolicy	A route map with the same name exists in the database.
5849	cveRtfAsPathFilterMaxLimitReached	Cannot add a new AS path filter; the maximum number of filters allowed per switch is 1000.
5850	cveRtfCommFilterMaxLimitReached	Cannot add a new Community filter; the maximum number of filters allowed per switch is 1000.
5851	cveRtfTagFilterMaxLimitReached	Cannot adda new tag filter; the maximum number of filters allowed per switch is 1000.
5852	cveRtfNetFilterMaxLimitReached	Cannot add a new network filter; the maximum number of filters allowed per switch are 5000.
5853	cveRtfAccessListMaxLimitReached	Cannot add new access list; the maximum number of access lists allowed per switch is 100.
5854	cveRtfMapMaxLimitReached	Cannot add a new route map; the maximum number of route maps allowed per switch is 150.
5855	cveRtfNetFilterMaskInsufficient	Network Mask does not sufficiently mask the Network Address. Example: Address (152.148.10.0), Mask (255.255.255.0).
5856	cveRtfMaxNetFilterPerNetList	Network Filter association limit (300) per Network Access List reached.
5857	cveRtfMaxNetListPerRouteMap	Network Access List association limit (100) per Route Map reached.
5858	cveRtfMaxRouteMapPerBgpNbr	Route Map association limit (50) per BGP Neighbor reached.
5859	cveRtfMaxRouteMapPerRip2	Route Map association limit (50) per RIP Interface reached.
5860	cveRtfInvalidOriginAs	Invalid Origin AS (1 to 65535).
5861	cveRtfInvalidTransitAs	Invalid Transit AS (1 to 65535).
5862	cveRtfInvalidLastAs	Invalid Last AS (1 to 65535).

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
5863	cveRtfAssignToBgpNbr	This Route Map will not take effect unless it is assigned to a BGP Neighbor.
5864	cveRtfAssignToRip	This Route Map will not take effect unless it is assigned to a RIP Interface.
5865	cveRtfInvalidSetBgpPath	Invalid AS Repeat Count (0 to 65535).
5866	cveRtfMaxNetPrefixLen	Max Net Prefix Len is out of range (0 - 32).
5867	cveRtfMaxNetPrefixLenTooSmall	Max Net Prefix Len must be greater or equal to Min Net Prefix Len.
5868	cveRtfCreateBgpAggr	Once BGP aggregate is defined, aggregation will occur. But neither the aggregate not its contributors will be advertised until an Aggregate->BGP route map is assigned to one of more BGP Neighbors.
5869	cveDupOspfPolSeqNum	The specified Sequence Number is already in use on this switch. The sequence number has to be unique within those route maps whose action is Accept/Deny or within those route maps whose action is Originate Default.
5870	cveRtfNullSetMetric	Metric field in the Set Parameters cannot be empty. Valid range for the value is 1-16.
5871	cveRtfInvalidSetMetric	Invalid value for Metric field in the Set Parameters is specified. Valid range for the value is 1-16.
5872	cveRtfInvalidAction	The action cannot be set to Originate Default if the to-protocol is Routing Table.
5873	cveRtfActionCannotChange	The action cannot be changed from Accept/Deny to Originate Default, or from Originate Default to Accept/Deny when the route map is currently assigned to one or more BGP Neighbors or RIP Interfaces.
5874	cveOsfRtfActionCannotChange	The action cannot be changed from Accept/Deny to Originate Default, or from Originate Default to Accept/Deny for OSPF Route maps.
5900	cveBgpInvalidIpAddress	Network (IP) Address is Invalid.
5901	cveBgpAggregPrefOutOfRange	Preference field in BgpAggreg is out of range.
5902	cveBgpDuplicateIpAddress	An entry with this IP address exists in the database.
5903	cveBgpAggregMaskInvalid	Network (IP) Mask is invalid.
5904	cveBgpLocalPrefOutOfRange	Default Local Pref is out of range (1 - 65535).

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
5905	cveBgpInvalidClusterId	Cluster ID must be non-zero.
5906	cveBgpLocalAsOutOfRange	Local AS is out of range (0 - 65535).
5907	cveBgpAsWeightInvalid	AS Weight is out of range (1 - 65535).
5908	cveBgpAsDuplicateAsNumber	An entry with this AS number exists in the database.
5909	cveBgpPeerInvalidRemoteAddr	Invalid Remote Address entered.
5910	cveBgpPeerInvalidRemoteAs	Invalid Remote AS (1 to 65535).
5911	cveBgpPeerInvalidConnRetry	Connection Retry interval is out of range (0 - 65535).
5912	cveBgpPeerInvalidKeepAlive	Keep Alive interval is out of range (1 - 21845).
5913	cveBgpPeerInvalidHoldTime	Hold Time interval is out of range (3 - 65535).
5914	cveBgpPeerInvalidMinAsOrig	Min AS Origination interval is out of range (0 - 65535).
5915	cveBgpPeerInvalidMinRouteAdv	Min Route Advertisement interval is out of range (0 - 65535).
5916	cveNullBgpPeerName	BGP Neighbor Name cannot be empty.
5917	cveDupBgpPeerName	The specified BGP Neighbor Name is already in use on this switch.
5918	cveBgpPeerMaxLimitReached	Error adding BGP Neighbor. The maximum number of BGP neighbors allowed to add per switch is 50.
5919	cveBgpNetMaxLimitReached	Error adding BGP Network. The maximum number of BGP networks allowed to add per switch is 256.
5920	cveBgpAggrMaxLimitReached	Error adding BGP Aggregate. The maximum number of BGP Aggregates allowed to add per switch is 256.
5921	cveBgpAsMaxLimitReached	Error adding BGP AS. The maximum number of BGP ASes allowed to add per switch is 75.
5922	cveBgpClusterIdIsNotMust	You have changed the default value of Route Reflector Cluster ID parameter. You do not need to configure this parameter unless you will be configuring one or more BGP peers as route reflector clients and there will be more than one route reflectors existing in this cluster. If there is going to be only one route reflector in a cluster, by default, the router ID is used as the cluster ID. Do you really want to continue?
5923	cveBgpAsNumberInvalid	AS number is out of range (0 - 65535).

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
5924	cveBgpAggregTagInvalid	AS Tag value is out of range (0 - 255).
5925	cveBgpAggregCreateRouteMap	This aggregate will have no effect until a routemap is defined that advertises it.
5926	cveBgpLocalAsCannotBeZero	Local AS cannot be set to zero when one or more BGP neighbors existing.
5927	cveBgpCannotAddNeighbors	Cannot add BGP Neighbors when the local AS number is zero. Please go to the BGP Parameters screen, change the Local AS to a non-zero value, and add the neighbors.
5928	cveBgpCannotModifyNeighbors	Cannot modify BGP Neighbors when the local AS number is zero. Please go to the BGP Parameters screen, change the Local AS to a non-zero value, and modify the neighbors.
5929	cveBgpNoRouteRefClientForEBGP	Route Reflector Client cannot be set to Enable for EBGP Neighbor.
5930	cveBgpConfedIDInvalid	Confederation ID is out of range (0 - 65535).
5950	cveCfm_CommitGatewaySwAttr	Changes in Gateway Switch Attributes must be committed before setting RIP Configuration attributes.
5951	cveInvalidIpAddress	Invalid IP Address.
5952	cveInvalidIpMaskBit	Invalid IP Mask bit (1-32)
5953	cveSmdsIsNotSupported	SMDS is not supported by this switch version.
5954	cveNetFilterNoUsingAccessLists	There are no Network access lists that are currently using the selected Network Filter.
5955	cveNetAccessNoUsingRouteMaps	There are no Route Maps that are currently using the selected Network Access List.
5956	cveRouteMapNoUsingBgpPeers	There are no BGP Neighbors that are currently using the selected Route Map.
5957	cveRouteMapNoUsingRip2Ifs	There are no RIP Interfaces that are currently using the selected Route Map.
5958	cveMaskInsufficient	Network Mask does not sufficiently mask the Network Address. Example: Address (152.148.10.0), Mask (255.255.255.0).
5959	cveCPBasicOrPlusNotSupported	The CP Basic/Plus is not supported by this switch version.
5960	cveCktSwitchConflict	The endpoints cannot be on the same switch.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
5961	cveSRoutePriorityRange	Priority specification is out of the range (0 - 255).
5962	cveIp LPortUnnumberedIfDefinedOn NonPPP	Unnumbered interface is supported only on Point-To-Point LPorts. Please Disable unnumbered interface on this LPort.
5963	cvDelFilterWithIpCktAssoc	Cannot delete filter. A protocol connection ID (IP Circuit) association still exists.
5964	cveInvalidIpCktDlid	IP Packet filters cannot be associated to Protocol Connection IDs (IP Circuits) created on LPorts other than FrameRelay or ATM LPorts.
5965	cveIpLoopbackAddrDuplicate	Duplicate IP Loopback Address entry defined.
5966	cveIpLoopbackAddrNotFound	The IP Loopback Address object could not be found in the database.
5967	cvIpLoopbackAddrInvalidAddr	Invalid Address.
5968	cvIpLoopbackAddrInvalidAreaID	Invalid Area ID.
5969	cveIpFilterNoUsingIp LPorts	The selected IP Filter is not associated with any LPort.
5970	cveIpFilterNoUsingIpCkts	The selected IP Filter is not associated to any IP Circuit.
5971	cveIpLpbkMaxLimitReached	Cannot add new IP Loopback Address. Maximum number of IP Loopback Address allowed per switch is 16.
5972	cveIpIfAddressDuplicatePrimaryIf	Primary Interface is enabled for another IP Interface Address.
5973	cveSubnetInUseInSwitch	Another IP Interface on this switch is configured for the same subnet.
5974	cveHiProtLowProt	The High Protocol ID is less than the Low Protocol ID.
5975	cveHiSrvLowSrv	The High Service value is less than the Low Service value.
5976	cveIpArpMapToNoneDlci LPorts	The IP address is mapped to a none-Frame Relay Logical port.
5977	cve LPortModIpDlciExist	Cannot modify the number of valid VPI/VCI bits when one or more IP Protocol connection IDs exist on the LPort.
5978	cveSRouteNextHopMatchLocalIpAddr	The Next Hop Address cannot be the same as the local address.
6000	cveIgmpInvalidQueryIntvl	Query Interval Out of Range: 2..999.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
6001	cveIgmpInvalidQueryRespIntvl	Query Response Interval out of range: 1 - 100.
6002	cveIgmpInvalidRobustness	Robustness out of range: 2 - 10.
6003	cveIgmpInvalidVersion	Version out of range: 1 - 2.
6004	cveIgmpInvalidLastMemQIntvl	Last Member Query Interval out of range: 1 - 100.
6050	cveIpDvmrpAltNetDefined	Cannot delete IP LPort. Dvmrp Alternate Subnet object still exists.
6051	cveIpMcastBoundaryDefined	Cannot delete IP LPort. Dvmrp IP Mcast Boundary object still exists.
6052	cveDvmrpLocalAddressEmpty	Must enter a Local IP Address.
6053	cveDvmrpLocalAddressInvalid	Must enter a valid Local IP Address.
6054	cveDvmrpRemoteAddressEmpty	Must enter a Remote IP Address
6055	cveDvmrpRemoteAddressInvalid	Must enter a valid Remote IP Address.
6056	cveDvmrpAltNetNetworkAddress Empty	An Alternate Network Address is required.
6057	cveDvmrpAltNetNetworkAddress Invalid	A valid Alternate Network Address is required.
6058	cveDvmrpAltNetNetworkMaskEmpty	An Alternate Network Mask is required.
6059	cveDvmrpAltNetNetworkMaskInvalid	A valid Alternate Network Mask is required.
6060	cveIpMcastNetworkAddressEmpty	A Scoped Boundary Address is required.
6061	cveIpMcastNetNetworkAddressInvalid	A valid Scoped Boundary Address is required.
6062	cveIpMcastNetNetworkMaskEmpty	A Scoped Boundary Mask is required.
6063	cveIpMcastNetNetworkMaskInvalid	A valid Scoped Boundary Mask is required.
6064	cveDvmrpMetricInvalid	A valid Mertic is required. Range for Metric is 0 - 31.
6065	cveDvmrpThresholdInvalid	A valid Threshold is required. Range for Threshold is 0 - 255.
6066	cveDvmrpVIfInUse	Identical DVMRP object has already been configured.
6067	cveDvmrpVIfObjectNotFound	DVMRP VIF object not found.
6068	cveDvmrpAltNetInUse	Identical DVMRP Alternate object has already been configured.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
6069	cveIpMcastBoundaryObjectNotFound	IP Mcast Boundary object not found.
6070	cveIpMcastBoundaryInUse	Identical IP Mcast Boundary object has already been configured.
6071	cveCktIpPortConflict	The specified logical ports do not have valid types for creating a circuit for IP traffic.
6072	cveMvCktInvalid LPortType	Circuits cannot be moved to a logical port with different LPort type.
6073	cveIpAddressMaxLimitPerServer LPortReached	Can not configure more than 1 IP interface address per IP Server LPort.
7000	cveoOverrideError	All overridable error messages must be greater than cveo-OverrideError.
7001	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?
7002	cveoBgpAsModified	One or more BGP Neighbors have been defined using a different AS number. Changing the AS number may significantly alter routing behavior. Any routemaps redistributing routes from protocols other than BGP will no longer have effect if they have been applied to neighbors that are configured in the same AS as the new AS on this switch. Do you really want to change the AS number for this switch?
7003	cveoRtfNextHopNotRecommended	Setting/Changing the Next Hop parameter in the BGP->TABLE route map is not recommended. Do you want to continue?
7004	cveMul LPortUsingSameShaper	Other LPort(s) have been assigned to this shaper. The resulting cell stream will not be shaped for an individual VPI, but rather for the LPort's data stream, which may not be the desired function. Do you want to continue?.
7005	cveShaperRateExceedsBandwidth	Shaper rate exceeds bandwidth. Do you want to continue?.
7006	cveFRTrkNni LPortUsingsameShaper	Shaper has been assigned to the Frame trunk or Frame NNI LPort. Do you still want to change shaping type from VC to VP?

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
7007	cveUniDteDceCktUsingSameShaper	Shaper has been assigned to the circuit. Configuring this shaper will cause these circuits to shape the traffic as an aggregation of circuits, not as individual VC's. Do you still want to change the shaping type from VC to VP?
7008	cveAgentAssigned	The logical port has signalling clients attached to it.
7009	cveCktVpiVciConflictSignalling1	Invalid VPI/VCI due to conflict with ILMI VPI/VCI Settings of Logical Port Endpoint 1.
7010	cveCktVpiVciConflictSignalling2	Invalid VPI/VCI due to conflict with ILMI VPI/VCI Settings of Logical Port Endpoint 2.
7011	cveTrunkSubPportMismatch	Cannot allow trunk creation due to a sub-PPorts mismatch. Select LPorts with the same Pport IDs.
7012	cveCktInvalid550NNIvpi	VPI = 4095 is not allowed for configuring a PVC with an ATM NNI LPort on GX 550 switch.
7013	cveSpvcNameTooLong	The max name length is 20 characters.
7014	cveDefUniAddrAscTooLong	Length of DefaultUniAddrAsc should not exceed 15 digits.
8000	cveMptPtPtExists	There is already an MPT Point-to-Point connection defined between these two switches.
8001	cveMptPtPtConnectionNameMissing	Missing MPT Point-to-Point connection name.
8002	cveMptPtPtConnectionNameInUse	MPT Point-to-Point connection name already in use.
8003	cveMptPtPtConnectionSameSwitch	Cannot use same switch for both endpoints.
8004	cveMptPtPtDefinePathMaxHop Reached	Maximum hops reached for defined path.
8005	cveMptPtPtDefinePathTrunkInPath	Cannot delete trunk. It is needed in a MPT Point-to-Point define path.
8006	cveCfm_MptPtPtResetConnection	Reset MPT connection. Click OK to continue.
8007	cveMptPtPtDefPathNotSameArea	All trunks in the define path must have the same area ID.
8008	cveMptPtPtDefPathMixTrunk	Cannot mix Frame trunk and Cell Trunk in the define path.
8009	cveMptPtPtDefPathTrunkNotFound	One of the trunks in the define path does not exist in the database.
8010	cveModTrkMptPtPtDefinePath	Cannot modify Trunk Traffic Allowed field when the trunk is used in a MPT Point-to-Point define path.

Table 2-2. Error Codes (Continued)

Number	Message	Explanation
8011	cveMptPtPtOneOrAnotherDisable	One or both endpoints currently have MPTs disabled. Cannot allow any configuration.

Index

A

Adding
  a member 3-2 to 3-3, 3-35 to 3-36, 6-9 to 6-12
  an argument to an argument list 5-2 to 5-3
  an object 3-4 to 3-5, 3-37 to 3-38, 6-13 to 6-16
  objects to an ObjectList 5-115, 10-4
  two argument lists 5-4, 5-116, 9-3
  two ObjectLists 10-3
API, overview 1-1
APS, creating ID 5-29
Argument list
  adding an argument to 5-2 to 5-3
  adding two lists 5-4, 5-116, 9-3
  comparing lists for equality 5-6, 9-3
  comparing lists for inequality 9-3
  converting to printable string format 5-27
  creating a list 5-15
  creating attribute IDs 5-16, 9-2
  creating values for attributes 5-17, 9-2
  deleting pointer to 5-8
  determining error status 5-7, 9-5
  determining if specified argument exists 9-6
  printing 5-18
  reading a value's type at a location 5-28
  reading an integer that represents the data type of an argument 9-8
  reading argument ID at a location 5-14, 9-9
  reading argument value at a location 9-2, 9-12, 10-2
  reading integer value of an argument 5-9
  reading IP address value of an argument 5-10
  reading object ID value of an argument 5-11
  reading string value of an argument 5-13
  reading the list returned by a synchronous function 5-104, 6-36
  retrieving error status of an argument 5-12, 5-26, 9-7, 9-11
  retrieving number of arguments 5-5, 9-4
  returning the intersection of two lists 5-20, 9-3
  setting integer value for an argument 5-21
  setting IP address value for an argument 5-22
  setting null value for an argument 5-23, 9-10
  setting object ID value for an argument 5-24
  setting string value for an argument 5-25
  specifying variable arguments 1-4, 9-2
  subtracting two lists 5-19, 5-125, 9-3
  writing text descriptions to an ostream 9-3
Argument, converting to printable string format 5-86
AssignedSvcSecScn, creating ID 5-30
Assigning
  Objectlist to Objectlist 10-3
Asynchronous functions, description 1-1 to 1-3
Attribute
  retrieving an object's attribute list 10-5
Attribute list. *See* Argument list

C

C interface for API functions 1-1
C++
  classes 6-1
  interface for API functions 1-1
Callback handler 4-2, 7-2
Card
  creating an object's card ID 5-33
  reading slot number from object ID 5-87, 8-4
Card status checks preceding circuit provisioning, obtaining retry setting 5-96, 8-12
CardTCA
  creating an object's CardTCA 5-34
Channel

creating an object's channel ID 5-35, 8-28
reading channel number from object ID 5-88, 8-5
Circuit
creating an object's ATM circuit ID 5-31, 8-26
creating an object's circuit ID 8-27
creating an object's Frame Relay circuit ID 5-38, 8-29
reading DLCI number from object ID 5-89, 8-6
reading VCI number from object ID 5-110, 8-21
reading VPI number from object ID 5-112, 8-23
Converting
an argument list to printable string format 5-27
an argument to printable string format 5-86
an enumerated value to printable string format 5-90
an object ID to printable string format 5-114
an object type to printable string format 5-98
text descriptions of an object ID to an object ID 5-130
text descriptions to an object type to an object type 5-131
writing text descriptions of an argument list to an ostream 9-3
writing text descriptions of an object ID to an ostream 8-3
writing text descriptions of an ObjectList to an ostream 10-3
Creating
a null object ID 5-44
an argument list 5-15
an object's APS ID 5-29
an object's AssignedSvcSecScn ID 5-30
an object's ATM circuit ID 5-31, 8-26
an object's ATM LPort ID 5-32
an object's ATM Virtual UNI LPort start VPI value 5-82, 8-52
an object's card ID 5-33, 8-27
an object's CardTCA ID 5-34
an object's channel ID 5-35, 8-28
an object's Customer ID 5-37
an object's DefinePath ID 5-36
an object's E.164 address 8-30
an object's Frame Relay circuit ID 5-38, 8-29
an object's Frame Relay LPort ID 5-40, 8-32
an object's name 8-33 to 8-34
an object's name and type 8-36 to 8-37
an object's NetCac ID 5-41
an object's network ID 5-42, 8-38 to 8-39
an object's network ID from a string 5-43
an object's PerformanceMonitor ID 5-45
an object's Pfdl ID 5-39
an object's PMP circuit leaf ID 5-46
an object's PMP circuit root ID 5-47
an object's PMP soft PVC root ID 5-49
an object's PMP SpvcLeaf ID 8-40 to 8-41
an object's PMPSpvcLeaf ID 5-48
an object's PnniNodeId 5-50, 8-42 to 8-43
an object's PPort ID 5-51, 8-44
an object's PPortTCA ID 5-52
an object's reference time server 5-53, 8-45 to 8-46
an object's Service Name ID 5-54
an object's SMDS address prefix ID 5-63
an object's SMDS alien group address ID 5-55
an object's SMDS alien individual address ID 5-56
an object's SMDS country code ID 5-57
an object's SMDS group address ID 5-58
an object's SMDS group screen ID 5-59
an object's SMDS individual screen ID 5-60
an object's SMDS local individual address ID 5-61
an object's SMDS netwide group address ID 5-62
an object's SMDS SSI individual address ID 5-64
an object's soft PVC circuit ID 5-65
an object's SvcAddress 8-47
an object's SvcAddress ID 5-66
an object's SvcConfig ID 5-67
an object's SvcCUG ID 5-68
an object's SvcCUGMbr ID 5-69
an object's SvcCUGMbr member name 8-35
an object's SvcCUGMbrRule ID 5-70
an object's SvcNetworkId 5-71
an object's SvcNodePrefix ID 5-72
an object's SvcPrefix ID 5-73
an object's SvcSecScn ID 5-74
an object's SvcSecScnActParam ID 5-75
an object's SvcUserPart ID 5-76
an object's switch ID 5-77 to 5-78, 8-49

an object's TrafficDescriptor ID 5-79
an object's TrafficShaper ID 5-80, 8-50 to 8-51
an object's trunk ID 5-81
an object's Vpci ID 8-53 to 8-54
an object's VpciTable ID 5-83
an object's VPN ID 5-84
an ObjectList 5-123
an SMDS object's E.164 address 8-24 to 8-25
attribute IDs in an argument list 5-16, 9-2
values for attributes in an argument list 5-17, 9-2
Customer, creating ID 5-37
CvClient class 6-1, 6-3 to 6-4
CvClient::ApiStatus class 6-1
CvClient::Args class 6-1, 9-1 to 9-3
CvClient::Callback class 6-1, 7-2
CvClient::IpAddr class 6-1, 11-1, 11-2 to 11-3
CvClient::ObjectId class 6-1, 8-1, 8-2 to 8-3
CvClient::ObjectList class 6-1, 10-2
CvClient::Value class 6-1
CvSVCAddress class 12-1, 12-2 to 12-4

D

DefinedPath
creating an object's circuit DefinedPath 5-36
Deleting
a member 3-6 to 3-7, 3-39 to 3-40, 6-17 to 6-20
a pointer to a string 5-133
a pointer to an argument list 5-8
a pointer to an ObjectList 5-119
an object 3-8 to 3-9, 3-41, 6-21 to 6-22
Determining if timeout occurred 4-7, 7-8
Diagnosing
retrieving diagnostic results 6-23 to 6-24, 6-25
starting diagnostics on an object 6-62 to 6-63, 6-64 to 6-65
stopping diagnostics on an object 6-66, 6-67
updating diagnostic parameters 6-68 to 6-69, 6-70
Diagnostics
retrieving 3-10 to 3-11, 3-42 to 3-43
starting 3-30 to 3-31, 3-64 to 3-65
stopping 3-32, 3-66
updating 3-33 to 3-34, 3-67 to 3-68
DLCI
creating an object's Frame Relay circuit ID 5-38, 8-29
reading DLCI number from object ID 5-89, 8-6

E

E.164 address
creating an object's address 8-30
creating for an SMDS object ID 8-24 to 8-25
reading from object ID 8-7
Enumerated value, converting to printable string format 5-90
Equality, comparing argument lists for 5-6, 9-3
Equality, comparing ObjectLists for 10-3
Errors
determining if an error occurred in the response for a synchronous request 6-37
determining if any argument in an argument list contains an error status 5-7, 9-5
determining if any object in an ObjectList contains an error status 5-118, 10-7
obtaining an error message 5-91, 6-26
retrieving error code for an argument in argument list 5-12, 5-26, 9-7, 9-11
retrieving error code for an object in ObjectList 5-121, 5-128, 10-11, 10-12
Establishing a session with the Provisioning Server 2-3 to 2-4, 6-3 to 6-4
Existence of an argument in an argument list 9-6
Existence of an object in an ObjectList 10-8
Extended Super Frame. *See* Pfdl

F

File descriptors, processing activity on 4-5 to 4-6, 7-6 to 7-7
Functions
asynchronous, description 1-1 to 1-3
naming conventions 1-3
operational functions 1-4, 1-5 to 1-6, 3-1 to 3-63, 6-8 to 6-61
select loop processing functions 1-4, 1-6, 4-1 to 4-7, 7-1 to 7-8
session control functions 1-4, 2-1 to 2-6, 6-2 to

6-7
synchronous, description 1-1
utility functions 1-4, 1-6 to 1-9, 8-1 to 8-55, 9-1 to 9-12

G

Getting
an error message 5-91, 6-26
an object 3-14 to 3-15, 3-46 to 3-47
an object, synchronous 6-31
an ObjectList 3-12 to 3-13
an ObjectList, asynchronous 6-29 to 6-30
an ObjectList, synchronous 3-44 to 3-45, 6-32
information for select system call 4-3 to 4-4, 7-4 to 7-5
OperInfo attributes, asynchronous 6-33 to 6-34
OperInfo attributes, synchronous 6-35

I

Inequality, comparing argument lists for 9-3
Inequality, comparing ObjectLists for 10-3
Intersection, of two argument lists 5-20, 9-3
Intersection, of two ObjectLists 5-126, 10-3
IP address
reading from object ID 5-95, 5-103, 5-108, 8-11, 8-16, 8-19
reading value from argument list 5-10
setting value in argument list 5-22

L

Listing an object 3-18 to 3-25, 3-49 to 3-57, 6-38 to 6-53
LPort
creating an object's ATM LPort ID 5-32
creating an object's ATM Virtual UNI LPort start VPI value 5-82, 8-52
creating an object's Frame Relay LPort ID 5-40, 8-32
reading an object's ATM Virtual UNI LPort start VPI value 5-106, 8-17
reading number from object ID 5-92, 8-8

M

Management DLCI
reading number from object ID 5-111, 8-22
Member
adding 3-2 to 3-3, 3-35 to 3-36, 6-9 to 6-12
deleting 3-6 to 3-7, 3-39 to 3-40, 6-17 to 6-20
Modifying an object 3-28 to 3-29, 3-60 to 3-61, 6-54 to 6-59
Modifying an object in an ObjectList, asynchronous 6-56 to 6-57
Modifying an object in an ObjectList, synchronous 6-60
Modifying an ObjectList, asynchronous 3-26 to 3-27
Modifying an ObjectList, synchronous 3-58 to 3-59

N

Name
creating an object's name 8-33 to 8-34
creating an object's SvcCUGMbr member name 8-35
reading name from object ID 8-9
Name and type, creating 8-36 to 8-37
Naming conventions for functions 1-3
NavisXtend Provisioning Server. *See* Provisioning Server
NetCac, creating ID 5-41
Network ID
creating an object's network ID 5-42, 8-38 to 8-39
creating an object's network ID from a string 5-43
reading IP address from object ID 5-95, 8-11
Next object, retrieving 3-62 to 3-63, 6-61
Null object, creating 5-44
Null value, setting 5-23, 5-127, 9-10

O

Object
- adding 3-4 to 3-5, 3-37 to 3-38, 6-13 to 6-16
- deleting 3-8 to 3-9, 3-41, 6-21 to 6-22
- diagnosing 6-23 to 6-24, 6-25, 6-68 to 6-69, 6-70
- listing 3-18 to 3-25, 3-49 to 3-57, 6-38 to 6-53
- modifying 3-28 to 3-29, 3-60 to 3-61, 6-54 to 6-59
- removing from an ObjectList 10-10
- retrieving 3-14 to 3-15, 3-46 to 3-47
- retrieving attributes, synchronous 6-31
- retrieving next object 3-62 to 3-63, 6-61
- starting diagnostics 6-62 to 6-63, 6-64 to 6-65
- stopping diagnostics 6-66, 6-67

Object ID
- converting from text descriptions 5-130
- converting to printable string format 5-114
- creating E.164 address for 8-24 to 8-25
- printing 5-113
- reading a name from 5-93
- reading an SMDS address from 5-85
- reading an SvcAddress from 5-107
- reading an SvcCUGMbr member name from 5-94, 8-10
- reading card slot number from 5-87, 8-4
- reading channeI number from 5-88
- reading channel number from 8-5
- reading DLCI number from 5-89, 8-6
- reading E.164 address from 8-7
- reading LPort number from 5-92, 8-8
- reading name from 8-9
- reading network IP address from 5-95, 8-11
- reading object type from 5-97, 8-55
- reading PMP SpvcLeaf number from 8-13
- reading PMPSpvcLeaf number from 5-99
- reading PNNI Peer Group string from 5-100, 8-14
- reading PPort number from 5-101, 8-15
- reading reference time server IP address from 5-103, 8-16
- reading SMDS address prefix from 5-102
- reading SvcAddress from 8-18
- reading switch IP address from 5-108, 8-19
- reading TrafficShaper identifier value from 5-109
- reading VCI number from 5-110, 8-21
- reading Vpci number from 5-111, 8-22
- reading VPI number from 5-112, 8-23
- setting the object type 8-51
- writing text descriptions of an object ID to an ostream 8-3

Object type
- converting from text description 5-131
- converting to printable string format 5-98
- setting 8-51

ObjectList
- adding objects to 5-115, 10-4
- adding two lists 10-3
- assigning 10-3
- class 10-2
- comparing lists for equality 10-3
- comparing lists for inequality 10-3
- creating a list 5-123
- deleting pointer to 5-119
- determining error status 5-118, 10-7
- determining if specified object exists 10-8
- modifying objects in, asynchronous 6-56 to 6-57
- modifying objects in, synchronous 6-60
- modifying, asynchronous 3-26 to 3-27
- modifying, synchronous 3-58 to 3-59
- printing 5-124
- reading integer value of an object 5-120
- reading object ID at a location 5-122, 10-9
- reading the list returned by a synchronous function 5-105
- removing an object 10-10
- retrieving 3-12 to 3-13
- retrieving an object's attribute list 10-5
- retrieving error status of an object 5-121, 5-128, 10-11, 10-12
- retrieving number of objects 5-117, 10-6
- retrieving, asynchronous 6-29 to 6-30
- retrieving, synchronous 3-44 to 3-45, 6-32
- returning the intersection of two lists 5-126, 10-3
- returning the subtraction of two lists 5-129
- setting null value for an argument 5-127
- subtracting two lists 10-3
- writing text descriptions to an ostream 10-3

Objects
- adding to ObjectList 5-115

Operational functions 1-4, 1-5 to 1-6, 3-1 to 3-63, 6-8 to 6-61
OperInfo
  retrieving attributes, asynchronous 6-33 to 6-34
  retrieving attributes, synchronous 6-35
Ostream
  writing text descriptions of an argument list to 9-3
  writing text descriptions of an object ID to 8-3
  writing text descriptions of an ObjectList to 10-3

P

PerformanceMonitor, creating ID 5-45
Pfld, creating ID 5-39
PMP circuit leaf, creating ID 5-46
PMP circuit root, creating ID 5-47
PMP soft PVC root, creating ID 5-49
PMPSpvcLeaf
  creating an object's PMP SpvcLeaf ID 8-40 to 8-41
  creating an object's PMPSpvcLeaf ID 5-48
  reading number from object ID 8-13
  reading PMPSpvcLeaf number from object ID 5-99
PnniNode
  creating 5-50, 8-42 to 8-43
  reading PNNI Peer Group string from object ID 5-100, 8-14
PPort
  creating an object's PPort ID 5-51, 8-44
  reading number from object ID 5-101, 8-15
PPortTCA
  creating an object's PPortTCA 5-52
Printing
  an argument list 5-18
  an object ID 5-113
  an ObjectList 5-124
Processing activity on file descriptors 4-5 to 4-6, 7-6 to 7-7
Provisioning Server
  adding a member 3-2 to 3-3, 3-35 to 3-36, 6-9 to 6-12
  adding an object 3-4 to 3-5, 3-37 to 3-38, 6-13 to 6-16
callback handler 4-2, 7-2
deleting a member 3-6 to 3-7, 3-39 to 3-40, 6-17 to 6-20
deleting an object 3-8 to 3-9, 3-41, 6-21 to 6-22
determining if timeout occurred 4-7, 7-8
listing an object 3-18 to 3-25, 3-49 to 3-57, 6-38 to 6-53
modifying an object 3-28 to 3-29, 3-60 to 3-61, 6-54 to 6-59
modifying an object in an ObjectList, asynchronous 6-56 to 6-57
modifying an object in an ObjectList, synchronous 6-60
modifying an ObjectList, asynchronous 3-26 to 3-27
modifying an ObjectList, synchronous 3-58 to 3-59
ObjectList class 10-2
obtaining an error message 5-91, 6-26
obtaining current retry setting for card status checks preceding circuit provisioning 5-96, 8-12
obtaining information for select system call 4-3 to 4-4, 7-4 to 7-5
processing activity on file descriptors 4-5 to 4-6, 7-6 to 7-7
real time status 3-16 to 3-17, 3-48
retreiving diagnostic results 6-23 to 6-24, 6-25
retrieving an object 3-14 to 3-15, 3-46 to 3-47
retrieving an object, synchronous 6-31
retrieving an ObjectList 3-12 to 3-13
retrieving an ObjectList, asynchronous 6-29 to 6-30
retrieving an ObjectList, synchronous 3-44 to 3-45, 6-32
retrieving object diagnostic results 3-10 to 3-11, 3-42 to 3-43
retrieving OperInfo attributes, asynchronous 6-33 to 6-34
retrieving OperInfo attributes, synchronous 6-35
retrieving the next object 3-62 to 3-63, 6-61
session, establishing 2-3 to 2-4, 6-3 to 6-4
session, terminating 2-2, 6-7
setting number of retries to check card status preceding circuit provisioning 2-6, 5-132, 5-132, 6-6
starting diagnostics for an object 6-62 to 6-63,

6-64 to 6-65
starting diagnostics on an object 3-30 to 3-31, 3-64 to 3-65
stopping diagnostics for an object 6-66, 6-67
stopping diagnostics on an object 3-32, 3-66
updating diagnostic parameters 6-68 to 6-69, 6-70
updating diagnostics on an object 3-33 to 3-34, 3-67 to 3-68
updating the database 2-5, 6-5

R

Reading
a card slot number out of an object ID 5-87, 8-4
a channel number out of an object ID 5-88, 8-5
a circuit VCI out of an object ID 5-110, 8-21
a circuit VPI out of an object ID 5-112, 8-23
a DLCI number out of an object ID 5-89, 8-6
a name out of an object ID 5-93, 8-9
a network IP address out of an object ID 5-95, 8-11
a PMP SpvcLeaf number out of an object ID 8-13
a PNNI Peer Group string out of an object ID 5-100, 8-14
a PPort number out of an object ID 5-101, 8-15
a reference time server IP address out of an object ID 5-103, 8-16
a string value of argument in an argument list 5-13
a switch IP address out of an object ID 5-108, 8-19
a TrafficShaper identifier value out of an object ID 5-109
a value's type at a location in an argument list 5-28
a Vpci number out of an object ID 5-111, 8-22
an argument ID at a location in an argument list 5-14, 9-9
an argument ID at a location in an ObjectList 10-9
an argument value at a location in an argument list 9-12
an E.164 address out of an object ID 8-7
an integer that represents the data type of an argument in an argument list 9-8
an integer value of an object in an ObjectList 5-120
an integer value of argument in an argument list 5-9
an IP address value of argument in an argument list 5-10
an LPort number out of an object ID 5-92, 8-8
an object ID at a location in an ObjectList 5-122
an object ID value of argument in an argument list 5-11
an object type out of an object ID 5-97, 8-55
an PMPSpvcLeaf number out of an object ID 5-99
an SMDS address out of an object ID 5-85
an SMDS address prefix out of an object ID 5-102
an SvcAddress out of an object ID 5-107, 8-18
an SvcCUGMbr member name out of an object ID 5-94, 8-10
argument list returned by a synchronous function 5-104, 6-36
argument value at a location in an argument list 9-2, 10-2
ObjectList returned by a synchronous function 5-105
the start VPI value out of an object ID for Virtual UNI LPort 5-106, 8-17
Real time status
retrieving 3-16 to 3-17, 3-48
Reference time server
creating 5-53, 8-45 to 8-46
reading IP address from object ID 5-103, 8-16
Removing
object from an ObjectList 10-10
Required parameters, determining if one was omitted 5-104, 6-36
Retries to check card status preceding circuit provisioning, setting 2-6, 5-132, 5-132, 6-6
Retrieving
an object 3-14 to 3-15, 3-46 to 3-47
an object's attribute list 10-5
an ObjectList 3-12 to 3-13
an ObjectList, asynchronous 6-29 to 6-30
an ObjectList, synchronous 3-44 to 3-45, 6-32
error status of an argument in an argument list 5-12, 5-26, 9-7, 9-11

error status of an object in an ObjectList 5-128, 10-11, 10-12
error status of an objectin an ObjectList 5-121
number of arguments in an argument list 5-5, 9-4
number of objects in an ObjectList 5-117, 10-6
object attributes, synchronous 6-31
OperInfo attributes, asynchronous 6-33 to 6-34
OperInfo attributes, synchronous 6-35
real time status 3-16 to 3-17, 3-48
the next object 3-62 to 3-63, 6-61
Retrieving object diagnostic results 3-10 to 3-11, 3-42 to 3-43
Retrieving real time status 3-16 to 3-17, 3-48
Returning the intersection of two argument lists 5-20, 9-3
Returning the intersection of two ObjectLists 5-126, 10-3
Returning the subtraction of two ObjectLists 5-129

S

Select loop processing functions 1-4, 1-6, 4-1 to 4-7, 7-1 to 7-8
Service Name, creating ID 5-54
Session
establishing 2-3 to 2-4, 6-3 to 6-4
terminating 2-2, 6-7
Session control functions 1-4, 2-1 to 2-6, 6-2 to 6-7
Setting
a null value for argument in an argument list 5-23, 9-10
a null value for argument in an ObjectList 5-127
a string value for argument in an argument list 5-25
an integer value for argument in an argument list 5-21
an IP address value for argument in an argument list 5-22
an object ID value for argument in an argument list 5-24
the type of an object ID 8-51
SMDS address prefix, creating ID 5-63
SMDS address prefix, reading from object ID 5-102
SMDS alien group address, creating ID 5-55
SMDS alien individual address, creating ID 5-56
SMDS country code, creating ID 5-57
SMDS group address, creating ID 5-58
SMDS group screen, creating ID 5-59
SMDS individual screen, creating ID 5-60
SMDS local individual address, creating ID 5-61
SMDS netwide group address, creating ID 5-62
SMDS objects, creating an object's E.164 address 8-24 to 8-25
SMDS SSI individual address, creating ID 5-64
Soft PVC circuit, creating ID 5-65
Start VPI for Virtual UNI LPort 5-82, 5-106, 8-17, 8-52
Starting a session with the Provisioning Server 2-3 to 2-4, 6-3 to 6-4
Starting diagnostics on an object 3-30 to 3-31, 3-64 to 3-65
Stopping diagnostics on an object 3-32, 3-66
String, deleting pointer to 5-133
Subtracting two argument lists 5-19, 5-125, 9-3
Subtracting two ObjectLists 10-3
Subtraction
of two ObjectLists 5-129
SvcAddress
creating an object's address 5-66, 8-47
reading from object ID 8-18
SvcConfig, creating ID 5-67
SvcCUG, creating ID 5-68
SvcCUGMbr, creating ID 5-69
SvcCUGMbrRule, creating ID 5-70
SvcNetworkId, creating ID 5-71
SvcNodePrefix, creating ID 5-72
SvcPrefix, creating ID 5-73
SvcSecScn, creating ID 5-74
SvcSecScnActParam, creating ID 5-75
SvcUserPart, creating ID 5-76
Switch
creating ID 5-77 to 5-78, 8-49
reading IP address from object ID 5-108, 8-19
Synchronous functions, description 1-1

T

Terminating a session with the Provisioning Server 2-2, 6-7
Timeout, determining if occurred 4-7, 7-8
TrafficDescriptor, creating ID 5-79
TrafficShaper
  creating an object's TrafficShaper ID 5-80, 8-50 to 8-51
  reading index value from object ID 5-109
Trunk, creating ID 5-81

U

Updating diagnostics on an object 3-33 to 3-34, 3-67 to 3-68
Updating the database 2-5, 6-5
Utility functions 1-4, 1-6 to 1-9, 8-1 to 8-55, 9-1 to 9-12

V

Variable argument list 9-2
VpciTable
  creating an object's Vpci ID 8-53 to 8-54
  reading number from object ID 5-111, 8-22
VpciTable, creating ID 5-83
VPI
  creating the start VPI value for Virtual UNI LPort 5-82, 8-52
  reading the start VPI value for Virtual UNI LPort 5-106, 8-17
VPN, creating ID 5-84

W

Writing
  text descriptions of an argument list to an ostream 9-3
  text descriptions of an object ID to an ostream 8-3
  text descriptions of an ObjectList to an ostream 10-3