NavisXtend Provisioning Server User's Guide

Ascend Communications, Inc.

Product Code: 80065
Revision 00
November 1998

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

List of Figures

List of Tables

About This Guide

The *NavisXtend Provisioning Server User's Guide* describes how to use the NavisXtend™ Provisioning Server Application Toolkit to develop a *provisioning client* — an application that runs on a workstation in an Ascend™ network and interacts with a Provisioning Server. You use the client to query and configure switch nodes, cards, physical ports, logical ports, circuits, and other objects. The Application Toolkit includes a special series of libraries and header files that support client development.

In addition, the *NavisXtend Provisioning Server User's Guide* describes how to use the Toolkit Command Line Interface (CLI) to develop a *provisioning script* — a set of shell commands used for either interactive or batch provisioning of network objects.

The *NavisXtend Provisioning Server User's Guide* also describes how to use the enterprise-specific MIB, which provides SNMP access to the Provisioning Server.

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.

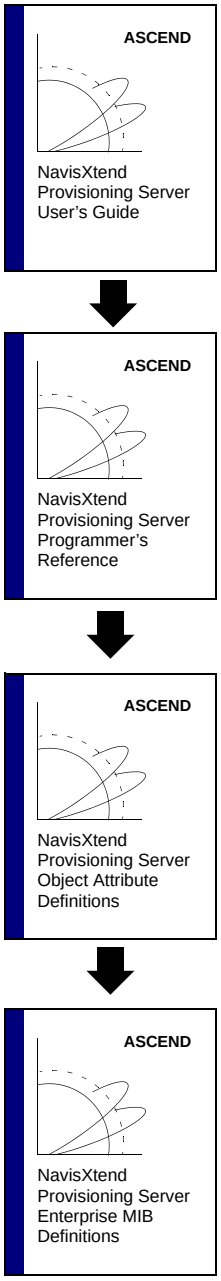
To develop a provisioning client, you need to be familiar with programming in C or C++ in a UNIX environment. Programming experience is *not* required if you plan to use the Command Line Interface only.

To use the SNMP MIB, you need to be familiar with the SNMP protocol, operations supported by the protocol, and MIB structure in general.

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
Chapter 1	General aspects of the NavisXtend Provisioning Server and the client and how they interact with other components on the network. This chapter describes the interface, features, and typical applications of the NavisXtend Provisioning Server Application Toolkit.
Chapter 2	How to install and administer the various components of the Provisioning Server system. This chapter also describes the steps required to develop a provisioning application.
Chapter 3	How to use the CLI.
Chapter 4	How to use the SNMP MIB.

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.
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.

New Features/Functions	Description
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.
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	**GetOperInfo**, 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
Removed the containment hierarchy tables that listed the parent-child relation used to build object IDs to name objects in the network. This information is now included in attribute matrixes in the *NavisXtend Provisioning Server Object Attribute Definitions*.	Previously in Appendix A
Real time status and diagnostic operational function descriptions.	1
Real time status and diagnostic CLI command descriptions	3
New object descriptions.	1
SNMPv2 support description.	4

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
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. Book titles, new terms, and emphasized text.	*UserArg* argument *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*™.

1

Overview

This chapter describes what you need to know before developing an NavisXtend Provisioning client or a provisioning script. It describes the features of the Application Programming Interface (API) and presents some basic procedures that show how to perform tasks with the API.

NavisXtend Provisioning Server

The Ascend NavisXtend Provisioning Server is based on a client-server network management architecture:

NavisXtend Provisioning Client — The client is an application responsible for generating requests to provision Ascend network components. Much of the provisioning functionality of NavisCore is available: the client can query and configure Frame Relay, ATM, ATM Network Interworking, and SMDS objects including switch nodes, cards, physical ports, logical ports, circuits, and so on.

NavisXtend Provisioning Server — The server is a UNIX process that responds to requests from NavisXtend Provisioning clients and updates the Ascend switches and the NavisCore database.

The NavisXtend Provisioning client runs on a workstation and interacts with an NavisXtend Provisioning Server. The NavisXtend Provisioning Server responds to client requests to manage Ascend switches and updates the NavisCore database. While there can be multiple instances of NavisCore, the NavisXtend Provisioning client, and the NavisXtend Provisioning Server running on the network, any client typically interacts with only one Provisioning Server at a time. Each Provisioning Server can manage all the Ascend switches and update the shared NavisCore database.

Because the Provisioning Server shares the same Sybase database with other NavisCore processes, the server should reside in the same TCP/IP subnetwork as NavisCore and the Sybase database. As the Provisioning Server makes changes to the Ascend network, it maintains consistency with NavisCore. The Provisioning Server uses a locking mechanism so that NavisXtend Provisioning clients and other NavisCore processes that share the same database cannot update the same object at the same time.

Figure 1-1 shows the relationship among the NavisXtend Provisioning client, the NavisXtend Provisioning Server, and other components on the network.

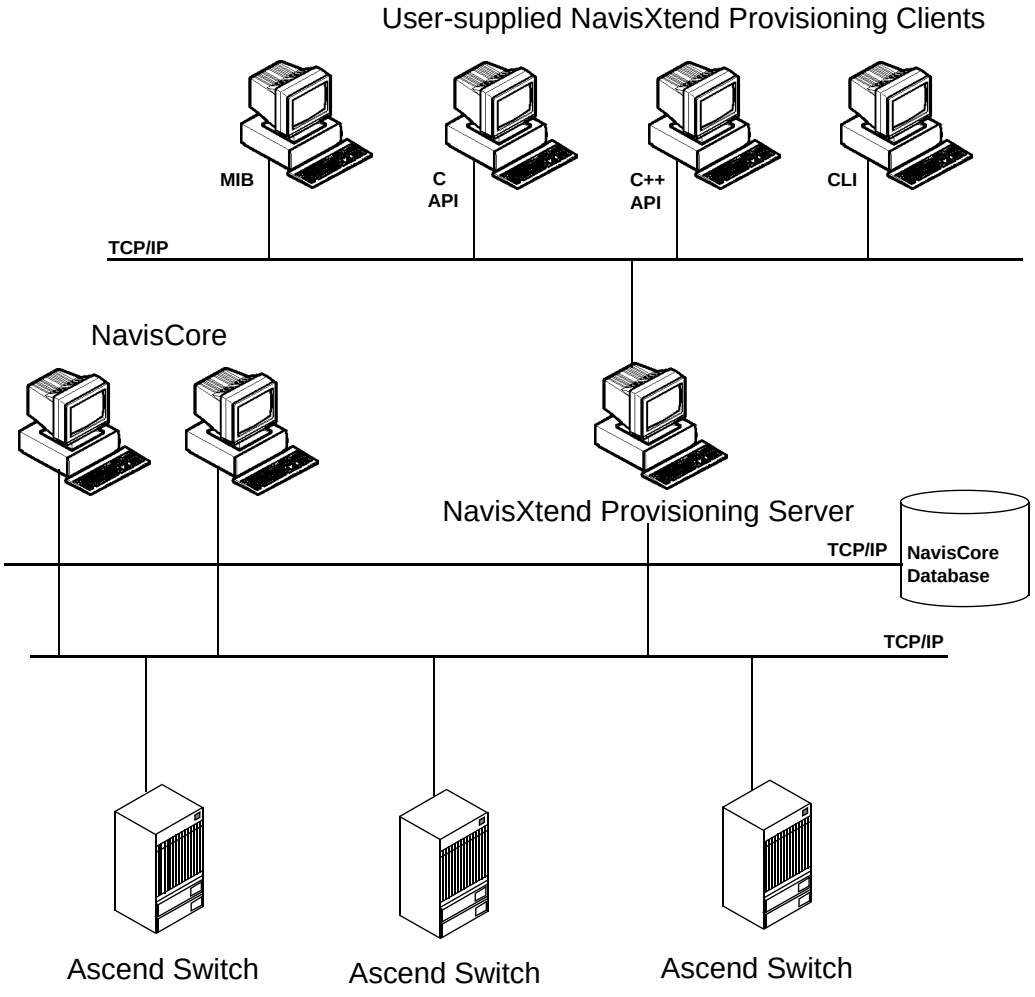


Figure 1-1. Components in the NavisXtend Provisioning Server System

The NavisXtend Provisioning Server product includes the following software:

NavisXtend Provisioning Server — Installed and maintained on a UNIX workstation on a TCP/IP network.

NavisXtend Provisioning Server Application Toolkit — Installed and used by the application developer to create an NavisXtend Provisioning client or script that submits requests to the Provisioning Server.

The next section describes the NavisXtend Provisioning Server Application Toolkit.

Application Toolkit

The Application Toolkit provides the following components:

API — Used by an application developer to write a new Provisioning client or to integrate a client into an existing provisioning system.

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. And, the toolkit provides both a synchronous and asynchronous version of each function that performs provisioning operations.

For details on how to use the API to develop a Provisioning client, see Chapter 2 in this guide and to the *NavisXtend Provisioning Server Programmer's Reference*.

Command Line Interface (CLI) — Used to build a provisioning script. The CLI is a set of command-line programs that hide the code details of the API. Users can issue these commands from any UNIX shell to provision network objects in either interactive or batch mode.

For details on how to use the CLI to develop a provisioning script, see Chapter 3.

Enterprise-specific MIB — Used to access switches in the network via SNMP commands. The MIB supports all the attributes and functionality of the API and provides access via SNMP **get**, **set**, and **get-next** operations.

For details on how to use the MIB to develop a provisioning script, see Chapter 4 in this guide and to the *NavisXtend Provisioning Server Enterprise MIB Definitions*.

Figure 1-2 illustrates how the Application Toolkit is organized.

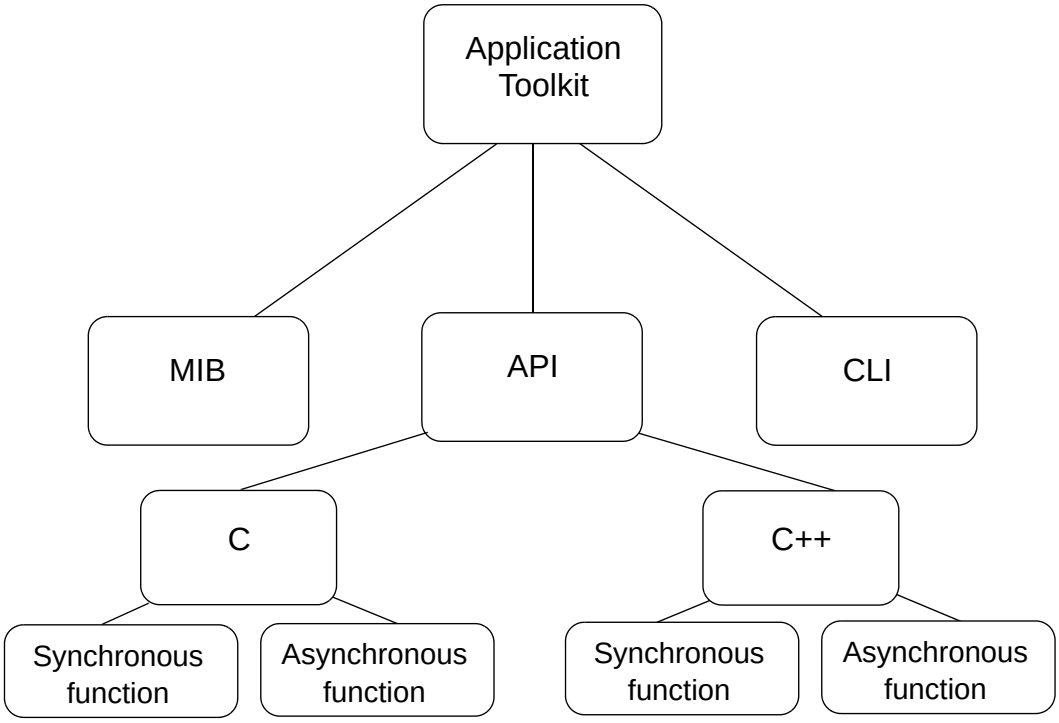


Figure 1-2. Application Toolkit Organization

Synchronous and Asynchronous Functions

The Application Toolkit provides two communication methods for each API function that performs provisioning operations. You can issue either:

- A *synchronous* function and wait for a response to your request. 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.
- An *asynchronous* function and perform other operations while your request is being processed.

Figure 1-3 shows the flow between the application and the Provisioning Server for a synchronous request. The application does not regain control of the program until the response returns from the server.

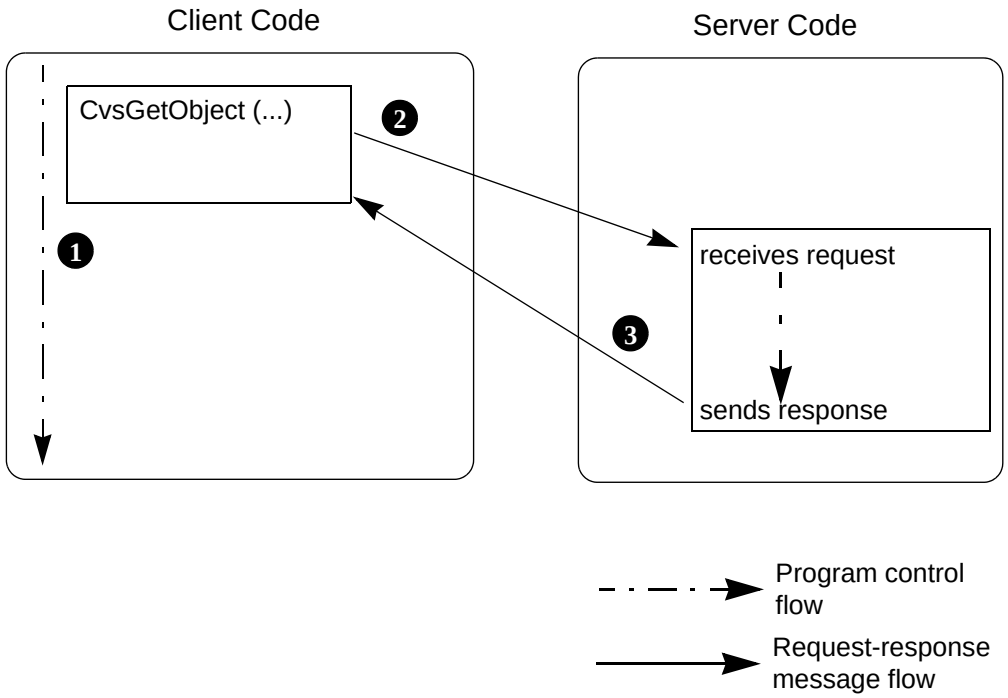


Figure 1-3. Flow Between Client and Server for a Synchronous Function

With an asynchronous function, the application continues with other work while waiting for the response. The application supplies a callback handler to the API. The function invokes this callback handler function to deliver the response from the server.

Figure 1-4 shows the flow between the application and the Provisioning Server for an asynchronous request.

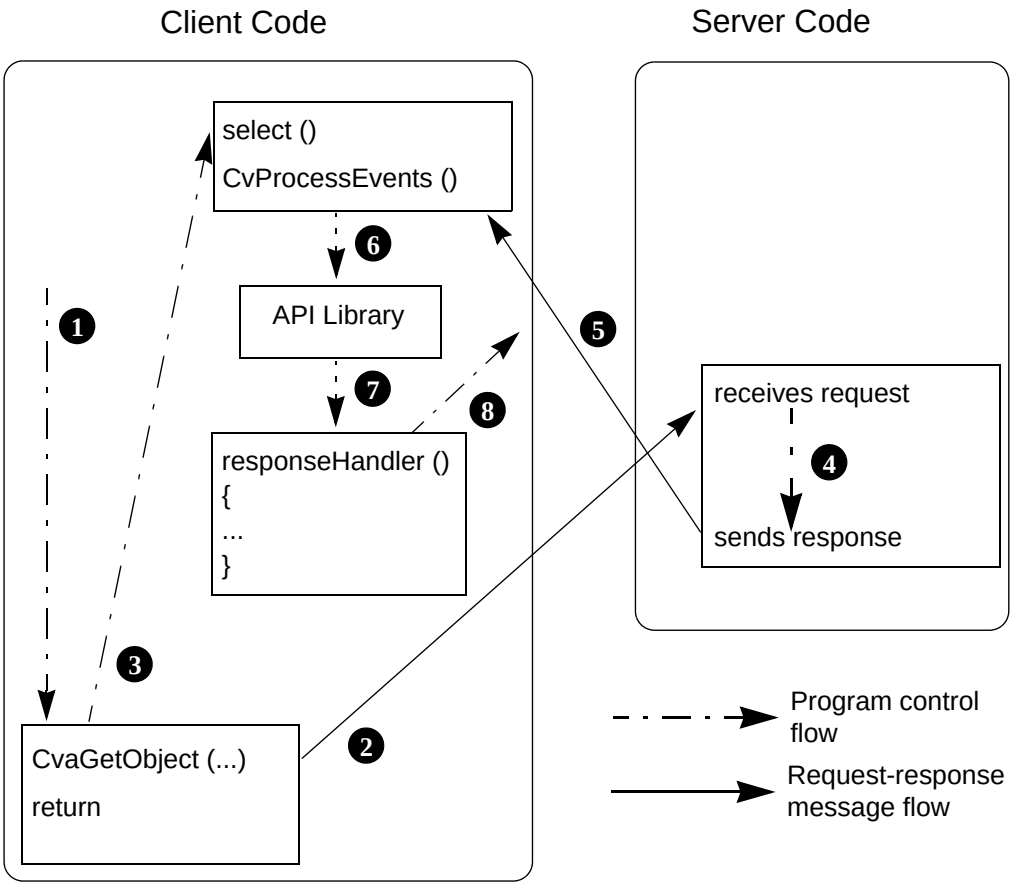


Figure 1-4. Flow Between Client and Server for an Asynchronous Function

During an asynchronous request, the following steps occur:

Step 1 The application code issues an asynchronous function.

Step 2 The application sends the request to the Provisioning Server.

Step 3 The application immediately returns to the **select** loop. If the application calls **select** directly, it needs to know what file descriptor is being used for communication with the server. The application can issue **CvGetSelectInfo** to obtain the information needed to pass to **select**.

Step 4 The Provisioning Server processes the request.

Step 5 The Provisioning Server sends the response to **select**.

Step 6 When **select** notifies the application of pending messages from the server, the application issues **CvProcessEvents**, which goes into the API library to receive and process the response.

Step 7 In turn, the API passes the response to the client response handler.

Step 8 The client code continues.

Synchronous functions involve less coding but may be less efficient, because the process 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 make sure 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**.

Functions That Take an Argument List

Most of the C functions that perform provisioning operations on network components take one or more attributes. The attributes are specified in an argument list. 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.

Function Names

The name of each function varies, depending on the version of the function. Table 1-1 shows the different names for the same function that adds an object to the database.

Table 1-1. Naming Conventions for Toolkit Functions

Version	Function Name
Asynchronous C function	**CvaAddObject**
Synchronous C function	**CvsAddObject**

Table 1-1. Naming Conventions for Toolkit Functions

Version	Function Name
C++ function	**CvClient::addObject**
CLI command	**cvadd**

Toolkit Functionality

The toolkit functions are divided into the following groups:

- *Session Control* functions open and close sockets and control session settings.
- *Operational* functions perform provisioning operations on 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 sections describe the toolkit functions by group.

Session Control Functions

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

The CLI uses environment variables to specify session control settings (for details, see "Setting Environment Variables" on page 2-11).

The session control functions are:

Connect, **open** — Establishes a session with the Provisioning Server.

Close — Terminates a session with the Provisioning Server.

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

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

Operational Functions

Operational functions perform provisioning operations on network components. Most operational functions of the API have a CLI command counterpart.

The operational functions and CLI commands are:

AddObject (Object ID, Attributes) — Creates an object in the database and (optionally) in the switch.

AddMember (Object ID, Object ID) — Adds a member to an object list.

DeleteObject (Object ID) — Deletes an object from the database and (optionally) from the switch.

DeleteMember (Object ID, Object ID) — Deletes a member from an object list.

GetDiagObject (Object ID, Attributes) — Retrieves object diagnostic results in the network.

GetErrorMsg — Returns an error message.

GetList (Object ID, ObjectList, Attributes) — Retrieves the values of specific object attributes that are specified as an ObjectList.

GetObject (Object ID, Attributes) — Retrieves the values of specific attributes of an object.

GetOperInfoObject (Object ID, Attributes) — Retrieves the values of specific OperInfo attributes from the switch.

GetResponseArgs — Returns the argument list returned by a synchronous function.

ListAllContainedObjects (Object ID) — Queries the database for a list of objects of any type that are contained by a specified parent.

ListContainedObjects (Object ID, type, Attributes) — Queries the database for a list of objects of the given type that are contained by a specified parent.

ModifyObject (Object ID, Attributes) — Modifies specific attributes of an object in the database and (optionally) in the switch.

ModifyList (Object ID, ObjectList, Attributes) — Modifies specific object attributes that are specified as an ObjectList in the database and (optionally) in the switch.

StartDiagObject (Object ID, Attributes) — Starts diagnostics on an object in the network.

StopDiagObject (Object ID, Attributes) — Stops diagnostics on an object in the network.

UpdateDiagObject (Object ID, Attributes) — Modifies diagnostics on an object in the network.

The following operational functions are used by the API only:

NextObject — Retrieves the next object in a list of objects.

The operational functions are supported for most target object types, with a few restrictions. For example, you cannot specify a switch when you issue an Add or Delete command, because the Provisioning Server does not support adding or deleting switches.

Select Loop Processing Functions

Select Loop Processing functions support loop processing of the **select** system call. The functions that support **select** loop processing are:

Callback Handler — Is the prototype for a function supplied by the client. The APIs call these callback handlers to deliver a response to an asynchronous request. CVCBH is the CvArgs callback function, and OLCBH is the ObjectList callback function.

GetSelectInfo — Obtains information needed to pass to a **select** system call.

ProcessEvents — Processes activity on file descriptors to receive responses from an asynchronous request.

Timeout — Determines if an outstanding asynchronous request timed out.

Utility Functions

Utility functions build argument lists, handle initialization, and manage storage. The utility functions are:

ArgsMake — Creates an argument list.

ArgsMakeVals, **ArgsMakeIds** — Creates and adds arguments to an argument list.

ArgsFree — Deletes a pointer to an argument list created either explicitly or implicitly by another function.

ArgsSet*AttrType* — A series of functions that add or modify an argument in an argument list.

ArgsGet*AttrType* — A series of functions that read values out of an argument list.

ArgsCount — Retrieves the number of arguments in an argument list.

ArgsIdAt — Retrieves a specified argument ID in an argument list.

ArgsTypeAt, **ArgsValueAt** — Retrieves the type or value of a specified argument in an argument list.

ArgsErrorIndex — Indicates if any argument in an argument list has an error status.

ArgsExist — Determines if a specified argument exists in an argument list.

ArgsGetStatus, **ArgsStatusAt** — Returns the error status code of a specified argument in an argument list.

ArgsEqual — Compares two argument lists for equality.

ArgsCombine — Adds two argument lists, resulting in a new argument list that combines both sets of arguments. When an argument exists in both lists, the value from the second list is used.

ArgsRemove — Subtracts two argument lists, resulting in a new argument list that contains the arguments from the first list that were not in the second list.

ArgsSelect — Returns the intersection of two argument lists, resulting in a new argument list that contains the arguments that existed in both lists. Each argument uses the value from the first list.

ArgsToString — Converts an argument list to a printable string format.

StringFree — Deletes a pointer to a string returned by **ArgsToString** and **ObjectIdToString**.

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

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

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

CreateChanPerformanceMonitorId — Creates a CVT_ChanPerformanceMonitor object

Create*ObjectType*Id, **set*ObjectType*** — A series of functions that create an object identifier of a specified type.

CreateNetworkIdFromString, **CreateSwitchIdFromString** — Creates an object identifier of a specified type, based on a text description.

ObjectIdToString — Converts an object identifier to text descriptions.

Get*ObjectTypeValue*, **get*ObjectType*** — A series of functions that read a specified value out of an object identifier.

GetNumRetries — Returns the number of retries to check card status preceding circuit provisioning requests.

ObjectIdToPrint — Prints the text description of an object identifier to a file.

ObjectListAdd — Adds a CvObjectId/CvArgs to an ObjectList.

ObjectListCombine — Returns the union of two ObjectLists.

ObjectListCount — Retrieves the number of objects in the ObjectList.

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

ObjectListFree — Deletes a pointer to an ObjectList.

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

ObjectListIdAt — Retrieves the objectId at a specified position in the ObjectList.

ObjectListMake — Creates an ObjectList.

ObjectListPrint — Prints the contents of CvObjectList to a file.

GetArgumentName — Converts an argument ID to a printable string format.

GetEnumName — Converts an enumerated value to a printable string format.

GetObjectTypeName — Converts an object type to a printable string format.

ParseObjectId — Converts text descriptions of an object to an object identifier.

ParseObjectType — Converts a text description to an object type.

Managed Objects

Managed objects are network components managed on the network. Each managed object is represented by its *object identifier* (object ID), which is expressed as a concatenated, ordered list of type-value identifiers, each separated by periods. To specify an object ID, you first specify the object's parent (if any), including the parent type and value. Then, you specify the child type and value.

For example, an object ID for a PPort would be expressed as:

```
switch.100.101.102.103.card.6.pport.4
```

The object is identified by identifying its parent in the containment hierarchy (switch.100.101.102.103.card.6.), and then identifying the object relative to that parent (pport.4).

Note that the numbering of an object is *not* a globally unique ID; rather, it is relative to the parent object. Thus, this PPort is expressed as the fourth PPort of the card:

```
. . . pport.4
```

When you specify a switch parent, you can identify it in either of the following ways:

- IP address, using switch as the type and the switch's IP address as the value:

switch.100.101.102.103.card.6.pport.4

- String name, using swName as the type and the switch's name as the value:

  swName.abcdefg.card.6.pport.4

If the switch string name contains one of the following special characters: ` * ! { } ( ) $ & ; \ | " or blank character, you must enclose at least the special character with /" characters:

swName.my/" /"switch.card.6.pport.4

If the switch string name contains a period, you must enclose the entire string with /" characters:

swName./"my.switch/".card.6.pport.4

You do not need to enclose the following special characters: + = - _ @ # ^ % , : [ ] / ~

Keep in mind that if a switch name is not unique among networks, a command issued using the switch string name is executed on the first switch found with that name. To ensure that the command is executed on a particular switch, identify the object using the switch's IP address.

In C, an object is represented as a data structure that is manipulated using utility functions. In C++, an object is represented by a class that is manipulated using member functions. For the CLI, an object is represented by string representation.

Object Types

Table 1-2 list the object types supported by the Provisioning Server. These object types are defined in the file CvObjectType.H.

The names of several objects differ from the names used in NavisCore.

Table 1-2. Object Types Supported by the Provisioning Server

Object Name	Enumerated Object Type (API)	Object Type (CLI)
Automatic Protection Switching	CVT_Aps	Aps
Assigned SVC Security Screen	CVT_AssignedSvcSecScn	AssignedSvcSecScn
Card	CVT_Card	Card
Card Threshold Crossing Alarm	CVT_CardTca	CardTca

Table 1-2. Object Types Supported by the Provisioning Server (Continued)

Object Name	Enumerated Object Type (API)	Object Type (CLI)
Channel	CVT_Channel	Channel
Channel Performance Monitor	CVT_ChanPerformanceMonitor	PM
Circuit	CVT_Circuit	Circuit
Customer	CVT_Customer	Customer
Circuit Defined Path	CVT_DefinedPath	DefinedPath
Logical Port	CVT_LPort	Lport
Network Connection Admission Control	CVT_NetCac	NetCac
Network	CVT_Network	Network
Performance Monitor	CVT_PerformanceMonitor	PM
Pnni Node	CVT_PnniNode	PnniNode
Extended Super Frame Data Link	CVT_PFdl	Fdl
PMP Circuit Leaf Endpoint	CVT_PMPCkt	PMPCktLeaf
PMP Circuit Root Endpoint	CVT_PMPCktRoot	PMPCktRoot
PMP SPVC Leaf Endpoint	CVT_PMPSpvcLeaf	PMPSpvcLeaf
PMP SPVC Root Endpoint	CVT_PMPSpvcRoot	PMPSpvcRoot
Physical Port	CVT_PPort	Pport
Physical Port Threshold Crossing Alarm	CVT_PPortTca	PportTca
Reference Time Server	CVT_RefTimeServer	RefTimeServer
Service Name	CVT_ServiceName	ServiceName
SMDS Address Prefix	CVT_SmdsAddressPrefix	AddressPrefix
SMDS Alien Group Address	CVT_SmdsAlienGroupAddress	AlienGroupAddress
SMDS Alien Individual Address	CVT_SmdsAlienIndividualAddress	AlienIndividualAddress
SMDS Country Code	CVT_SmdsCountryCode	CountryCode
SMDS Group Screen	CVT_SmdsGroupScreen	GroupScreen

Table 1-2. Object Types Supported by the Provisioning Server (Continued)

Object Name	Enumerated Object Type (API)	Object Type (CLI)
SMDS Individual Screen	CVT_SmdsIndividualScreen	IndividualScreen
SMDS Local Individual Address	CVT_SmdsLocalIndividualAddress	LocalIndividualAddress
SMDS Netwide Group Address	CVT_SmdsNetwideGroupAddress	NetwideGroupAddress
SMDS Switch Group Address	CVT_SmdsSwitchGroupAddress	SwitchGroupAddress
Soft PVC Circuit	CVT_Spvc	Spvc
SVC Address	CVT_SvcAddress	SvcAddress
SVC Config	CVT_SvcConfig	SvcConfig
SVC Close User Group	CVT_SvcCUG	SvcCUG
SVC Close User Group Member	CVT_SvcCUGMbr	SvcCUGMbr
SVC Close User Group Member Rule	CVT_SvcCUGMbrRule	SvcCUGMbrRule
SVC NetworkId	CVT_SvcNetworkId	SvcNetworkId
SVC Node Prefix	CVT_SvcNodePrefix	SvcNodePrefix
SVC Prefix	CVT_SvcPrefix	SvcPrefix
SVC Security Screen	CVT_SvcSecScn	SvcSecScn
SVC Security Screen Action Parameter	CVT_SvcSecScnActParam	SvcSecScnActParam
SVC User Part	CVT_SvcUserPart	SvcUserPart
Switch	CVT_Switch	Switch
Traffic Descriptor	CVT_TrafficDesc	TrafficDesc
Traffic Shaper	CVT_TrafficShaper	TS
Trunk	CVT_Trunk	Trunk
VPCI Table	CVT_VPCITable	VpciTable
Virtual Private Network	CVT_VPN	Vpn

Containment Hierarchy

Figure 1-5 shows the containment hierarchy (the parent-child relation) for building object IDs to name objects in the network.

Keep in mind that network ID is required only when you name an object directly below network in the containment hierarchy. You can omit the network ID for switch and objects lower in the hierarchy.

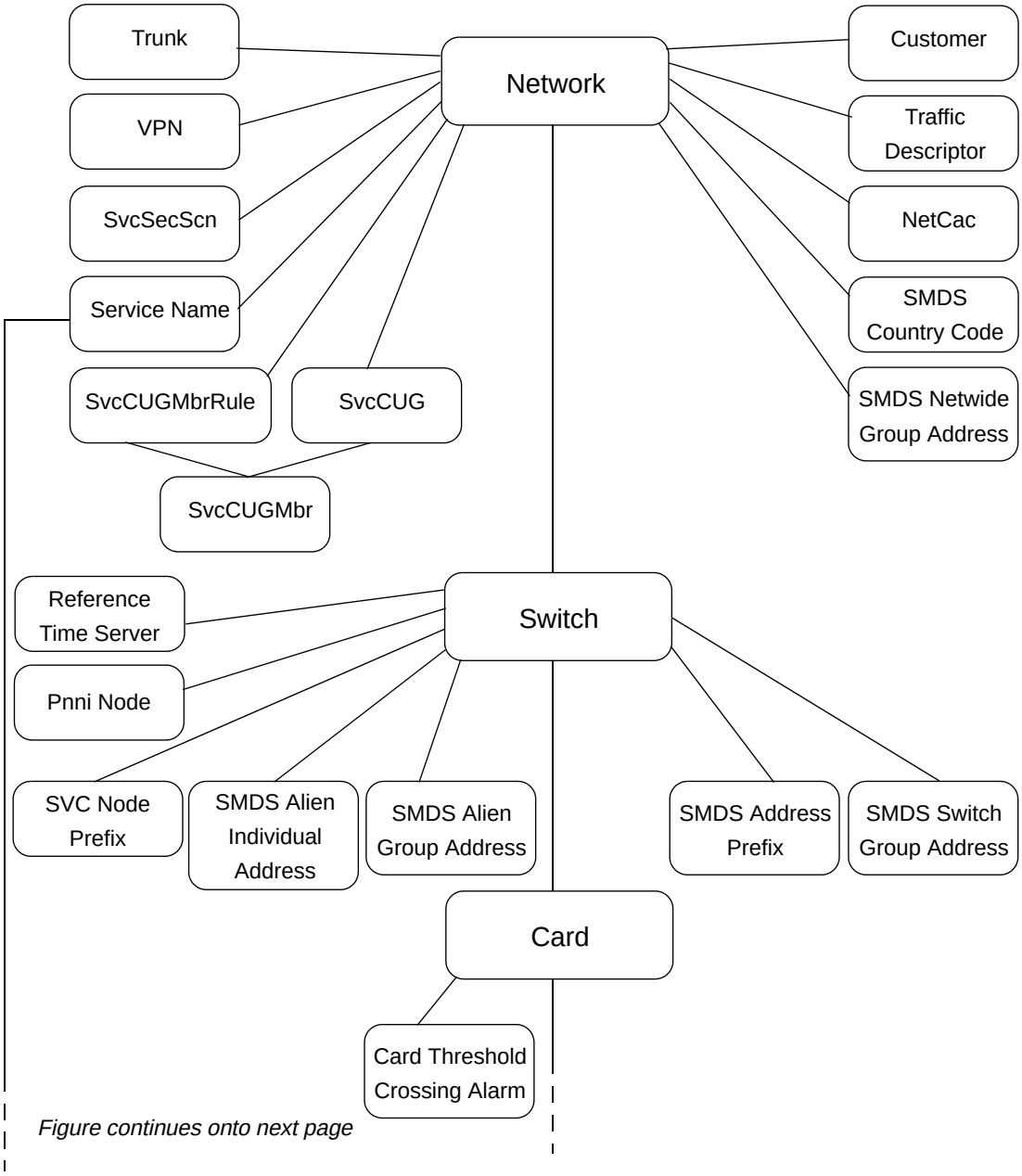


Figure continues onto next page

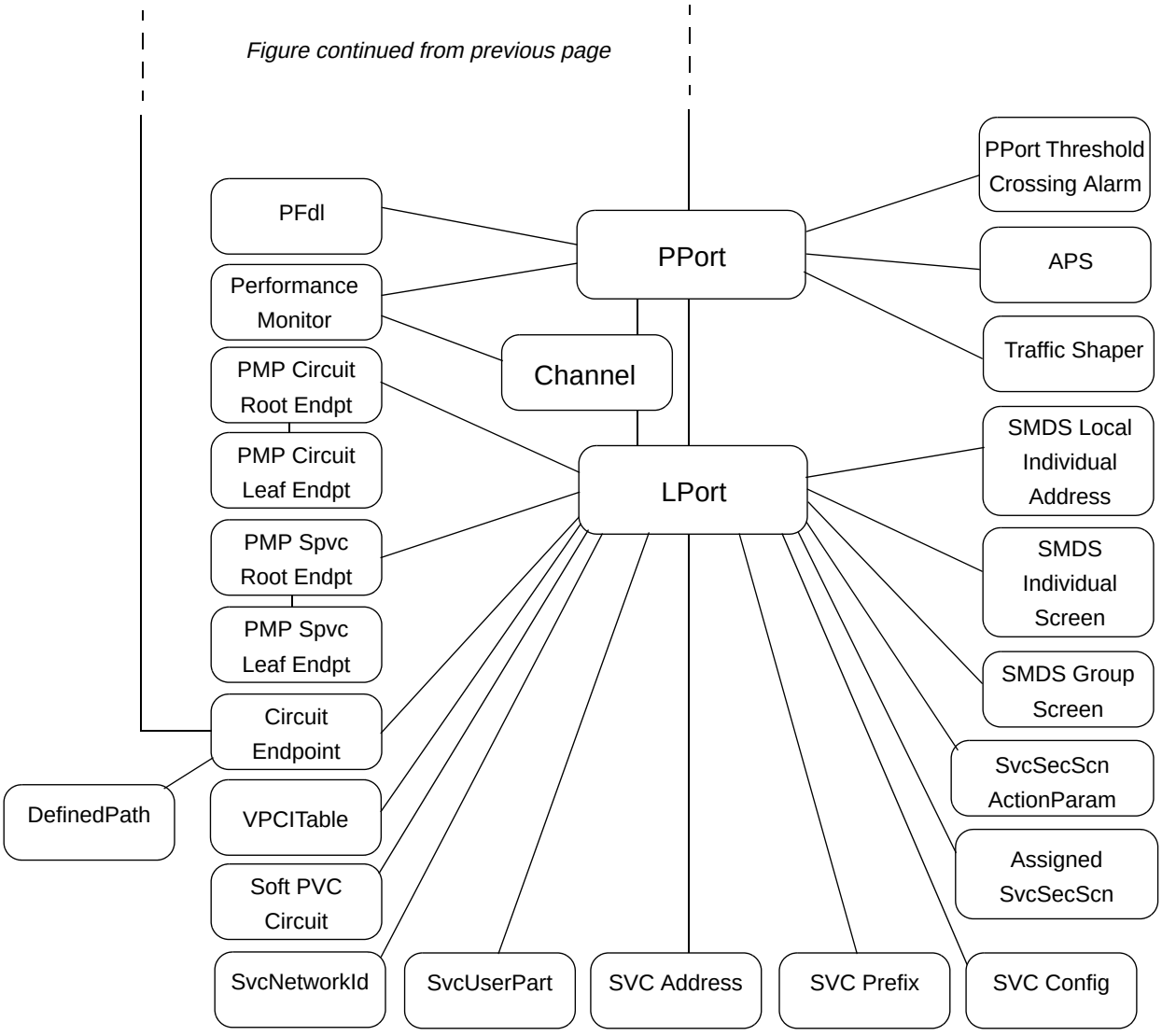


Figure 1-5. Containment Hierarchy for Managed Objects

Naming Conventions for Objects

Table 1-3 lists the rules for naming object type-value identifiers.

Table 1-3. Naming Conventions for Object ID

Object Type	How Identified
Aps CardTca DefinedPath NetCac PerformanceMonitor ChanPerformanceMonitor PFdl PPortTca SMDS group screen SMDS individual screen SvcConfig SvcSecScnActParam	The object is unique to its parent and requires no identifying value. Identify the object by the type name and the parent. For example, an SMDS individual screen is expressed as: *switch.100.101.102.103.card.6.pport.4.lport.2.individualscreen*
AssignedSvcSecScn Customer PnniNode SvcCUG SvcCUGMbr SvcCUGMbrRule SvcSecScn ServiceName TrafficDesc Trunk VPN	By a string name. For PnniNode, the object is identified by a 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 (range 0-104 bits); the number depends upon the Peer Group Level. For example, a PnniNode is expressed as: *switch.100.101.102.103.pnninode.45-01234567890abcdef. . .* where *45* is the number of bits allowed and *01234567890abcdef* is the Peer Group ID. The number of bits specified in the Peer Group ID can be less than Peer Group Level, but not more. For SvcCUGMbr, the object is identified by two names: the CUG name and the member name.

Table 1-3. Naming Conventions for Object ID (Continued)

Object Type	How Identified
Card LPort PMPSpvcLeaf PPort TrafficShaper	By relative number. For example, the fourth PPort on a card is identified as: *switch.100.101.102.103.card.6.pport.4* For PMPSpvcLeaf objects, specify the Root parent. The first PMPSpvcLeaf is identified as: *switch.100.101.102.103.card.6.pport.4.lport.1.PMPSpvcRoot.vpi.5[.vci.43].PMPspvcleaf.1* Use the relative numbering scheme to identify LPorts; do not use the LPort Interface Number displayed in the NavisCore screens. ATM Transport for FR NNI LPorts are identified with VPI and VCI numbers. ATM Virtual UNI LPorts are identified with the VPI start value. MLFRBundleLPorts are created on the card, not a specific PPort. Thus, they are identified as: *switch.100.101.102.103.card.6.lport.1* MLFRMemberLPorts are created on the PPort. Thus, they are identified as: *switch.100.101.102.103.card.6.pport.4.lport.1*
Channel	By a number in the range of 1 - 28. The channel object applies only to channelized cards. For example, a Frame Relay circuit on a channelized DS3 card is identified as: *switch.100.101.102.103.card.6.pport.4.channel.25.lport.1.dlci.55*
Circuit	By the number(s) of its first endpoint. An endpoint can be an LPort or a Service Name; the object ID representation differs accordingly. In the case of LPorts, either endpoint can be a Frame Relay or an ATM endpoint. For Frame Relay endpoints, use the DLCI number. For ATM endpoints, use both the VPI and VCI values. For ATM Network Interworking for Frame Relay NNI endpoints, include the VPI, VCI, and DLCI numbers. For example, a Frame Relay endpoint is expressed as: *switch.100.101.102.103.card.6.pport.4.lport.2.dlci.55* An ATM endpoint is expressed as: *switch.100.101.102.103.card.6.pport.4.lport.2.vpi.8.vci.65* In the case of ServiceName, the endpoint is identified by the network number, the name of the ServiceName binding, and the VPI/VCI pair or DLCI number (depending on endpoint type). For example, a ServiceName endpoint is represented as either: *network.154.188.0.0.ServiceName.xxx.vpi.14.vci.128* *network.154.188.0.0.ServiceName.xxx.dlci.55* where *xxx* is the name of the ServiceName binding.

Table 1-3. Naming Conventions for Object ID (Continued)

Object Type	How Identified
Network	By an IP address with the last 2 bytes set to 0 (Class B addresses) or the last 1 byte set to 0 (Class C addresses). This object type is used to specify a root when you issue a command to list objects contained by a specific parent.
PMPCkt PMPCktRoot PMPSpvcRoot Spvc	By the VPI and VCI values of its endpoint. A PMPSpvcRoot endpoint is expressed as: *switch.100.101.102.103.card.6.pport.4.lport.2.PMPSpvcRoot.vpi.8[.vci.65]*
SMDS address prefix	By an E.164 address string (3 to 6 characters).
SMDS country code	By an E.164 address string (up to 4 characters).
SMDS alien group address SMDS alien individual address SMDS local individual address SMDS netwide group address SMDS switch group address	By an E.164 address string (10 to 16 characters).
SvcAddress SvcNodePrefix SvcPrefix SvcNetworkId SvcUserPart	By a string that conforms to the convention used to specify addresses. For more information, see "SVC Addressing" on page 1-44.
RefTimeServer	By an IP address of the parent switch and IP address of the reference time server. For example, a Reference Time Server is expressed as: *switch.100.101.102.103.RefTimeServer.200.201.202.203*
Switch	By an IP address or by a string name.
VpciTable	By a number in the range of 0 - 65535.

Descriptions of Object Types

The following sections describe the object types, including the kinds of management operations that you can perform on an object and any operating restrictions. The objects types are listed alphabetically.

See the *NavisXtend Provisioning Server Software Release Notice* for this release for information about the object types supported by Provisioning Server on the GX 550 switch.

CVT_Aps

Automatic Protection Switching (APS) protects SONET media from line outages. Currently, APS support is provided for 1-port OC-12c/STM-4 and 4-port OC-3c/STM-1 cards on CBX 500™ switches. When the attribute CVA_PPortRedundancy is set to Aps1+1, the protection port forms a pair with the existing PPort on the card.

You can only modify APS objects; the Provisioning Server does not support adding or deleting them. Depending on whether the PPort is the working PPort or the protection line, you can configure a list of PPort and APS parameters. The attribute CVA_ApsApsCommand is supported for the APS PPort pair for sending external switch requests.

CVT_AssignedSvcSecScn

AssignedSvcSecScn specifies the association between an LPort and an SVC security screen. When you add or delete an object of this type, you are actually adding or removing screens from the parent LPort. This object exists only on ATM UNI/NNI LPorts configured on a CBX 500 switch. A limit of 16 screens can be added to one LPort.

CVT_Card

The NavisCore database automatically populates each switch with cards of type “empty”. To add a card, use the Modify command to change the card’s type from “empty” to a specified type. Specify the appropriate card type using the attribute CVA_CardDefinedType.

The attributes CVA_CardUioDefinedXface, CVA_CardDsx1DefinedXface, and CVA_CardE1DefinedXface provide subtypes for the UIO, Dsx1, and E1 card types, respectively. If you modify a card to one of these card types and do not specify the appropriate subtype, the card defaults to uioXfaceTypeV35, dsx1XfaceTypeRj48, or e1XfaceTypeCoaxPair75Ohm, respectively.

To delete a card, use the Modify command to change the card’s type to “empty”.

If you modify a card to a type that does not match the actual card type, the Provisioning Server does *not* inform you about the type mismatch.

CVT_CardTca

The CardTca object controls card threshold crossing alarm configuration.

CVT_ChanPerformanceMonitor

This object supports the 1-port Channelized DS3-1-0 card. The object identifies the DS1 channel PM Threshold object. It is similar to the existing CVT_PerformanceMonitor object type, but contains a channel identifier.

There are two levels of PM Threshold configuration on the Channelized DS3-1-0 card. PM Threshold can be configured on the PPort level and on the channel level.

The DS3 PM Threshold object has the PPort as its parent. However, similar to the LPort's dual parent identities to accommodate channels, PM threshold objects will have similar OID on this card.

The DS3 PM Threshold object is unique to its parent PPort, and requires no identifying value.

- cvlistcontained switch.1.1.1.1.card.2.pport.3 pm

  This command returns the attributes configurable for the DS3 PM Threshold object on PPort 3.

- cvmodify switch.1.1.1.1.card.2.pport.3.pm -DS3 PM Threshold attributes.

  This command modifies the configurable attributes of the DS3 PM Threshold object on the PPort.

The DS1 PM Threshold object is unique to its parent channel, and requires no identifying value.

- cvlistcontained switch.1.1.1.1.card.2.pport.3.channel.4 pm

  This command returns the attributes configurable for the DS1 PM Threshold object on channel 4.

- cvmodify switch.1.1.1.1.card.2.pport.3.channel.4.pm -DS1 PM Threshold attributes.

  This command modifies the DS1 PM threshold attributes on channel 4.

CVT_Channel

The channel object applies only to channelized cards. Requests that specify a channel and are sent to an object other than the channelized card return an error.

A channel is identified by a number in the range of 1 - 28. For example, a Frame Relay circuit on a channelized DS3 card is represented as:

```
switch.100.101.102.103.card.6.pport.4.channel.25.lport.1.dlci.55
```

Once a channelized card has been configured, NavisCore automatically populates the card with all necessary channels. You can only modify channels; the Provisioning Server does not support adding or deleting channels.

The CVT_Channel object supports the following diagnostic operations: startDiag, getDiag, updateDiag, and stopDiag. These operations enable you to retrieve diagnostic information such as loopbackstatus and errorcount, and change diagnostic parameters such as injecterror and clearcounter. See Chapter 2, “Object Attributes for APS Through LPort,” in *NavisXtend Provisioning Server Object Attributes Definitions* for descriptions of the diagnostic attributes.

CVT_Circuit

A circuit is identified by its first endpoint. An endpoint can be an LPort or a Service Name; the object ID representation differs accordingly.

In the case of LPorts, either endpoint can be a Frame Relay or an ATM endpoint:

- For Frame Relay endpoints, use the DLCI endpoint.
- For ATM endpoints, include both the VPI and VCI values.
- For ATM Network Interworking for Frame Relay NNI endpoints, include the DLCI numbers.
- For ATM Virtual UNI endpoints, use the start VPI value.

Specify a circuit’s second endpoint with the attribute CVA_CircuitEndpoint2.

For example, the Frame Relay endpoint that connects switch 100.101.102.103, card 6, PPort 4, LPort 2, DLCI 55 with ATM endpoint 154.188.162.44, card 3, PPort 5, LPort 11, VPI 8, VCI 65 is represented as either:

```
switch.100.101.102.103.card.6.pport.4.lport.2.dlci.55
switch.154.188.162.44.card.3.pport.5.lport.11.vpi.8.vci.65
```

The Provisioning Server supports VPI values of 0-15 for ATM circuit endpoints.

If an endpoint of a circuit is defined on a channelized DS3 card, the circuit is identified by the channel ID. For example, the Frame Relay endpoint that connects switch 100.101.102.103, card 6, PPort 4, channel 25, LPort 1, DLCI 55 with ATM endpoint 128.129.130.131, card 1, PPort 2, LPort 3, VPI 14, VCI 128 is represented as either:

```
switch.100.101.102.103.card.6.pport.4.channel.25.lport.1.dlci.55
switch.128.129.130.131.card.1.pport.2.lport.3.vpi.14.vci.128
```

When you add a circuit of type VPC, you do not provide the VCI part of the endpoint. For example, the endpoint for a VPC circuit is represented as:

```
switch.100.101.102.103.card.6.pport.4.channel.25.lport.1.vpi.14
```

In this case, the second endpoint must also be an ATM Cell endpoint on a CBX or GX switch, as VPC circuits only support ATM endpoints.

In the case of ServiceName, the endpoint is identified by the network number, the name of the ServiceName binding, and the VPI/VCI pair or DLCI number (depending on endpoint type). Specify the second endpoint with the attribute CVA_CircuitEndpoint2.

For example, a ServiceName endpoint is represented as either:

network.154.188.0.0.ServiceName.xxx.vpi.14.vci.128

network.154.188.0.0.ServiceName.xxx.dlci.55

where *xxx* is the name of the ServiceName binding.

For details on how the Provisioning Server ensures reliability and accuracy of circuit provisioning, see "Circuit Provisioning" on page 1-41.

The CVT_Circuit object supports the following diagnostic operations: startDiag, getDiag, updateDiag, and stopDiag. These operations enable you to retrieve diagnostic information such as loopbackstatus and errorcount, and change diagnostic parameters such as injecterror and clearcounter. See Chapter 2, "Object Attributes for APS Through LPort," in *NavisXtend Provisioning Server Object Attributes Definitions* for descriptions of the diagnostic attributes.

CVT_Customer

Customer objects are associated with VPN objects (Virtual Private Networks). Each customer object contains information that identifies both the customer and the VPN with which the customer is associated. The attribute CVA_CustomerVpnName associates the Customer object with a particular VPN.

The attribute CVA_LPortCustomerName specifies a customer name to which the LPort belongs.

CVT_DefinedPath

The DefinedPath object allows you to specify an exact path to be used when forwarding traffic over a circuit. The DefinedPath consists of a sequence of hops to be used to connect two circuit endpoints. DefinedPath hops are specified as either trunks or nodes. In the case of nodes, any trunk connecting the nodes can be used.

For example, consider a network consisting of four nodes and six trunks. A PVC is already defined (without a corresponding DefinedPath) between Switch.152.150.0.1 and Switch.152.150.0.4. Switch.152.150.0.4 is reachable from Switch.152.150.0.1 in three hops.

By convention, defined paths are specified from the node with the larger IP address to the node with the smaller IP address. The first hop of the path attribute specified that Trunk.T6 should be used to reach the first hop node from Switch.152.150.0.4. The second hop attribute specifies that switch.152.150.0.2 is the second hop and that any trunk connection between switch.152.150.0.2 and switch.152.150.0.3 can be used to reach it.

The third hop attribute is the same as the first. Trunk.T1 should be used from switch.152.150.0.2 to reach the switch.152.150.0.1 and complete the defined path of the circuit.

To specify this path through the CLI, set the pathlist attribute as follows:

```
switch.1.1.1.1.card.4.pport.5.lport.1.dlci.16
-pathlist   network.152.150.0.0.trunk.T6
            switch/152.150.0.2
            network.152.150.0.0.trunk.T1 -endlist
```

CVT_LPort

LPorts have different subtypes: Frame Relay, SMDS, ATM, and Other. When you issue a command, specify only those attributes that are appropriate for the particular LPort's subtype. See the *NavisXtend Provisioning Server Object Attribute Definitions* for the attributes that pertain to each object type and subtype.

For SMDS objects, you can set the CVA_LPortSsiLPort attribute to the object ID of an SMDS SSI DTE LPort. Or, to de-multiplex the LPort, you set the attribute to an object ID of type CVT_Null (use -nullObject in the CLI).

On a channelized DS3 card, an LPort is a child of a channel. Thus, when you issue an Add command, you must specify the channel parent.

For ATM Virtual UNI LPorts on the CBX 500 switch, first create a feeder LPort with ATM UNI type. Since the LPort number of a virtual UNI LPort is generated automatically from the combination of its VPI start number and the Interface Number (which is also generated automatically), you can use the VPI start number to identify the LPort. For example, a Virtual UNI LPort with the start VPI number set to 1 is represented as:

```
switch.128.129.130.131.card.1.pport.2.startvpi.1
```

ATM Network Interworking for Frame Relay NNI LPorts require a different object identifier. This LPort type is identified by VPI/VCI pair.

For example, an ATM Network Interworking for Frame Relay NNI LPort with VPI 1 and VCI 32 is represented as:

```
switch.100.101.102.103.card.6.pport.4.vpi.1.vci.32
```

MLFRBundle LPorts are identified by card, not by PPort. MLFRMember LPorts are identified by parent PPort. The MLFRMember LPorts are bound to a particular MLFRBundle LPort on the same card, which can be used as an endpoint for trunk creation. The bandwidth of the MLFRBundle LPort is the aggregate of its MLFRMember LPorts. A maximum of 16 MLFRMembers can be bound to a MLFRBundle LPort.

The CVT_LPort object supports the following diagnostic operations: startDiag, getDiag, updateDiag, and stopDiag. These operations enable you to retrieve diagnostic information such as loopbackstatus and errorcount, and change diagnostic parameters such as injecterror and clearcounter. See Chapter 2, "Object Attributes for APS Through LPort," in *NavisXtend Provisioning Server Object Attributes Definitions* for descriptions of the diagnostic attributes.

CVT_MLFRBinding

The MultiLink Frame-Relay (MLFR) Binding object is an internal object that the Provisioning Server uses to associate the MLFRBundle and MLFRMember LPorts. It is not possible to create, modify, or delete this object through any user interface. The MLFRBundle and MLFRMember LPorts are objects of type CVT_LPort. CLI commands **cvaddmember** and **cvdeletemember** bind and unbind an MLFRMember LPort to an MLFRBundle LPort. See the descriptions for **cvaddmember** and **cvdeletemember** in Chapter 3, "Using the CLI." Also, for more information about MLFRBundle and MLFRMember LPorts, see the description of the CVT_LPort object in this section.

CVT_NetCac

Network Connection Admission Control (NetCAC) allows you to compute the bandwidth allocation for any virtual circuit. The NetCac object exists under the Network object. This object is supported only for CBX 500 and B-STDX™ 9000 switches. If the attribute CVA_NetCacCacType is set to Cascade, then only Cell Loss Ratio and Cell Delay Variation parameters are configurable. For Customized CAC configuration, you must supply Port Scale Factors and SCR Limit Scale Factors.

In the case of Customized CAC, you must supply the three SCR Limit Scale Factors of Upper Limit, Scale Factor, and Maximum MBS values together. You can supply a maximum of 10 sets. No default values apply and no upper boundary checks are performed for any of these scale factor values.

CVT_Network

Use this object type to specify a root when you issue a command to list objects contained by a specific parent.

CVT_PerformanceMonitor

This object supports the CURRENT (15-minute) and the ONE-DAY threshold parameters for the following cards:

- 8-port T1/E1
- 8-port DS3/E3
- 1-port OC12
- 4-port OC3 cards
- 4-port 24-channel Fractional T1
- 1-port Atm Iwu Oc3
- 1-port Atm CsDs3
- 1-port 28-channel Ds3

Default values are set at the time of card configuration, which you can then modify.

CVT_PFdl

Once a card has been configured, NavisCore automatically populates the card with all necessary Physical Ports. In the case of the ATM-T1 card on the CBX 500 switch, the NavisCore database automatically populates the Extended Super Frame object.

You can only modify the Extended Super Frame object; the Provisioning Server does not support adding or deleting it.

CVT_PMPCkt

A Point-to-MultiPoint (PMP) circuit consists of one endpoint acting as the Root and the other endpoints acting as Leaves. Use this object to add PMP Leaves. A PMP Leaf can be added, modified, and deleted. This object type applies only to CBX 500 and GX switches. Since only ATM endpoints are supported, only VPI and VCI values are supported. To add a leaf using the CLI or the API, you must specify the Root object as one of the attributes. To add a leaf using the MIB, you specify the Root as an index.

CVT_PMPCktRoot

A Point-to-MultiPoint (PMP) circuit consists of one endpoint acting as the Root and the other endpoints acting as Leaves. PMP Root can be added or deleted. This object type applies only to CBX 500 and GX switches. Since only ATM endpoints are supported, only VPI and VCI values are supported. To add a PMP Circuit using the CLI or the API, add the Root and the Leaves separately. Root attributes are Create-Only attributes.

CVT_PMPSpvcLeaf

Use this object type to add the Point-to-MultiPoint (PMP) SPVC Leaf.

For PMPSpvcLeaf objects, specify the Root parent as part of the object ID representation. For example:

```
switch.100.101.102.103.card.6.pport.4.lport.1.PMPSpvcRoot.vpi.5[.vci.43].PMPspvcleaf.1
```

The Root parent specification does not include the vci value if the Root is a permanent virtual circuit (PVC) Spvc.

For the CLI or the API, you no longer need to specify the Root object as one of the attributes.

To add a leaf using the MIB, you specify the Root as an index.

You must specify the correct instance number when you perform an add, get, modify, or delete operation. To retrieve the correct instance number from the database, use the attribute CVA_PMPSpvcRootNextAvailableLeafNo.

CVT_PMPSpvcRoot

This object type is similar to CVT_Spvc, but is used to add a Point-to-MultiPoint (PMP) SPVC root. This object type applies only to CBX 500 and GX switches. You can add, modify, and delete this object type.

When you add the Root, the first leaf is automatically added. To modify the first leaf, use the object type CVT_PMPSpvcLeaf.

CVT_PnniNode

The PnniNode object is an ATM routing and signalling protocol designed for dynamically routing scalable, QoS-enabled, bandwidth adaptive, ATM switched virtual circuits (SVCs). The PnniNode object specifies which Peer Group ID that the node uses when communicating with the PNNI neighbor nodes.

CVT_PPort

Once a card has been configured, NavisCore automatically populates the card with all necessary Physical Ports. You can only modify PPorts; the Provisioning Server does not support adding or deleting PPorts.

The CVT_PPort object supports the following diagnostic operations: startDiag, getDiag, updateDiag, and stopDiag. These operations enable you to retrieve diagnostic information such as loopbackstatus and errorcount, and change diagnostic parameters such as injecterror and clearcounter. See Chapter 2, “Object Attributes for APS Through LPort,” in *NavisXtend Provisioning Server Object Attributes Definitions* for descriptions of the diagnostic attributes.

CVT_PPortTca

The PPortTca object controls PPort threshold crossing alarm configuration.

CVT_RefTimeServer

The Reference Time Server object synchronizes time between an NTP reference time server and a switch.

CVT_ServiceName

ServiceName binding support allows you to identify a primary port (UNI/NNI) with a name so that a circuit can identify its service endpoint by this name instead of by the LPort name. The primary LPort can be a Frame Relay or an ATM UNI/NNI LPort. To associate a backup binding with the primary service name binding, associate a switch port to act as a backup LPort.

When creating a service name binding, specify only the primary LPort. This primary binding cannot be modified.

To set up or modify a backup binding, modify the ServiceName object by specifying the backup LPort. The attribute CVA_ServiceNameActiveBinding indicates the current status of binding. To revert from backup binding to primary binding, set the attribute CVA_ServiceNameActiveBinding to Primary in the modify request.

CVT_SmdsAddressPrefix

An SMDS address prefix is created on a switch to indicate that the switch handles all E.164 addresses that begin with that prefix. You must create an address prefix before you can create an SMDS local individual address that uses that prefix.

You can create an address prefix at any time. You can delete an address prefix only if it is not referenced by any SMDS local individual address. No attributes apply to address prefixes.

CVT_SmdsAlienGroupAddress

Objects of this type are used only as members of a group screen. Use this object type to add a group address to an SMDS group screen, if the group address does not exist on the switch as a switch group address (see "CVT_SmdsSwitchGroupAddress" on page 1-32). In this case, you must first create an SMDS alien group address before you can add the group address to the SMDS group screen. To do so, issue an Add command with no arguments. Then, issue the Add Member command to add the address to the group screen.

CVT_SmdsAlienIndividualAddress

Objects of this type are used only as members of an individual screen. Use this object type to add an address to an SMDS individual screen, if the address does not exist on the switch as a local individual address (see "CVT_SmdsLocalIndividualAddress" on page 1-31). In this case, you must first create an SMDS alien individual address before you can add the address to the SMDS individual screen. To do so, issue an Add command with no arguments. Then, issue the Add Member command to add the address to the individual screen.

The Add command can fail or cause problems in the switch if the alien individual address uses a prefix that is assigned to the switch. By definition, an alien individual address should use a prefix not currently defined anywhere in the network.

CVT_SmdsCountryCode

Specify this object on the network level using up to 3 digits in E.164 format. To use a country code in an SMDS local individual address, include a dash (-) between the country code and the prefix (for example: 1-9789521111). If you omit the country code, the server uses the default country code specified in the environment variable CV_DFLT_SMDS_CC (for details, see "Setting Environment Variables" on page 2-11).

CVT_SmdsGroupScreen

There is only one group screen per SMDS LPort, and you must explicitly create it. Create the group screen before you add addresses to it. Once you create the group screen, you can add switch group addresses or alien group addresses as members. When you create a netwide group address, a switch group address is created automatically.

You can use an SMDS screen to either allow or disallow specific addresses for an LPort. Use the attribute CVA_GroupScreenOperation to do so.

CVT_SmdsIndividualScreen

There is only one individual screen per SMDS LPort, and you must explicitly create it. Create an individual screen before you add addresses to it. Once you create the individual screen, you can add addresses or alien addresses as members.

An SMDS screen can be used to allow or disallow specific addresses for the LPort. Use the CVA_IndividualScreenOperation attribute to do so.

CVT_SmdsLocalIndividualAddress

In NavisCore, an SMDS local individual address is known as an individual address. You create this object on an LPort to associate that address with the LPort. The address must use a prefix that has already been created on that switch. You can also use an existing country code specified in the network (see "CVT_SmdsCountryCode" on page 1-30) as part of the local individual address. You cannot delete an LPort until you have deleted all its local individual addresses.

CVT_SmdsNetwideGroupAddress

A netwide group address is a collection of SMDS switch group address objects. You create and manage group addresses through these objects and not through SMDS switch group address objects. You must create a netwide group address in the appropriate subnetwork before you can add members (individual addresses) to it. You must also use the Delete Member command to remove all of the netwide group address members before you can delete the netwide group address itself.

CVT_SmdsSSIIndividualAddress

This object is obsolete, but is maintained for compatibility with previous versions of the Provisioning Server. The SniDxi LPort does not have to subscribe to an address from an SSI LPort's address pool even if the LPort is multiplexed to an SSI LPort. You can perform any SMDS configuration without creating the SSI individual address pool.

CVT_SmdsSwitchGroupAddress

An SMDS switch group address does not appear in NavisCore. This object represents a group address that is local to a switch. A switch group address with a given address should exist on a switch only if the equivalent netwide group address has members on that switch; it lists the addresses of that switch that are members of the equivalent netwide group address. The only time you should need to reference a group address directly is to add one to a group screen.

You should not have to create or delete objects of this type; they are created and deleted automatically during the management of SMDS netwide group addresses. However, since you cannot delete a netwide group address if it contains any group address members, in rare cases, you may need to delete a group address manually. You cannot delete a group address until it no longer contains individual address members.

CVT_Spvc

Soft PVC circuits are identified by an endpoint at one end and the SVC Address at the other end. The other endpoint may not necessarily exist in the same network. You can add, modify, and delete this object. This object type applies only to CBX 500 and GX switches. Since only ATM endpoints are supported, only VPI and VCI values are supported. Specify the SVC Address using attributes.

CVT_SvcAddress

SvcAddress provides an interface to set up full ATM addresses (20 octets) on an LPort. This address is associated with the following LPort types located on CBX 500 switches:

- atmUniDte
- atmUniDce
- atmNni

Frame Relay addresses are associated with the following LPort types located on B-STDX 8000 and 9000 switches:

- frUniDte
- frUniDce

There can be zero or more SvcAddresses configured per LPort.

An ATM Address can be one of the following format types:

- E.164native

- AESA addresses:
  - E.164AESA
  - DCCAESA
  - ICDAESA
  - CustomAESA
  - DCCAnycastAESA
  - ICDAnycastAESA
  - E.164AnycastAESA

For AESA addresses, if the address prefix is 39 characters, the Provisioning Server appends a zero to the address to make it 20 octets.

For Frame Relay SVCs, E.164native and X.121 are valid formats.

For information on the convention used to specify SVC addresses, see "SVC Addressing" on page 1-44.

CVT_SvcConfig

Use this object to configure an LPort for switched virtual circuits.

Only one SvcConfig object is associated with an LPort. An LPort is created with a default SvcConfig; you can only modify this object to change an SVC configuration. The SvcConfig is deleted when its LPort is deleted.

CVT_SvcCUG

Use this object to configure SVC Closed User Groups. You can create this object under the Network object. Each SVC Closed User Group can contain up to 128 members. You cannot perform a database-only modification on this object, since the modification has to be distributed throughout the network.

CVT_SvcCUGMbr

Use this object to create the association of an SvcCUG object and an SvcCUGMbrRule. Deleting this type of object disassociates the specified SvcCUG with the SvcCUGMbrRule. Adding, modifying, and deleting an SvcCUGMbr object requires network distribution; you cannot perform a database-only modification on this object.

CVT_SvcCUGMbrRule

During creation of this object, a distribution list is created by matching its rule to ATM SVC prefixes, addresses, and user parts configured on nodes in the network. Adding, modifying, and deleting an SvcCUGMbrRule object requires network distribution; you cannot perform a database-only modification on this object.

CVT_SvcNetworkId

This object provides an interface to add SVC Network IDs to an LPort. A NetworkID object can be configured only on the following types of LPorts:

- atmUniDte
- atmUniDce
- atmNni
- frUniDte
- frUniDce

Valid address format types for this object are:

- Carrier ID Code (CIC)

  The address prefix can be from 1 to 8 characters long. The nBits value is calculated as the string length of the address prefix * 8.

- Data Network ID Code (DNIC)

  Only the frUniDte and frUniDce LPorts support this format. The address prefix is 4 characters long. Thus, the calculated nBits value is always 32.

Setting the nBits value is optional. If you set an incorrect value, the Provisioning Server returns an error.

CVT_SvcNodePrefix

This object provides an interface to add node prefixes on a switch. The switch imposes no constraints on the node prefixes except to enforce the maximum length (which is the same as a full address length — 20 octets). 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.

The valid address format types for ATM SVCs are the same as described for the CVT_SvcAddress object (see "CVT_SvcAddress" on page 1-32). For Frame Relay SVCs, E.164native and X.121 are valid formats. There can be zero or more SvcNodePrefixes configured per switch.

For information on the convention used to specify SVC addresses, see "SVC Addressing" on page 1-44.

CVT_SvcPrefix

This object provides an interface to set up prefixes on an LPort. Prefix is one of the three classes of static addressing used for switched virtual circuits. The other classes are full ATM Address and Node Prefix.

Prefix is associated with the following LPort types located on CBX 500 switches:

- atmUniDte
- atmUniDce
- atmNni

Frame Relay SVC prefixes are associated with the following LPort types located on B-STDX 8000 and 9000 switches:

- frUniDte
- frUniDce

There can be zero or more SvcPrefixes configured per LPort.

An ATM Address prefix can be of the following format types:

- E.164native
- AESA addresses:
  - E.164AESA
  - DCCAESA
  - ICDAESA
  - CustomAESA
  - DCCAnycastAESA
  - ICDAnycastAESA
  - E.164AnycastAESA
- DefaultRoute: Use this format to configure the port with an ATM address length of zero bits. This enables this port to receive messages that could not be routed to other ports because of ATM address mismatch.

For Frame Relay SVCs, E.164native, DefaultRoute, and X.121 are valid formats.

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.

For information on the convention used to specify SVC addresses, see "SVC Addressing" on page 1-44.

CVT_SvcSecScn

This object exists on the network level. Its name is used by the AssignedSvcSecScn object to apply screening to an LPort. Modify and delete operations require network wide distribution; you cannot perform database-only modification on this object.

CVT_SvcSecScnActParam

This object exists under the LPort object and is populated/deleted automatically with LPort creation. One instance exists for each ATM UNI LPort on a CBX 500 switch.

CVT_SvcUserPart

Use this object to set up the user part on a DTE LPort on a CBX 500 switch. It is used for dynamic address registration at a UNI. The user part address length is 7 octets, of which End System Identifier (ESI) represents 6 octets and the selector represents 1 octet. The user part represents a partial SVC address associated with ATM DTE LPorts on the node. The rest of the address is the network prefix, which is supplied by the network side of the UNI. To obtain an ATM address for a terminal on the user side of a Private UNI, append values for the user part to network prefix(es) for that UNI.

You can create UserParts (zero or more) only on ATM DTE LPorts.

For information on the convention used to specify SVC addresses, see "SVC Addressing" on page 1-44.

CVT_Switch

The Provisioning Server does not support adding or deleting switches; you must use NavisCore to do so. For an existing switch, you can read or modify any switch attribute except for the CVA_SwitchName attribute, which is Read-Only.

CVT_TrafficDesc

The Provisioning Server provides support for maintaining a pool of ATM traffic descriptors. The traffic descriptors are required for setting up forward and backward traffic descriptors for Soft PVCs. Each traffic descriptor is identified by a name. An ID is automatically associated with each name.

Depending on the Quality of Service (QoS) class you select and the Type of Service associated with it, you need to provide the PCR, SCR, and MBS values. Do so using the attributes CVA_TrafficDescParam1, CVA_TrafficDescParam2, and CVA_TrafficDescParam3. Only add and delete operations are supported for ATM traffic descriptors.

CVT_TrafficShaper

Traffic shaper objects are located under PPort in the containment hierarchy. Only PPorts on the following cards can have traffic shaper objects:

- 1-port ATM IWU OC3 card
- 1-port ATM CS/DS3 card
- 1-port ATM CS/E3 card

A traffic shaper object is not an independent object. It represents a group of traffic shaper attributes under a particular PPort type. All the traffic shaper attributes are essentially the attributes for the belonging PPort. The traffic shaper attributes are treated as a separate object to provide a clear user interface for the attributes.

Once a PPort is created through NavisCore, NavisCore automatically populates the PPort with traffic shaper attributes. You can only modify traffic shaper attributes; the Provisioning Server does not support creating them.

When you issue a ListContained command on a traffic shaper object, the only valid parent object type is PPort.

CVT_Trunk

A trunk object allows two switches to pass data to each other. A Direct Line trunk connects two switches directly. An OPTimum trunk connects two switches via a public data network.

You can use the Provisioning Server to add a trunk. However, if you want NavisCore map to show the trunk connection between the switches, you must use NavisCore to add the connection to the map.

You specify a trunk by its name:

```
Trunk.NYLA
```

If the trunk string name contains one of the following special characters: ‘ * ! { } ( ) $ & ; \ | " or blank character, you must enclose at least the special character with /” characters:

```
TrunkName.Boston/”&/”NY
```

If the trunk string name contains a period, you must enclose the entire string with /” characters:

```
TrunkName./”Boston.NY/”
```

You do not need to enclose the following special characters: + = - _ @ # ^ % , : [ ] / ~

CVT_VPCITable

The VPCI table object maps a particular PSA VPCI to a PSC VPI when proxy signaling is in use. When proxy signaling is not in use, the VPCI table augments the currently-available VPCI to VCI mapping mechanism by allowing you to customize the mapping. For the VPCI table to work properly, both the Proxy Signaling Agent and all Proxy Signaling Clients must use the same type of VPCI to VPI mapping.

CVT_VPN

Creation and deletion of VPN objects occur at the network level. Use the attribute CVA_LPortVPNName to specify the VPN to which an LPort belongs. Setting CVA_LPortVPNName to Public makes that LPort a normal public LPort. Similarly, use the attribute CVA_CircuitVPNName to specify the VPN to which a circuit belongs. Both end points of a circuit may not belong to different VPNs.

Valid Object Types for Operational Functions

Table 1-4 lists the object types you can use when you issue the operational functions and commands of the API and CLI.

Table 1-4. Valid Object Types for Operational Functions

Object Type	Add Object	Add Member	Delete Object	Delete Member	Get	List All	List	Modify	Diag	Get Oper Info
APS					✓		✓	✓		
AssignedSvcSecScn	✓		✓		✓	✓	✓	✓		
Card					✓	✓	✓	✓		
CardTca					✓			✓		
Channel					✓	✓	✓	✓	✓	
ChanPerformanceMonitor					✓		✓	✓		
Circuit	✓		✓		✓		✓	✓	✓	✓
Customer	✓		✓		✓	✓	✓	✓		
DefinedPath					✓			✓		
LPort	✓	✓[a]	✓	✓[b]	✓	✓	✓	✓	✓	
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 1-4. Valid Object Types for Operational Functions (Continued)

Object Type	Add Object	Add Member	Delete Object	Delete Member	Get	List All	List	Modify	Diag	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	✓		✓		✓	✓	✓	✓		

[a] For binding an MLFRMember LPort to an MLFRBundle LPort

[b] For unbinding an MLFRMember LPort to an MLFRBundle LPort

Object Attributes

For each of the managed objects supported by the Provisioning Server, there are arguments (attributes) that can be read or configured through the API or CLI. An argument list is represented as follows:

In:	Argument List represented as:
C	An opaque pointer that is manipulated using utility functions.
C++	A class that is manipulated using member functions.
CLI	String representations of the attributes.

The *NavisXtend Provisioning Server Object Attribute Definitions* lists the various object types supported by the Provisioning Server and their associated attributes. See that guide to determine which attributes apply to which object types.

Circuit Provisioning

The Provisioning Server uses a retry control to ensure reliability and accuracy of circuit provisioning. The retry control is used for provisioning circuits on B-STDX 8000, B-STDX 9000, CBX 500, and GX 550 switches. This control specifies retry behavior in the event of a failed attempt to add, delete, or modify a circuit.

By default, when the Provisioning Server receives a request to add, delete, or modify a circuit, the server obtains card status for both circuit endpoints:

- If both cards are up, the Provisioning Server performs the add, delete, or modify request as normal.
- If either card is down or is not reachable (for example, because of an SNMP timeout), the server retries the request for card status as many times as specified by the retry control (in the range of 0 to 5 times):
  - If the card becomes reachable and is up, the Provisioning Server performs the circuit provisioning request.
  - Once all retries have been issued, if the card is still not reachable or is still down, the provisioning request is not performed.

The control that specifies retry behavior of circuit provisioning requests takes the following forms:

In:	Retry Control represented by:
API	C functions: **CvSetNumRetries** and **CvGetNumRetries** C++ functions: **setNumRetries** and **getNumRetries** (See the *NavisXtend Provisioning Server Programmer's Reference*.)
CLI	Environment variable CV_CLI_NUM_RETRIES (See "Specifying Retry Behavior" on page 2-13.)
MIB	NumRetries attribute (See "NumRetries Attribute" on page 4-9.)

This control prevents circuits from being partially provisioned and the database from becoming out of sync with the switch. However, it can increase the time it takes to provision a circuit, depending on how many card status checks occur.

Keep in mind that this control affects the retry behavior of circuit provisioning requests only. Other retry controls specified in cascadeview.cfg (CV_SNMP_MAX_RETRIES, CV_SNMP_RETRY_INTERVAL, and CV_SNMP_REQUEST_TIMEOUT) also apply to each request. Remember to consider these other retry controls when specifying retry behavior of a circuit request.

Related Error Reporting

Whenever the Provisioning Server determines that a card is down or is not reachable, and thus a circuit provisioning request cannot be performed, the server returns an error indicating the reason for failure and specifying which card is affected.

If the card becomes reachable and is up, the Provisioning Server performs the circuit provisioning request. During the provisioning process, if either endpoint returns an error (such as SnmpTimedOut, SnmpBadValue, or SnmpNoSuchName), the server returns an error indicating the reason for failure and specifying which endpoint is affected (including switch name, LPort name, slot ID, PPort ID, and DLCI number).

MIB clients can query the Command Error Table to obtain this error information.

Environment Variable to Override Status Check

If you do not want the Provisioning Server to obtain card status prior to provisioning circuits, you can override the server's default behavior. To do so, set the server environment variable CV_CARD_STATS to DISABLE.

For details, see "Disabling Card Status Checking" on page 2-20.

Bit Mask

Table 1-5 describes Provisioning Server bit mask configuration.

Table 1-5. Bit Mask Configuration

Card	PPortChDs3ChannelsInUse[a] or ChannelsInUse?[b]	PPort Allocated Channel Count/[c] Allocated Channels[d]	Channel Allocated Channel Count[c]/ Allocated Channels[d]	LPort Fractional DS0s[e]
4Ports24ChannelsFractT1	ChannelsInUse	✓		✓
4Ports30ChannelsFractE1	ChannelsInUse	✓		✓
4PortsUnchannelizedT1	ChannelsInUse	✓		
4PortsUnchannelizedE1	ChannelsInUse	✓		
4Ports24ChannelsDSX	ChannelsInUse	✓		✓
10PortsDSX1	CannelsInUse	✓		
1PortChannelizedDs3	ChDS3ChannelsInUse	✓	✓	
1PortChannelizedDS310	ChDS3ChannelsInUse		✓	✓
12PortsUnchannelizedE1	ChannelsInUse	✓		

[a] Indicates which of the 28 DS1s has logical port allocations. For example, if channel 28 contains a logical port, then the value of ChDS3ChannelsInUse is 1342177278, which is the equivalent of 10000000000000000000000000000 in binary. If channel 11 contains a logical port, the value of ChDS3ChannelsInUse is 1024, or 10000000000 in binary. The bit set is the 11th bit in the bit mask, corresponding to the 11th channel.

[b] Indicates which of the DS0s/TS0s have been assigned to logical ports. For example, if channel 28 is assigned to a logical port, then the value of ChannelsInUse is 1342177278, which is the equivalent of 10000000000000000000000000000 in binary. If channel 11 is assigned to a logical port, then the value of ChannelsInUse is 1024, or 10000000000 in binary. The bit set is the 11th bit in the bit mask, corresponding to the 11th channel.

[c] Shows the number of DS0s that are allocated for this DS1 channel.

[d] Shows which DS0s are allocated. For example, if DS0s 1, 2, and 8 are allocated on one DS1, then the value for AllocatedChannelCount is 3, because 3 DS0s are allocated. AllocatedChannels has the value of 131, which is the equivalent of 10000011. The first bit is DS0 8, and the last 2 bits are for DS0s 1 and 2. The 8th, 2nd, and 1st bits settings correspond to the 8th, 2nd, and 1st DS0s.

[e] Shows which DS0s are assigned to this logical port. For example, DS0s 1, 2, and 8 are allocated for a particular DS1(see the description for ChannelsInUse), so they are available for assignment to a logical port. However, if only DS0s 1 and 8 are assigned to a logical port, the value of FractionalDS0s is 129, or the equivalent of 10000001. The first bit is for DS0 8, and the last bit is for DS0 1. The 8th and 1st bits settings correspond to the 8th and 1st DS0s.

SVC Addressing

SVC addresses are represented as strings, using the following convention:

<Address-Format-Type-ID>-<Address-Prefix>-<nBits>

Address-Format-Type-ID is the number that represents the format of the SVC address. Valid values are as follows:

1 — E.164native

2 — DCCAESA

3 — ICDAESA

4 — E.164AESA

5 — CustomAESA

6 — DefaultRoute

7 — UserPart

8 — Carrier ID Code (CIC)

9 — Data Network ID Code (DNIC)

10 — X.121

11 — DCCAnycastAESA

12 — ICDAnycastAESA

13 — E.164AnycastAESA

Address-Prefix is the complete 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. For CIC addresses, the address prefix can be from 1 to 8 characters long. For DNIC addresses, the address prefix is 4 characters long.

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 1-6.

Table 1-6. Calculated nBits Values

Address Format Type	Address-Format-Type-ID	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)

Table 1-6. Calculated nBits Values (Continued)

Address Format Type	Address-Format-Type-ID	Calculated nBits Value
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.121	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)

String Conversion

In the following cases, the Provisioning Server performs an address string conversion:

- 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.

Each of the address formats is described in the following sections.

E.164native

Specify an E.164native address as a numeric string of 1 - 15 characters.

For example:

```
1-12345
```

where 1 specifies the address format type E.164native, and 12345 represents the address prefix. Since no nBits value is specified, the Provisioning Server calculates a value and appends it to the address. The string is converted to:

```
1-12345-40
```

AESA Addresses

Specify an AESA address as a hexadecimal string. The first two characters of the address prefix represent the AFI value. The address prefix must be in the range of 2 - 40 characters. The number of bits must be in the following range:

(integral-part-of((string length of address - 1)/2)*8)< nBits ≤
(integral-part-of((string length of address + 1)/2))*8

The minimum value for nBits is 8.

In the case of CustomAESA format, the AFI value can be any two hexadecimal characters.

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.

Standard AFI values are:

39 — DCCAESA

45 — E.164AESA

47 — ICDAESA

BD — DCCAnycastAESA

C5 — ICDAnycastAESA

C3 — E.164AnycastAESA

Example 1

```
2-39
```

where 2 specifies the address format type DCCAESA, and 39 represents the address prefix (consisting of the AFI value only).

Since no nBits value is specified, the Provisioning Server calculates a value and appends it to the address. The string is converted to:

```
2-39-8
```

Example 2

```
2-391234567890abcde
```

where 2 specifies the address format type DCCAESA, and 39123456789abcde represents the address prefix. Since no nBits value is specified, the Provisioning Server calculates a value and appends it to the address. And, because the address prefix is an odd number of characters, the Provisioning Server appends a zero to the address prefix. The string is converted to:

```
2-391234567890abcde0-72
```

Example 3

```
2-391234567890abcde-70
```

where 2 specifies the address format type DCCAESA, 391234567890abcde represents the address prefix, and 70 represents the nBits value. In this example, the valid range for nBits is:

(integral-part-of((17 - 1)/2)*8) < nBits ≤ (integral-part-of((17 + 1)/2))*8

64 < nBits ≤ 72

Because the address prefix is an odd number of characters, the Provisioning Server appends a zero to the address prefix. The string is converted to:

```
2-391234567890abcde0-70
```

Example 4

```
5-ff1234-23
```

where 5 specifies the address format type CustomAESA, ff1234 represents the address prefix (with AFI value ff), and 23 represents the nBits value. In this example, the valid range for nBits is:

(integral-part-of((6 - 1)/2)*8) < nBits ≤ (integral-part-of((6 + 1)/2))*8

16 < nBits ≤ 24

Example 5

```
11-BD1234567890abcde-70
```

where 11 specifies the address format type DCCAnycastAESA, BD1234567890abcde represents the address prefix (with AFI value BD), and 70 represents the nBits value. In this example, the valid range for nBits is:

(integral-part-of((17 - 1)/2)*8) < nBits £ (integral-part-of((17 + 1)/2))*8

64 < nBits ≤ 72

DefaultRoute

Specify a Default Route address as the address prefix 00 and 0 bits. For example:

```
6-00-0
```

where 6 specifies the address format type DefaultRoute, 00 represents the address prefix, and 0 represents the number of bits.

UserPart

Specify a User Part address as a hexadecimal string of 14 characters.

The value for nBits is 56.

For example:

```
7-1234567890abcd
```

where 7 specifies the address format type UserPart, and 1234567890abcd represents the address prefix. Since no nBits value is specified, the Provisioning Server calculates a value and appends it to the address:

```
7-1234567890abcd-56
```

X.121

Specify an X.121 address as a numeric string of 1 - 15 characters.

For example:

```
10-12345
```

where 10 specifies the address format type X.121, and 123456 represents the address prefix. Since no nBits value is specified, the Provisioning Server calculates a value and appends it to the address:

```
10-12345-40
```

Class B Addressing

The Provisioning Server treats all IP Addresses as Class B addresses. The server interprets all addresses as follows:

- First 2 bytes of an IP address are used as the network ID.
- Second 2 bytes of an IP address are used as the switch ID.

The Provisioning Server uses the third byte of the address as the Class B subnet number. For example, the server interprets the following network address:

128.100.111.0

as network address 128.100.0.0 and subnet number 111.

General API Usage

This section provides the basic procedures for performing operations with the Provisioning Server API.

The API operates by establishing a session to the Provisioning Server. The session maintains internal context between the client and the server: it opens a socket and an associated file descriptor. More than one session can be open at a time. You should close a session before the program terminates.

C Program

To use most of the C commands, a client program must follow the following general steps:

1. Issue **CvConnect** to establish a session with the Provisioning Server.
2. Identify the object to be operated on. To do so, issue **CvCreate***ObjectType***Id** to fill in the CvObjectId structure.
3. Identify necessary arguments (object attributes) and set values, if needed. To do so, 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.
4. Issue an operational function on the object.
5. Use **select** loop processing functions to receive and process the response.

6. Once the request has been processed, issue **CvArgsFree** to free the memory used by the argument list.
7. When the application exits, issue **CvClose** to terminate the session with the Provisioning Server.

C++ Program

To use most of the C++ commands, a client program must follow these general steps:

1. Establish a session with the Provisioning Server. To do so, create a **CvClient** class and issue the **CvClient::open** function to pass **CvClient** arguments that provide session context.
2. Identify the object to be operated on. To do so, create and set values in a **CvClient::ObjectId** object.
3. Identify necessary arguments (object attributes) and set values, if needed. To do so, create and set values in a **CvClient::Args** object.
4. Issue an operational function on the object.
5. Use **select** loop processing functions to receive and process the response.
6. When the application exits, terminate the session with the Provisioning Server. To do so, either:
   - 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.

2

Installation and Administration

This chapter describes hardware and software requirements and how to perform a new installation of the Provisioning Server and the Application Toolkit. It also describes the steps required for the following administrative tasks:

- Setting environment variables to configure the various components of the Provisioning Server system
- Stopping and restarting the Provisioning Server and the CLI
- Troubleshooting problems with the Provisioning Server
- Developing a provisioning application

Prerequisites

This section describes the hardware and software required by the NavisXtend Provisioning Server.

This product requires one or more workstations: one is designated as the Provisioning Server and the others are designated as the Provisioning clients. The Provisioning Server and clients can reside on the same workstation.

Provisioning Server Requirements

This section lists the minimum requirements for the Provisioning Server.

Server Hardware

To run the Provisioning Server, you must have an UltraSparc2 or equivalent with the following minimum hardware:

- 70 MB disk space
- CD-ROM drive

The CPU and RAM requirements for the Provisioning Server depend on the number of clients that will issue requests to the server. Typically, CPU or RAM requirements are less than those required for a NavisCore installation. For details, see the *NavisCore Network Management Station Installation Guide*.

Server Software

The Provisioning Server requires NavisCore. A minimum of NavisCore Release 04.01.01.00 must be installed on a network workstation to at least the point where the Sybase database is installed and configured. The Provisioning Server can be installed on either the same host as Naviscore or on a different host, as long as the Provisioning Server can reach the Sybase database and the switches.

Before you install the Provisioning Server software, verify that the following software programs are installed (for instructions, see the *NavisCore Network Management Station Installation Guide*):

Sun Microsystems SunSoft™ Solaris® 2.5.1 cluster patch (2.5.1_Recommended.tar.Z) OR Solaris 2.6 plus Solaris 2.6 cluster patch (2.6_Recommended.tar.Z)

SYBASE Open Server™, Release 11 — The relational database software program for storing database information and providing backup and recovery of database files. This software must be installed on the network and the Provisioning Server must be a client of that database.

NavisCore, Version 04.01.01.00 — The Provisioning Server installation utilizes the NavisCore-specific installation procedures. Thus, at a minimum, this software must be installed to the point where the SYBASE database is installed and configured.

Ascend recommneds that this release of the Provisioning Server be used with the following software versions:
- Solaris 2.6
- SYBASE 11.0.3.3

Provisioning Client Requirements

This section lists the minimum requirements for the Provisioning client.

Client Hardware

The minimum hardware required to run a NavisXtend Provisioning client is any Sun SPARCstation or equivalent. The Provisioning Server Application Toolkit requires approximately 15 MB of disk space.

Client Software

Before you install the Application Toolkit software, verify that the following software programs are already installed on the workstation:

Sun Microsystems SunSoft Solaris 2.5.1 plus Solaris 2.5.1 cluster patch (2.5.1_Recommended.tar.Z), OR Solaris 2.6 plus Solaris 2.6 cluster patch (2.6_Recommended.tar.Z)

SPARCWorks™ compiler version 4.0, 4.1, or 4.2 — The compiler required to compile a C or C++ program. This software is required only if you plan to write a C or C++ program; it is *not* required if you plan to use the CLI only.

SMIv2 MIB compiler — An SMIv2-compliant compiler required to compile the Provisioning Server MIB. This software is required only if you plan to use the Provisioning Server MIB; it is *not* required if you plan to write a C or C++ program or use the CLI only.

Ascend recommneds that this release of the Provisioning Server be used with the following software versions:
- Solaris 2.6
- SPARCWorks compiler 4.2

Switch Requirements

For minimum switch software revisions required by the Provisioning Server, see the *Software Release Notice for NavisCore Release 04.01.01.00.*

Network Requirements

The Provisioning Server must be configured in a TCP/IP network and must have access to the Ascend switches.

The Provisioning client must have access to the Provisioning Server over a local-area or wide-area network.

Installation Instructions

This section describes how to install the Provisioning Server and the Provisioning Server Application Toolkit.

For instructions on upgrading your Provisioning Server software from a previous version, see the *Software Release Notice for NavisXtend Provisioning Server.*

For any updates to this installation procedure, see the *Software Release Notice for NavisXtend Provisioning Server*.

Installing the Provisioning Software in a Single-System Configuration

This section describes how to install the Provisioning Server software on the same workstation as the SYBASE database and NavisCore. The procedure requires that you already have NavisCore installed and that the database contains information on the switches that you wish to access through the server.

The installation script prompts you for the Sybase DSQUERY name, NavisCore Sybase database name, and Sybase administrator name and password. Determine these values before you begin the installation.

To install the Provisioning Server and the Application Toolkit, perform the following steps:

1. Log on as the root user and enter the root password.
2. Insert the Provisioning Server media into the media drive.
3. Enter the following command to start the installation script:

   `[media device]/install_NAVISeps`

   where [media device] is the name of the machine media device (for example, /cdrom/cdrom0).

   The pkgadd menu appears, listing the NAVISeps package.

```
The following packages are available:
   1 NAVISeps NavisXtend Provisioning Server
             (sparc) [version #]

Select package(s) you wish to process (or 'all' to process all
packages). (default: all) [?,??,q]:
```

If the installation utility detects another instance of the Provisioning Server on your system, it prompts whether you want to remove that instance. If you answer **yes**, it removes the instance and performs a fresh install. If you answer **no**, the installation script quits.

4. Select the NavisXtend Provisioning Server package.

   The installation utility prompts you to select the components you want to install on the machine.

   ```
   c) Install NAVISXtend Client
   s) Install NAVISXtend Server
   b) Install both NAVISXtend Client and Server
   q) Exit this install

   Selection:
   ```

5. Specify which components you want to install on the machine. You can install the Provisioning client (which includes the CLI, the Provisioning Server Application Toolkit client libraries, and the client include files), the Provisioning Server, or both. The Provisioning Server and client occupy approximately 50 MBytes of disk space.

   Keep in mind that if you choose to install only the Provisioning Server on a machine, the CLI binaries and associated links will not be present on that server machine.

   If you choose to install only the client on a machine, skip to Step 19.

6. When prompted, specify whether NavisCore is installed on the machine.

   If you answer yes, the installation utility prompts you to enter the base directory where NavisCore is installed.

7. Enter the path to the directory where NavisCore is installed.

8. If the installation utility detects configuration files on your system, it prompts whether you want to use these existing files for the installation (instead of having to enter configuration values). If you answer **yes**, the utility will create symbolic links to the configuration files.

9. The installation utility prompts whether the file start-server.sh was saved from a previous installation and asks whether you want to re-use the file for this installation. If you answer **yes**, the utility prompts you for the path to the file.

10. If the installation utility detects MIB files in /opt/CascadeView/snmp_mibs, it prompts whether you want to create symbolic links to the MIB files in the /opt/ProvServ/snmp_mibs directory. And, it prompts whether you want to create a symbolic link for the Provisioning Server MIB file (provserv.mib) to in /opt/CascadeView/snmp_mibs.

11. Indicate your choices to these prompts.

12. When prompted, enter the Sybase DSQUERY name.

13. When prompted, enter the Sybase Database name for the NavisCore database.

14. When prompted, enter the Sybase system administrator user name for the NavisCore database.

15. When prompted, enter the system administrator password.

16. At the verification prompt, re-enter the system administrator password.

    The installation utility displays the values you input and allows you to change them.

17. Make any necessary changes.

18. When prompted, specify whether you want the installation utility to save copies of the configuration files and start-server.sh at de-install time. If you answer yes, the utility prompts you for the path where you want to save the file.

19. The installation utility displays the confirmation message:

    ```
    Install NAVISXtend  [version #]? (y) [y,n,?,q]
    ```

20. Enter **y** to continue.

    The installation utility prompts you to enter the package base directory.

    ```
    Enter path to package base directory [?,q]
    ```

21. Enter the path to the directory where you want the package installed.

    The installation utility performs various verification functions and displays the message:

    ```
    This package contains scripts which will be executed with
    super-user permission during the process of installing this
    package.

    Do you want to continue with the installation of this package
    [y,n,?]
    ```

22. Enter **y** to continue.

    The installation utility completes the installation and displays the message:

    ```
    Installation of <NAVISeps> was successful.
    ```

The installation of the Provisioning Server is complete. Before you run the server, perform the post-installation tasks described in "Post-Installation Tasks" on page 2-7.

Installing the Provisioning Software in a Two-System Configuration

This section describes how to install the Provisioning Server software on a separate host from NavisCore and SYBASE. For details on how to perform these tasks in NavisCore, see the *NavisCore NMS Getting Started Guide*.

1. In NavisCore, add an NMS entry to each switch the Provisioning Server will provision. Specify the IP address of the host on which the Provisioning Server will reside.
2. In NavisCore, add an NMS path, specifying the IP address of the host on which the Provisioning Server will reside.
3. On the host on which the Provisioning Server will reside, log in as the root user and enter the root password.
4. Create the /opt/sybase directory.
5. On the NavisCore host, copy the file /opt/sybase/interfaces to the /opt/sybase directory on the host on which the Provisioning Server will reside.
6. Install the Provisioning Server. Follow the instructions in "Installing the Provisioning Software in a Single-System Configuration" on page 2-4.

The installation of the Provisioning Server in a two-system configuration is complete.

Before you run the server or the CLI, perform the post-installation tasks described in the next section.

Post-Installation Tasks

This section describes post-installation steps you need to perform on the Provisioning Server, the CLI, and the Provisioning client.

Modifying the Configuration File

If, during installation, you specified that the installation utility use the default configuration files provided by the Provisioning Server, and your NavisCore database name is different than cascview, you need to modify the cvdb.cfg file. To do so, use a text editor to modify the cvdb.cfg file located in /<*install directory*>/ProvServ/etc.

Change the following entries:

```
CVDB_DB_NAME=<database-name>

CVDB_USER_NAME=<database-name>
```

where <*database-name*> is the name of the Sybase database for the NavisCore database.

Testing the Server

During server installation, the init program (/etc/inittab) was modified to cause the system to automatically restart the server process whenever the system reboots. To start the server manually for testing, issue the following command:

```
/sbin/init Q <Return>
```

This command causes the init program to read the file /etc/inittab.

Test the server to make sure that it is running and is accessible. To do so:

1. Log on as a user other than root.
2. Issue a CLI command for an existing switch in the NavisCore database:

   ```
   /opt/ProvServ/bin/cvget switch.nn.nn.nn.nn -Location<Return>
   ```

   where nn.nn.nn.nn is the decimal IP address of the switch. If the Provisioning Server is operating, the **cvget** command prints the location of the switch you specified. Verify that the returned location is valid for that switch.

For instructions on how to troubleshoot problems with the server, see "Troubleshooting Problems" on page 2-22.

Setting Environment Variables

There are several environment variables you can set to configure the Provisioning Server. Specifically, you can:

- Specify the server's local port
- Specify the server's core file location
- Enable server trace files
- Control certain SNMP parameters

For instructions on how to set these environment variables, see "Configuring the Provisioning Server" on page 2-15.

If the CLI and the Provisioning Server are located on the same host, the CLI can locate the Provisioning Server by default. If the CLI and the Provisioning Server are remote from one another, you need to identify the location and port number of the Provisioning Server. To do so, set the following environment variables in the user's shell start-up script (such as .cshrc, .login, or .profile):

CV_CLI_SERVER_HOST — Set this variable to the IP address of the remote Provisioning Server. Specify the address in either numeric format (such as 152.148.50.2) or in text format (such as provserv.xyz.com).

CV_CLI_SERVER_PORT — Set this variable to the port number of the remote Provisioning Server.

There are other environment variables you can set to configure the CLI. Specifically, you can:

- Specify whether updates are made to the network component and the database, or to the database only
- Specify security settings
- Control certain SNMP parameters

For instructions on how to set these environment variables, see "Configuring the CLI" on page 2-12.

Testing the CLI

Test the CLI to make sure that it is running and can access the Provisioning Server. To do so:

1. Log on as a user other than root.
2. Issue a CLI command for an existing switch in the NavisCore database:

   ```
   /opt/ProvServ/bin/cvget switch.nn.nn.nn.nn -Location<Return>
   ```

   where nn.nn.nn.nn is the decimal IP address of the switch. If the CLI is operating and can access the Provisioning Server, the **cvget** command prints the location of the switch you specified. Verify that the returned location is valid for that switch.

For instructions on how to troubleshoot problems with the CLI, see "Troubleshooting Problems" on page 2-22.

Recompiling an Existing Provisioning Client

If you have a Provisioning application that was built with a previous version of the Provisioning Server Application Toolkit and you want to use the new features of the Provisioning Server API, you need to make the necessary code changes for the new functions and attributes, and recompile and relink your program with the new API.

If you *do not* want to use the new features of the Provisioning Server API, no code changes are necessary. You need only to recompile and relink your program with the current version of the API include files and libraries.

Installed Files

Once you install the toolkit, the CLI commands and the files you need to write a program with the API are present on the workstation hard disk:

Command line interface and binary file — Contained in the file /opt/ProvServ/bin/cli, as well as various links contained in /opt/ProvServ/bin.

Client libraries — Contained in the directory /opt/ProvServ/lib.

Client include files — Contained in the directory /opt/ProvServ/include.

Sample code — Contained in the directory /opt/ProvServ/src. The C++ sample code is in the file CircuitDefinedPath.C.

Programming Files

Table 2-1 lists the files (located in the directory /opt/ProvServ/include) necessary for development of an NavisXtend Provisioning client program.

Table 2-1. Programming Files for Client Development

File	Description
ProvClient.h	Header file for the C APIs; contains definitions and function prototypes. If you are programming in C, include this file in your source code.
CvClient.H	Header file for the C++ APIs; contains definitions and function prototypes. If you are programming in C++, include this file in your source code.
CvDefs.H	Contains some definitions that are common to both the C and C++ APIs. The ProvClient.h and CvClient.H files contain a **#include** statement that incorporates CvDefs.H. Therefore, as long as you include either ProvClient.h or CvClient.H, you do not need to explicitly include CvDefs.H in your source code.
CvObjectType.H	Defines the enumerated object types used by both the C and C++ APIs. The ProvClient.h and CvClient.H files contain a **#include** statement that incorporates CvObjectType.H. Therefore, as long as you include either ProvClient.h or CvClient.H, you do not need to explicitly include CvObjectType.H in your source code.

Table 2-1. Programming Files for Client Development (Continued)

File	Description
CvArgId.H	Defines all argument IDs used by both the C and C++ APIs. The ProvClient.h and CvClient.H files contain a **#include** statement that incorporates CvArgId.H. Therefore, as long as you include either ProvClient.h or CvClient.H, you do not need to explicitly include CvObjectId.H in your source code.
CvParamValues.H	Defines the values for each of the enumerated attributes used by both the C and C++ APIs. Include this file in your source code.
CvObjectId.H	Defines the CvObjectId structure used by the C API to identify objects. ProvClient.h contains a **#include** statement that incorporates CvObjectId.H. Therefore, as long as you include the ProvClient.h file, you do not need to explicitly include CvArgId.H in your source code.
CvUSL.H	Defines simple wrapper classes for various unsigned long data types used by the C++ APIs. CvClient.H contains a **#include** statement that incorporates CvUSL.H. Therefore, as long as you include CvClient.H, you do not need to explicitly include CvUSL.H in your source code.
CvE164Address.H	Defines a helper class used in the C++ APIs. CvClient.H contains a **#include** statement that incorporates CvE164Address.H. Therefore, as long as you include CvClient.H, you do not need to explicitly include CvE164Address.H in your source code.
CvSVCAddress.H	Defines a helper class used in the C++ APIs. CvClient.H contains a **#include** statement that incorporates CvSVCAddress.H. Therefore, as long as you include CvClient.H, you do not need to explicitly include CvSVCAddress.H in your source code.
CvErrors.H and CvErrors.h	Define the errors that can be returned by the APIs as well as errors implemented by NavisCore. You do *not* need to include either of these files in your source code.

Setting Environment Variables

This section describes how to set environment values to configure the behavior of the CLI, the Provisioning client, and the Provisioning Server. To configure the Provisioning Server, add the environment variables to the start-up script that launches the server. To configure the Provisioning client or the CLI, add the environment variables to the user's shell start-up script, such as .cshrc, .login, or .profile.

Configuring the CLI

There are several environment variables you can use to configure the CLI. Specifically, environment variables perform the following:

- Identify the Provisioning Server to which the CLI sends requests.
- Specify whether updates are made to the network component and the database, or to the database only.
- Control retry behavior of circuit provisioning requests.
- Specify security settings.
- Control certain SNMP parameters.

The best way to set the environment variables is to add them to the user's shell start-up script (such as .cshrc, .login, or .profile)

Identifying the Provisioning Server to the CLI

If the CLI and the Provisioning Server are running on the same host, the CLI can locate the Provisioning Server by default. If the CLI and the Provisioning Server are remote from one another, you need to identify the location and port number of the Provisioning Server. To do so, set the following environment variables:

CV_CLI_SERVER_HOST — Set this variable to the IP address or hostname of the remote Provisioning Server. Specify the address in numeric format (for example, **152.148.50.2**). Specify the hostname in text format (for example, **provserv.xyz.com**). If you do not set this variable, the CLI uses the local host by default.

CV_CLI_SERVER_PORT — Set this variable to the port number of the remote Provisioning Server. If you do not set this variable, the CLI uses port 4001 by default.

Specifying Modification Type

You can specify whether updates are made to the network components and the database, or to the database only. Set the following environment variable:

CV_CLI_MOD_TYPE — Set this variable to the number that represents the update method, as follows:

1 — Sends updates to both the network component and the database. If the network component updates successfully, the database is updated.

4 — Sends updates to the database only.

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

If you do not set this variable, the CLI sends updates to both the network component and the database by default.

Specifying Retry Behavior

You can specify retry behavior in the event of a failed attempt to add, delete, or modify a circuit.

By default, when the Provisioning Server receives a request to add, delete, or modify a circuit, the server obtains card status for both circuit endpoints:

- If both cards are up, the Provisioning Server performs the add, delete, or modify request as normal.
- If either card is down or is not reachable (for example, because of an SNMP timeout), the server retries the request for card status as many times as specified by the retry control (in the range of 0 to 5 times):
  - If the card becomes reachable and is up, the Provisioning Server performs the circuit provisioning request.
  - Once all retries have been issued, if the card is still not reachable or is still down, the provisioning request is not performed.

This control prevents circuits from being partially provisioned and the database from becoming out of sync with the switch. However, it can increase the time it takes to provision a circuit, depending on how many card status checks occur.

To specify the retry control, set the following environment variable:

CV_CLI_NUM_RETRIES — Set this variable to the number of retries (from 0 - 5) for requests of card status to precede circuit provisioning requests. 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.

If you do not set this variable, or you set it out of range, the CLI does not retry the request for card status.

Keep in mind that this control affects the retry behavior of circuit provisioning requests only. Other retry controls specified in cascadeview.cfg (CV_SNMP_MAX_RETRIES, CV_SNMP_RETRY_INTERVAL, and CV_SNMP_REQUEST_TIMEOUT) also apply to each request. Remember to consider these other retry controls when specifying retry behavior of a circuit request.

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 "Disabling Card Status Checking" on page 2-20.

Specifying Security Settings

By default, the Provisioning Server accepts requests from the CLI without requiring authorization. You can implement a security feature that authenticates user logins. The feature is intended to prevent users from accidentally modifying the database; it is *not* intended to prevent intentional misuse by users. To implement the security feature, you must specify environment variables for both the CLI and the Provisioning Server. To do so for the CLI, set the following environment variables:

CV_CLI_USE_LOGINS — Set this variable to any value (including a null value) to turn on the security feature. If you do not set this variable, the Provisioning Server accepts requests from the CLI without requiring authorization. If you set this variable, you must also set the following variables:

CV_CLI_USERNAME — Set this variable to the username character string required by NavisCore (for example, operator).

CV_CLI_PASSWORD — Set this variable to the password character string required by NavisCore.

The username and password character strings are sent over the network as nonencrypted text.

To fully implement the security feature, you must also specify security settings on the server side. For instructions, see "Implementing the Security Feature" on page 2-21.

Controlling SNMP Parameters

You can specify how certain SNMP parameters are controlled. To do so, set the following environment variables:

CV_SNMP_REQUEST_TIMEOUT — Set this variable to the amount of time (in 0.01 second increments) that the CLI waits for a response from the server. If you do not set this variable, the CLI uses the value 256 by default.

CV_SNMP_MAX_RETRIES — Set this variable to the number of times that the CLI retries a request that times out. If you do not set this variable, the CLI uses the value 0 by default.

Configuring the Provisioning Client

There are several environment variables you can set to configure the Provisioning client. Specifically, environment variables perform the following functions:

- Enable a client trace file
- Control certain SNMP parameters

The best way to set the environment variables is to add them to the user's shell start-up script (such as .cshrc, .login, or .profile).

Enabling a Client Trace File

You can specify that the client create a trace file. Such a file can be useful for debugging your Provisioning client. It is recommended that you enable the trace file until the Provisioning Server and your Provisioning client are running in a production environment. To enable a client trace file, set the following environment variable:

CV_CLIENT_TRACE_FILE — Set this variable to the pathname of the file to contain the trace output (for example, /tmp/ctrace.log). If you do not set this variable, no trace file is created.

Once you enable a client trace file, each session of the client is recorded in the file. Output is continuously appended to the file. If you are not debugging your Provisioning client, it is recommended that you periodically delete the file.

Controlling SNMP Parameters

You can specify how certain SNMP parameters are controlled. To do so, set the following environment variables:

CV_SNMP_REQUEST_TIMEOUT — Set this variable to the amount of time (in 0.01 second increments) that the client waits for a response from the server. If you do not set this variable, the client uses the value 256 by default.

CV_SNMP_MAX_RETRIES — Set this variable to the number of times that the client retries a request that times out. If you do not set this variable, the client uses the value 0 by default. For non-MIB clients, you can set this variable to any value. For MIB clients, you can set this variable to any value only if you set the variable CV_SNMP_DISCARD_RETRY to 1.

Configuring the Provisioning Server

There are several environment variables you can set to configure the Provisioning Server. Specifically, environment variables perform the following functions:

- Specify the server's local port and the MIB agent's port
- Specify the server's core file location
- Enable server trace files
- Control certain SNMP parameters
- Control ListContained context timeout
- Control MIB cache

- Control object locking
- Disable card status checking when provisioning circuits
- Specify SNMP community strings
- Specify how the server formats SMDS addresses
- Implement security feature

The Provisioning Server reads its configuration settings from two files also used by NavisCore. These shared configuration files are as follows:

- /opt/ProvServ/etc/cvdb.cfg
- /opt/ProvServ/etc/cascadeview.cfg

Rather than modifying Provisioning Server's environment variables directly in these files (which would also affect NavisCore), you can enable them in the server's start-up script (/opt/ProvServ/bin/start-server.sh). Look for the invocation of cvdb.cfg and cascadeview.cfg in start-server.sh and make the necessary modifications after that point in the file.

Identifying the Provisioning Server Port

The Provisioning Server uses a command line argument to identify which port to listen for API and CLI requests. To specify this command line argument, set the following environment variable in start-server.sh:

CV_PSRV_ARGS — Set this variable to the command **-lport** and the port number. Enclose the command in quotation marks, for example:

```
"-lport 4002"
```

If you do not set this variable, the Provisioning Server uses port 4001 by default.

Identifying the MIB Agent Port

The Provisioning Server implements an SNMP agent as a separate entity to service MIB interface requests. The server uses an environment variable to identify which port to listen for SNMP requests. This port is different from the port number used to listen for API and CLI requests.

To specify this MIB agent port, set the following environment variable in start-server.sh:

CV_SNMP_AGENT_PORT — Set this variable to the port number. If you do not set this variable, the Provisioning Server uses port 9090 by default.

Specifying the Core File Location

If the Provisioning Server crashes, it creates a core file. Such a file can be useful for debugging the server. The core file is written to the Provisioning Server's working directory (/tmp by default). You can specify the directory where the Provisioning Server runs and where it writes any core files. You may want to specify a directory other than the default if /tmp gets deleted frequently and you want to ensure a valid core file. To specify a working directory, set the following environment variable in start-server.sh:

CV_WORKING_DIR — Set this variable to the pathname of the directory. If you do not set this variable, the Provisioning Server writes its core file to the /tmp directory by default.

Enabling Server Trace Files

By default, the Provisioning Server creates three trace files, two of which are enabled by environment variables specified in the configuration file cascadeview.cfg. Rather than turning these trace files on or off directly in cascadeview.cfg (which would also affect NavisCore), you can enable them in the server's start-up script (/opt/ProvServ/bin/start-server.sh). Look for the invocation of cascadeview.cfg in start-server.sh and make the necessary modifications after that point in the file.

These files can be useful for troubleshooting and diagnosing problems. It is recommended that you enable the trace files until the Provisioning Server is running in a production environment. To enable the trace files, set the following environment variables:

CV_TRACE_ENABLE — Set this variable to 1 to enable the application-level trace output for the server. If you set this variable, you must also set the **CV_TRACEFILE** variable.

CV_TRACEFILE — Set this variable to the pathname of the file to contain the application-level trace output for the server. To avoid conflicts with the NavisCore trace file, the suffix .psrv will be appended to the filename you specify. By default, this trace file is written to the /tmp directory.

CVDB_TRACE_FILE_NAME — Set this variable to the pathname of the file to contain the database trace output for the server. To avoid conflicts with the NavisCore trace file, the suffix .psrv will be appended to the filename you specify. By default, this trace file is written to the /tmp directory.

CV_PSRV_TRACE_FILE — Set this variable to the pathname of the file to contain trace output specific to the Provisioning Server. By default, this trace file is written to the /tmp directory and is called strace.log.

Once you enable a trace file, specific activity is recorded in the file. Output is continuously appended to the file. It is recommended that you periodically delete the trace files.

If you are troubleshooting a problem, it can be useful to know what kinds of transactions occur between the Provisioning Server and the Provisioning client. For this reason, you should enable the client trace file as well. For instructions, see "Enabling a Client Trace File" on page 2-15.

Controlling SNMP Parameters

You can specify how certain SNMP parameters are controlled. To do so, set the following environment variables in start-server.sh:

CV_SNMP_REQUEST_TIMEOUT — Set this variable to the amount of time (in 0.01 second increments) that the server waits for a response from the switch. If you do not set this variable, the server uses the value 256 by default.

CV_SNMP_MAX_RETRIES — Set this variable to the number of times that the server retries a request that times out. If you do not set this variable, the server uses the value 5 by default. It is recommended that you keep this setting as the default.

CV_SNMP_DISCARD_RETRY — Set this variable to 1 to enable the server to discard multiple SNMP request retries from a MIB client. If you set this variable to 1, the server checks the Request ID, the IP address, and the port number of every SNMP request. If these values match those of a request that the server is currently processing, the server ignores the retry request. If you do not set this variable, the server uses the value 1 by default. It is recommended that you keep this setting as the default, unless your SNMP client generates SNMP PDUs without unique Request IDs or port numbers.

Controlling Context Timeout

The Provisioning Server maintains context for outstanding ListContained requests. The server allows 500 ListContained requests to be outstanding. Any ListContained request for which a NextObject request has not been issued within a configurable time period is subject to deletion to make room for a new ListContained request to be processed. To configure this time period, set the following environment variable in start-server.sh:

CV_PSRV_CONTEXT_TIMEOUT — Set this variable to the amount of time (in minutes) that the server waits for a response to a ListContained request. Any request for which a NextObject request has not been issued in that time period is subject to deletion. If you do not set this variable, the server uses the value 10 by default. It is recommended that you keep this setting as the default.

Controlling MIB Cache

The Provisioning Server implements a MIB cache that stores data in memory for a fixed time period. The server uses this cache to optimize performance of **get-next** requests and to store data to be committed to the database during transactions involving multiple PDUs. Each table row stored in cache has a timestamp. The server uses an environment variable to purge older data by row.

To configure this purge time period, set the following environment variables in start-server.sh:

CV_SNMP_ROWENTRY_TIMEOUT — Set this variable to the amount of time (in seconds) that the server stores a particular row of data in cache during a **get-next** request. Based on this variable, the server flushes out entries in MIB cache that result from a **get-next** operation. Thus, the server uses this variable to optimize performance of **get-next** requests. The minimum value of this variable is 60, the maximum value is 1800. These values apply, even if you set a value lower than the minimum or greater than the maximum. If you do not set this variable, the server uses the timeout value 900 by default.

CV_SNMP_CMDERROR_CACHE_TIMEOUT — Set this variable to the amount of time (in seconds) that the server stores Command Error Table messages in cache. The Command Error Table contains error messages generated by the SNMP agent. Any error message older than this timeout value is subject to deletion. If you do not set this variable, the server uses the timeout value 3600 by default. To save all errors generated during creation and modification transactions, set this variable to a value greater than the value of the CV_SNMP_LOCK_TIMEOUT variable.

Controlling Object Locking

The Provisioning Server uses an object locking scheme for MIB objects in the database that differs from the locking behavior of the Provisioning Server API, CLI, or NavisCore. For these interfaces, the steps associated with locking are transparent to the user. When an object is created or modified, its parent object becomes locked. The user specifies all the information needed to create or modify the object in one PDU. Once the request completes, the parent becomes unlocked.

By contrast, in the case of the MIB, the information needed to create or modify an object may not be available in one PDU. As a result, the locks in the database must be held for a longer time. Thus, the steps associated with locking are not transparent to the user.

If the user initiates a transaction to create an object, the parent object becomes locked, preventing other users from modifying it. If the user initiates a transaction to modify an object, the object itself becomes locked, preventing other users from modifying it. To configure the time period that objects are locked, set the following environment variable in start-server.sh:

CV_SNMP_LOCK_TIMEOUT — Set this variable to the amount of time (in seconds) that the server:

- Locks a parent object when a child object is being created
- Locks an object that is being modified

The maximum value of this variable is 1800. This maximum value applies, even if you set a greater value. If you do not set this variable, the server uses the timeout value 900 by default.

Disabling Card Status Checking

The Provisioning Server uses a retry control to ensure reliability and accuracy of circuit provisioning. This control specifies retry behavior in the event of a failed attempt to add, delete, or modify a circuit.

By default, when the Provisioning Server receives a request to add, delete, or modify a circuit, the server obtains card status for both circuit endpoints before it performs the provisioning request.

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 following environment variable in start-server.sh:

CV_CARD_STATS — Set this variable to DISABLE to disable card status checking on circuit endpoints.

Specifying Community Strings

The Provisioning Server supports two community names, one for Read-Only operations and one for Read-Write operations. The community name provides a mechanism for authentication and access-control at the SNMP agent.

The community strings are defined using the following environment variables in start-server.sh:

CV_READONLY_COMMUNITY_STRING — Set this variable to the community string to be used when making a Read-Only SNMP request. If you do not set this variable, the server uses the value 'public' by default.

CV_READWRITE_COMMUNITY_STRING — Set this variable to the community string to be used when making a Read-Write SNMP request. If you do not set this variable, the server uses the value 'ascend' by default.

If these environment variables are defined in the script start-server.sh, the specified strings take precedence. If they are not set in the script or if the server shell environment does not define the variables, the server assumes the default values.

If the community name is not valid when you issue an snmp_set request, the request exceeds the time-out period and fails. You can access the Command Error Table in the MIB to see if the source of the problem is an invalid community name. Specify the Read-Only community name when you access the table, as that community name is used for validation purposes.

When you make an snmp_get request, specify either the Read-Only or the Read-Write community name. If you use a different community name and you encounter an error, the error is not propagated to the Command Error Table.

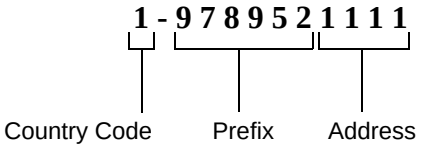
Controlling SMDS Addresses

You can specify how the Provisioning Server formats SMDS addresses. To do so, set the following environment variables in start-server.sh:

CV_DFLT_SMDS_CC — Set this variable to specify the default country code for SMDS addresses. The server will prepend this default country code to a given address that does not specify the country code. When using multiple country codes, you must specify the country code for addresses that do not use the default code. You have to create a country code before you can specify it as a default.

CV_DFLT_CC_PRT_ENABLE — Set this variable to control the format of individual addresses in responses to List operations. When this variable is set to 1, the default country code part of an address is returned in the List response. For other operations (AddObject, DeleteObject, Get, Modify), the server returns the address in the same format used by the client.

CV_ SMDS_MASK_SIZE — Set this variable to specify the character length of the address prefix in SMDS addresses. The server interprets characters preceding a dash (-) as the country code part of the address, the next *n* characters (specified by this variable) as the prefix part of the address, and the remainder as the address part. For example, if this variable is set to 6, the server interprets the address 1-9789521111 as follows:



Implementing the Security Feature

By default, the Provisioning Server accepts requests from Provisioning client and CLI users without requiring authorization. You can implement a security feature that authenticates user logins. The feature is intended to prevent users from accidentally modifying the database; it is *not* intended to prevent intentional misuse by users. To implement the security feature, you must specify environment variables for both the CLI and the Provisioning Server. To do so for the server, set the following environment variable in start-server.sh:

CV_PSRV_USE_LOGINS — Set this variable to any value (including a null value) to turn on the security feature. If you do not set this variable, the Provisioning Server accepts requests from clients without requiring user authorization.

Once you set this variable, any clients sending requests to the server must send a user ID and password for authorization. For a Provisioning client, this is accomplished when the client establishes the session with the Provisioning Server. For the CLI, the security settings are specified through environment variables. For instructions, see "Specifying Security Settings" on page 2-14.

Stopping and Restarting the Provisioning Server

To stop and restart the Provisioning Server running on a workstation:

1. On the host that runs the server, log on as the root user and enter the root password.
2. Determine the process ID of the Provisioning Server, using the following command:

   ```
   /bin/ps -ef | grep provserv <Return>
   ```

   The process ID is the second item in the resulting listing.
3. Kill the current server process, using the following command:

   ```
   kill [server process id] <Return>
   ```

Once the server process is killed, the init program restarts the server.

Stopping and Restarting the CLI

To quit the CLI, press <Ctrl-C>. To restart the CLI, issue a CLI command.

Troubleshooting Problems

This section describes how to troubleshoot problems with the Provisioning Server, the provisioning application, and the CLI.

Problem: Requests Frequently Time Out

Symptoms

Either:

- CLI prints an error message
- API-based application receives an error status

Possible Causes and Solutions

Scenario 1: Error message 4109 ("Request to the server timed out")

- The Provisioning Server may not be running or accessible to the client workstation. Verify that the client can access the server and that the Provisioning Server is running. To do so, follow the procedure in "Testing the CLI" on page 2-9.
- The client's timeout value may be too low. The client timeout value should allow for instances when the server times out and later retries a command to the switch. Since the server's second request may be successful, the client should not timeout while waiting for the server's response. Adjust the client timeout value by setting the client CV_SNMP_REQUEST_TIMEOUT environment variable. Start with the value 3000. If that value does not correct the problem, use the following formula to determine a "worst case" client timeout value:

  CV_SNMP_REQUEST_TIMEOUT = *CV_SNMP_REQUEST_TIMEOUT* * *CV_SNMP_MAX_RETRIES* + *n*

  where *CV_SNMP_REQUEST_TIMEOUT* is the timeout value for the server, *CV_SNMP_MAX_RETRIES* is the retry value for the server and *n* is a factor that allows for client-server round-trip. Start with an *n* value of 300. The result (CV_SNMP_REQUEST_TIMEOUT) is the timeout value to set for the client.

For details on values to use for these environment variables for the client and server, see "Setting Environment Variables" on page 2-11.

Scenario 2: Error message 42 ("The SNMP request to the agent timed out"):

The Provisioning Server may take a long time to process a request because it cannot locate the network device specified in the request (such as a switch):

- If the request is intended to modify the switch, verify that the switch is accessible from the server. To do so, remotely log into the Provisioning Server and issue the ping utility to elicit a response from the switch.

- If the request is intended to update the database only, retry the request with the modification type set to update the database only. For a CLI request, set the CV_CLI_MOD_TYPE environment variable to 4 or 5 (see "Configuring the CLI" on page 2-12.) For an API request, issue either the C function **CvSetModifyType** or the C++ member function **CvClient::setModifyType**, specifying that updates be made to the database only.

Problem: Object Is Locked by Others

Symptoms

Either:

- CLI prints an error message
- API-based application receives an error status

Possible Causes and Solutions

Either a NavisCore user has the object locked or the object appears to be locked when the client retries a request. To determine if the object is locked, change directories to /opt/CascadeView/bin and execute the cv-release-locks.sh shell script. The script lists the objects that are currently locked and who has them locked.

Do not use the cv-release-lock.sh script to release the locks. If you need to release locks, call the Ascend Technical Assistance Center.

Scenario 1: Object Is Locked by NavisCore User

If the cv-release-locks.sh shell script indicates that a NavisCore user has the object locked, either:

- Wait for the user to finish (or request that he or she finish) using the object.
- Call the Ascend Technical Assistance Center.

Scenario 2: Object Appears to Be Locked During Retries

If the cv-release-locks.sh shell script does not indicate that the object is locked, a client timeout may have occurred while the server was still processing the request. Then, when the client automatically retried the request, the object appeared to be locked.

Adjust one of the following environment variables:

- Adjust the client timeout value by setting the client's CV_SNMP_REQUEST_TIMEOUT environment variable to a higher value. To do so, follow the procedure in "Scenario 1: Error message 4109 ("Request to the server timed out")" on page 2-23.
- Adjust the client retry value by setting the CV_SNMP_MAX_RETRIES to 0. For details on this environment variable for the client, see "Setting Environment Variables" on page 2-11.

Technical Support

The Ascend Technical Assistance Center (TAC) is available to assist you with any problems encountered while using the NavisXtend Provisioning Server product. To contact the Ascend TAC, call 1-800-DIAL-WAN.

Information Checklist

Before contacting the Ascend TAC, review the following checklist to make sure you have gathered all the information you need:

Software Version Number

Use the UNIX utility pkginfo to obtain information such as version number and install date for the NavisXtend Provisioning Server package:

```
pkginfo -l NAVISeps
```

Note the version number listed in the output.

Problem Report

Collect as much information as possible about the problem:

- For CLI problems, describe what commands caused the problem, what commands preceded the problem, and how did the Provisioning Server respond (such as what error message was returned). If possible, provide the exact text for the commands.
- For API problems, provide the source code that caused the problem. Try to condense the problem to a few lines.
- You can use the API to create a CLI command that recreates the problem. This alternative provides an easy way to recreate a problem without having to provide code. The following code sample illustrates how to use the API to create a CLI command:

```
char *argString = CvArgsToString( args );
char *objString = CvObjectIdToString( objid );
printf( “cvadd %s %s”, objString, argString );
CvStringFree( argString );
CvStringFree( objString );
```

Trace Files

Collect any trace files that may exist:

- Server trace files, which you enable using environment variables. By default, these trace files are not produced. The easiest way to turn them on is to edit the start-server.sh script. They are usually written to the /tmp directory with the filenames strace.log or the file suffix .psrv.
- Client trace files, which you enable using environment variables. By default, these trace files are not produced. The easiest way to turn them on is to edit the user’s .cshrc file and adding the following line:

  `setenv CV_CLIENT_TRACE_FILE /tmp/ctrace.log`

  This command writes the client trace file ctrace.log to the /tmp directory.

If the resulting trace files are too large, collect the last 5000 lines of each file. If necessary, compress the files using the GZIP program. If you send the compressed files to Ascend by email, UUENCODE the files, if necessary.

For more information on how to enable trace logs, see “Enabling Server Trace Files” on page 2-17 and “Enabling a Client Trace File” on page 2-15.

Core Files

If the Provisioning Server crashes, it creates a core file. Collect the core file from the Provisioning Server’s working directory, which is either /tmp by default or another directory you specify using an environment variable. For information on how specify the working directory, see “Specifying the Core File Location” on page 2-17.

Un-installation Instructions

If you decide you want to un-install the current version of the Provisioning Server and Application Toolkit, use the pkgrm utility:

1. To un-install the Provisioning Server components using pkgrm, enter:

   `pkgrm NAVISeps`

   The utility prompts you to verify the un-install:

   ```
   The following package is currently installed:
      1 NAVISeps NavisXtend Provisioning Server
                (sparc) [version #]


   Do you want to remove this package?
   ```

2. To un-install the NavisXtend Provisioning Server package, enter **y**.

   The un-installation utility displays the message:

   ```
   ## Removing installed package instance <NAVISeps>

   This package contains scripts which will be executed with
   super-user permission during the process of removing this package.

   Do you want to continue with the removal of this package [y,n,?, q]
   ```

3. Enter **y** to continue.

   The un-installation utility performs various verification functions and displays the confirmation message:

   ```
   Are you sure you want to UNINSTALL the Provisioning Server [y/n]?
   ```

4. Enter **y** to continue.

   The utility completes the un-installation:

   ```
   Un-install complete

   Removal of <NAVISeps> was successful.
   ```

The un-installation of the Provisioning Server components is complete.

Writing a Provisioning Application

To write a Provisioning application, perform the following steps:

1. Install the Provisioning Server Application Toolkit, as described in "Installation Instructions" on page 2-3.
2. Set the environment variables that control SNMP parameters for the Provisioning client. For instructions, see "Configuring the Provisioning Client" on page 2-14.

3. Add the following entries to your makefile:

   ```
   -I/opt/ProvServ/include
   -L/opt/ProvServ/lib -lClient
   ```

   The first line is for all compilations; the second line is for the link step.

4. Write the program.
5. Compile the program.

Upgrading an Existing Application

If you have a Provisioning application that was built with a previous version of the Provision Server Application Toolkit and you want to use the new features of the Provisioning Server API, you need to make the necessary code changes for the new functions and attributes, and recompile and relink your program with the new API.

If you *do not* want to use the new features of the Provisioning Server API, no code changes are necessary. You need only to recompile and relink your program with the current version of the API include files and libraries.

3

Using the CLI

This chapter describes how to use the Command Line Interface (CLI) to build a provisioning script instead of a C or C++ program.

To understand the Provisioning Server object hierarchy, first read Chapter 1.

Using the CLI

The Application Toolkit provides a Command Line Interface (CLI) for users to build a provisioning script instead of a C or C++ program. The CLI is a set of command-line programs that you can issue from any UNIX shell to provision network objects in interactive or batch mode.

There is a CLI command for each operational function of the API. Each command uses a string representation to specify objects and attributes.

cvadd (Object ID, Attributes) — Creates an object in the database and (optionally) in the switch.

cvaddmember (Object ID, Object ID) — Adds a member to an object list.

cvCreateChanPerformanceMonitorId (Object ID, Channel ID) — Creates a CVT_ChanPerformanceMonitor object.

cvmodify (Object ID, Attributes) — Modifies specific attributes of an object.

cvdelete (Object ID) — Deletes an object from the database and (optionally) from the switch.

cvdeletemember (Object ID, Object ID) — Deletes a member from an object list.

cvget (Object ID, Attributes) — Retrieves specific attribute values from the database.

cvgetdiag (Object ID, Attributes) — Retrieves specific diagnostic information from the Provisioning Server.

cvgetoperinfo (Object ID, Attributes) — Retrieves the values of specific real time operational information from the switch.

cvlistcontained (Object ID, type, Attributes) — Retrieves a list of configuration attributes for objects of the given type contained by the specified parent.

cvlistallcontained (Object ID, Attributes) — Retrieves a list of configuration attributes for all objects contained by the specified parent.

cvstartdiag (Object ID, Attributes) — Starts diagnostics on an object in the network.

cvstopdiag (Object ID, Attributes) — Stops diagnostics on an object in the network.

cvupdatediag (Object ID, Attributes) — Modifies diagnostic parameters on the switch for an object being diagnosed on the network.

The commands are supported for most target object types, with a few restrictions. For example, you cannot specify a switch when you issue an Add or Delete command, as the Provisioning Server does not support adding or deleting switches.

The **cvhelp** command provides usage help for the CLI.

For a list of the object types you can use when you issue the operational functions of the CLI, see Table 1-4 on page 1-39.

There are several environment variables you can use to configure the behavior of the CLI. For details, see “Setting Environment Variables” on page 2-11.

CLI Usage Overview

Most of the CLI commands use the following syntax:

command object-name {*-attribute-name value*}

Syntax

command	The name of the command. If the command is in your path, you can enter just the command name, such as **cvadd**, **cvdelete**, **cdmodify**, **cvget**, **cvaddmember**, or **cvdeletemember**. Otherwise, you must prefix the command name with the path **/opt/ProvServ/bin/**.
object-name	The object ID. To specify an object ID, you first specify the object’s parent (if any), including the parent type and value. Then, you specify the child type and value. For rules on specifying object IDs for various types of objects, see “Managed Objects” on page 1-12.

-attribute-name	The attribute ID appropriate to the object ID. Specify the attribute name preceded by a dash (-). Use the attribute ID symbols listed in the *NavisXtend Provisioning Server Object Attribute Definitions*, but omit the CVA_*ObjectType* prefix. For example, specify location as: `-Location`.
value	The value of the attribute ID. The value requires a data type appropriate for the argument, such as integer, string, and so on. For data types, use the data types listed in the *NavisXtend Provisioning Server Object Attribute Definitions*. Note the following rules for values: • For integers, specify the integer value. • For strings, enclose the string within /" characters if it contains special characters, such as a period or a blank character. String values cannot begin with a hyphen. • For enumerated types, specify the text value that represents the integer value. In most cases, the CLI uses an abbreviated text value. • For Object ID, specify the Object ID that identifies the object in the containment hierarchy (see "Managed Objects" on page 1-12).

-attribute-name and *value* are optional parameters.

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

To maximize CLI efficiency, *do not* set all possible attributes in a request. Specify only attributes that are mandatory.

In some cases, a CLI command line may become too long for the shell to handle. This can happen most often when adding LPorts. The restriction is most likely to happen when using the `sh` or `csh` shells. It occurs only in certain circumstances when using `ksh`. To work around this buffer restriction, separate the CLI command into multiple lines. At the end of each line, insert the backslash character (\) immediately followed by the `<Return>` key. This instructs the shell that the next line is part of the same command.

For example:

```
cvadd switch.1.1.1.1.card.9.pport.1.lport.2 \<Return>
-serviceType smds -smdsType SsiDte \<Return>
-bandwidth 64000
```

The sections that follow present the CLI commands in alphabetical order. Usage examples are provided with each command. Use these examples as guidelines for syntax and usage. The exact attributes required by a particular command vary, depending on the type of LPort and Card specified. For additional examples, see "CLI Examples" on page 3-30.

cvadd

Purpose

Creates an object in the database and (optionally) in the switch. The attributes specified by the command are used to initialize the object.

Command Syntax

cvadd *object-name* {*-attribute-name value* } **. . .**

Parameters

object-name specifies the object to be added. The object is specified by its object ID, based on the containment hierarchy (for information, see "Managed Objects" on page 1-12).

-attribute-name value specifies an attribute and its value to be added to the object. The attribute is specified by its argument name. The value uses a data type appropriate for the argument.

Specify only those attributes and values appropriate for the object type. You can specify any attribute except one with either the Read-Only access restriction.

Notes

For a list of object types that you can add with this command, see Table 1-4 on page 1-39.

To create a card or PPort, use the Modify command (**cvmodify**). The NavisCore database automatically populates each switch with cards of type "empty". Use the Modify command to change the card's type from "empty" to the specified type. Likewise, once a card has been configured, NavisCore automatically populates the card with all necessary Physical Ports. Use the Modify command 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 command to change the channel specifications.

If **cvadd** is successful, it prints the command name followed by the arguments that the Provisioning Server returns. You can use this output to verify that the arguments are the same as those specified in the original request. Any attribute that is missing a valid value is a required attribute that you omitted.

Examples

The following **cvadd** command creates an LPort:

```
/opt/ProvServ/bin/cvadd

Switch.1.1.1.2.card.4.pport.3.lport.1 -Name lport1 -SmdsType SsiDte
-ServiceType Smds -Bandwidth 64000 -ErrorPerMinThreshold 0
-AdminStatus Up -ErrorCheckFlag Off -HeartBPFlag On
-SmdsPduViolTcaFlag Disable -HeartBPInterval 1 -HeartBPNAThresh 1
```

If successful, the command returns the following text:

```
/opt/ProvServ/bin/cvadd Switch.1.1.1.2.card.4.pport.3.lport.1
-Name lport1   -SmdsType SsiDte  -ServiceType Smds  -Bandwidth
64000  -ErrorPerMinThreshold 0  -AdminStatus Up  -ErrorCheckFlag
Off  -HeartBPFlag On  -SmdsPduViolTcaFlag Disable  -HeartBPInterval
1  -HeartBPNAThresh 1
```

The following **cvadd** command creates a circuit connecting a Frame Relay LPort to a PPPto1490 LPort:

```
/opt/ProvServ/bin/cvadd
-Name circuit1 Switch.1.1.1.1.card.5.pport.5.lport.5.dlci.22
-Endpoint2 Switch.1.1.1.2.card.5.pport.5.lport.5.dlci.23
-GracefulDiscard Enabled -AdminStatus Up -Priority Low
-RerouteBalance Disabled
```

In this example:

- The first endpoint (Switch.1.1.1.1.card.5.pport.5.lport.5) is a Frame Relay LPort.
- The second endpoint (Switch.1.1.1.2.card.5.pport.5.lport.5) is a PPPto1490 LPort.

cvaddmember

Purpose

Adds a member to an object list. Use this command 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 command, the address or LPort represented by the second object parameter is added to the object specified by the first object parameter.

Command Syntax

cvaddmember *object-name object-name*

Parameters

object-name specifies the objects. Each object is specified by its object ID, based on the containment hierarchy (for information, see "Managed Objects" on page 1-12). The first *object-name* specifies the container object.

Notes

For a list of object types that you can add with this command, see Table 1-4 on page 1-39.

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.

If **cvaddmember** is successful, it prints the command name followed by the arguments that the Provisioning Server returns. You can use this output to verify that the arguments are the same as those specified in the original request.

Example

The following **cvaddmember** command adds an SMDS local individual address to an SMDS netwide group address:

```
/opt/ProvServ/bin/cvaddmember
Network.1.1.1.0.NetwideGroupAddress.1234567899
Switch.1.1.1.1.card.3.pport.4.lport.1.LocalIndividualAddress.12345
67890
```

If successful, the command returns the following text:

```
/opt/ProvServ/cvaddmember Network.1.1.1.0.NetwideGroup
Address.123456789 9 Switch.1.1.1.1.card.3.pport.4.lport.1.
LocalIndividualAddress.1234567890
```

cvCreateChanPerformanceMonitorId

Purpose

Creates a CVT_ChanPerformanceMonitor object.

Command Syntax

cvCreateChanPerformanceMonitorId *object-name channelID*

Parameters

object-name specifies the object to be created. The object is specified by its object ID, based on the containment hierarchy (for information, see "Managed Objects" on).

channelID is the **CvObjectId** structure that specifies the channel that contains the DS1 channel PM Threshold object. This structure is built by **CvCreateChannelId**.

Notes

Example

The following **cvCreatePerformanceChannelMonitorId** command creates a CVT_ChanPerformanceMonitor object.:

If successful, the command returns the following text:

```
/opt/ProvServ/bin/cvdelete Switch.1.1.1.1.card.4.pport.1.lport.1.
dlci.16
```

cvdelete

Purpose

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

Command Syntax

cvdelete *object-name*

Parameters

object-name specifies the object to be deleted. The object is specified by its object ID, based on the containment hierarchy (for information, see "Managed Objects" on page 1-12).

Notes

For a list of object types that you can delete with this command, see Table 1-4 on page 1-39.

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 the Modify command (**cvmodify**) to change the card's type to "empty".

Some objects cannot be deleted until the objects they contain have been deleted. For example, you cannot delete an LPort until you delete all of its circuits and addresses.

If **cvdelete** is successful, it prints the command name followed by the arguments that the Provisioning Server returns. You can use this output to verify that the arguments are the same as those specified in the original request.

Example

The following **cvdelete** command deletes a circuit:

```
/opt/ProvServ/bin/cvdelete
Switch.1.1.1.1.card.4.pport.1.lport.1.Dlci.16
```

If successful, the command returns the following text:

```
/opt/ProvServ/bin/cvdelete Switch.1.1.1.1.card.4.pport.1.lport.1.
dlci.16
```

cvdeletemember

Purpose

Deletes a member from an object list. Use this command to delete an address from a screen or netwide group address or to unbind a MLFRMember LPort from a MLFRBundle LPort. Upon completion of the command, the address or LPort represented by the second object parameter is removed from the object specified by the first object parameter.

Command Syntax

cvdeletemember *object-name object-name*

Parameters

object-name specifies the objects. Each object is specified by its object ID, based on the containment hierarchy (for information, see "Managed Objects" on page 1-12). The first *object-name* specifies the container object.

Notes

For a list of object types that you can delete with this command, see Table 1-4 on page 1-39.

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.

If **cvdeletemember** is successful, it prints the command name followed by the arguments that the Provisioning Server returns. You can use this output to verify that the arguments are the same as those specified in the original request.

Example

The following **cvdeletemember** command removes an SMDS alien group address from an SMDS group screen:

```
/opt/ProvServ/bin/cvdeletemember
Switch.1.1.1.1.card.3.pport.4.lport.1.GroupScreen
Switch.1.1.1.1.AlienGroupAddress.0009998887
```

If successful, the command returns the following text:

```
/opt/ProvServ/cvdeletemember Switch.1.1.1.1.card.3.pport.4.lport.1
.GroupScreen Switch.1.1.1.1.AlienGroupAddress.0009998887
```

cvget

Purpose

Retrieves the values of specific attributes from the database.

Command Syntax

cvget *object-name* {*-attribute-name* } **...**

Parameters

object-name specifies the object whose attributes are to be retrieved. The object is specified by its object ID, based on the containment hierarchy (for information, see "Managed Objects" on page 1-12).

-attribute-name specifies an attribute to be retrieved. The attribute is specified by its argument name. Specify only attribute names with no values. You can specify up to 40 attributes.

Specify only those attributes appropriate for the object type.

Notes

For a list of object types that you can use with this command, see Table 1-4 on page 1-39.

If **cvget** is successful, it prints the command name followed by the arguments that the Provisioning Server returns. You can use this output to verify that the arguments are the same as those specified in the original request.

Examples

The following **cvget** command retrieves the type and administrative status of a card:

```
/opt/ProvServ/bin/cvget
Switch.1.1.1.1.card.4 -DefinedType -AdminStatus
```

If successful, the command returns the following text:

```
/opt/ProvServ/bin/cvget Switch.1.1.1.1.card.4 -DefinedType
1PortAtmDs3Uni
-AdminStatus Up
```

The following **cvget** command retrieves the location of a switch:

```
/opt/ProvServ/bin/cvget Switch.152.148.50.2 -Location
```

If successful, the command returns the following text:

```
/opt/ProvServ/bin/cvget Switch.152.148.50.2
-Location “XYZ Corporation”
```

cvgetdiag

Purpose

Retrieves the values of specific diagnostic information from the Provisioning Server.

Command Syntax

cvgetdiag *object-name* {*-attribute-name* } **. . .**

Parameters

object-name specifies the object whose attributes are to be retrieved. The object is specified by its object ID, based on the containment hierarchy (for information, see "Managed Objects" on page 1-12).

-attribute-name specifies an attribute to be retrieved. The attribute is specified by its argument name. Specify only attribute names with no values.

Specify only those attributes appropriate for the object type.

Notes

For a list of object types that you can use with this command, see Table 1-4 on page 1-39.

If **cvgetdiag** is successful, it prints the command name followed by the arguments that the Provisioning Server returns. You can use this output to verify that the arguments are the same as those specified in the original request.

Examples

The following **cvgetdiag** command retrieves LoopbackStatus diagnostic information from the Provisioning Server:

```
/opt/ProvServ/bin/cvgetdiag
Switch.1.1.1.1.card.4.pport.1 -LoopbackStatus
```

cvgetoperinfo

Purpose

Retrieves the values of specific real time operational information from the switch.

Command Syntax

cvgetoperinfo *object-name* {*-attribute-name* } **. . .**

Parameters

object-name specifies the object whose attributes are to be retrieved. The object is specified by its object ID, based on the containment hierarchy (for information, see "Managed Objects" on page 1-12).

-attribute-name specifies an attribute to be retrieved. The attribute is specified by its argument name. Specify only attribute names with no values.

Specify only those attributes appropriate for the object type.

Notes

For a list of object types that you can use with this command, see Table 1-4 on page 1-39.

If **cvgetoperinfo** is successful, it prints the command name followed by the arguments that the Provisioning Server returns. You can use this output to verify that the arguments are the same as those specified in the original request.

Examples

The following **cvgetoperinfo** command retrieves real time PvcDelay information from the Provisioning Server:

```
/opt/ProvServ/bin/cvgetoperinfo
Switch.1.1.1.1.card.4.pport.1.lport.1.dlci.100 -PvcDelay
```

cvhelp

Purpose

Provides usage help for the CLI.

Command Syntax

cvhelp { *object-type -attribute-name* }

Parameters

object-type specifies the object type (such as LPort, circuit, etc.) for which you want to print supported attributes or enumerated attribute values.

-attribute-name specifies an enumerated attribute for which you want to print supported enumerated values printed.

Notes

Issue **cvhelp** without arguments to print a command usage statement for each of the CLI commands.

Issue **cvhelp** with the *object-type* argument to print the attribute IDs and attribute types (such as INTEGER, STRING, and so on) that are supported for the specified object.

Issue **cvhelp** with the *object-type* and *-attribute-name* arguments to print the enumerated attribute values that are supported for the specified attribute and object.

Examples

The following **cvhelp** command prints a usage statement for each of the CLI commands:

```
/opt/ProvServ/bin/cvhelp
```

The following **cvhelp** command prints a list of all attributes supported for cards:

```
/opt/ProvServ/bin/cvhelp card
```

The following **cvhelp** command prints a list of all enumerated attribute values supported for the enumerated attribute CVA_LPortSmdsType belonging to the object LPort:

```
/opt/ProvServ/bin/cvhelp lport -smdstype
```

cvlistallcontained

Purpose

Queries the database for a list of objects of any type that are immediate children of a specified object.

Command Syntax

cvlistallcontained *object-name*

Parameters

object-name specifies the parent object that represents the immediate parent of the contained objects (such as a PPort that is a parent of multiple LPorts). The object is specified by its object ID, based on the containment hierarchy (for information, see "Managed Objects" on page 1-12).

Notes

You can issue the function on either:

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

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

Table 3-1 lists the valid parent and child object types you can specify with this command.

Table 3-1. Valid Parent and Child Object Types

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

Table 3-1. 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 4-port OC-3/STM-1 cards only) Channel ChanPerformanceMonitor LPort PerformanceMonitor PFdl (8-port ATM T1 card only) PPortTca TrafficShaper
ServiceName	Circuit
SmdsGroupScreen	SmdsAlienGroupAddress SmdsSwitchGroupAddress
SmdsIndividualScreen	SmdsAlienIndividualAddress SmdsLocalIndividualAddress

Table 3-1. 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

If **cvlistallcontained** is successful, it prints the command name followed by the child objects that the Provisioning Server returns. It prints out one line for each child object. The command does not print any attributes for the listed objects.

Example

The following **cvlistallcontained** command lists all immediate children of a switch:

```
cvlistallcontained switch.1.1.1.1
```

If successful, the command returns the following text:

```
cvlistallcontained Switch.1.1.1.1.card.1
cvlistallcontained Switch.1.1.1.1.card.2
cvlistallcontained Switch.1.1.1.1.card.3
cvlistallcontained Switch.1.1.1.1.card.4
cvlistallcontained Switch.1.1.1.1.card.5
cvlistallcontained Switch.1.1.1.1.card.6
cvlistallcontained Switch.1.1.1.1.card.7
cvlistallcontained Switch.1.1.1.1.card.8
cvlistallcontained Switch.1.1.1.1.SwitchGroupAddress.8889998889
cvlistallcontained Switch.1.1.1.1.AddressPrefix.123456
cvlistallcontained Switch.1.1.1.1.AddressPrefix.222333
cvlistallcontained Switch.1.1.1.1.AddressPrefix.890890
cvlistallcontained Switch.1.1.1.1.AddressPrefix.999000
cvlistallcontained
Switch.1.1.1.1.AlienIndividualAddress.8889998887
cvlistallcontained Switch.1.1.1.1.AlienGroupAddress.0009998887
```

cvlistcontained

Purpose

Queries the database for a list of objects of a specified type that are children of a specified object. The children can be positioned anywhere in the containment hierarchy of the root object.

Command Syntax

cvlistcontained *object-name object-type* {*-attribute-name* } **. . .**

Parameters

object-name 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). The object is specified by its object ID, based on the containment hierarchy (for information, see "Managed Objects" on page 1-12).

object-type specifies the enumerated value that specifies the type of the objects to be retrieved.

-attribute-name specifies an attribute to be retrieved for the object. The attribute is specified by its argument name. Specify only attribute names with no values. You can specify up to 40 attributes.

Specify only those attributes appropriate for the object type.

If you want all attributes to be retrieved, omit the *-attribute-name* argument. The command returns all readable attributes for the child objects.

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 command.

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
LPort	AssignedSvcSecScn Circuit PMPCktRoot PMPSpvcRoot SmdsGroupScreen SmdsIndividualScreen SmdsLocalIndividualAddress Spvc SvcAddress SvcConfig SvcNetworkId SvcPrefix SvcSecScnActParam SvcUserPart VpciTable Trunk

Table 3-2. 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 (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
SmdsGroupScreen	SmdsAlienGroupAddress SmdsSwitchGroupAddress
SmdsIndividualScreen	SmdsAlienIndividualAddress SmdsLocalIndividualAddress
SmdsNetwideGroupAddress	SmdsLocalIndividualAddress SmdsSwitchGroupAddress
SvcCUG	SvcCUGMbr
SvcCUGMbrRule	SvcCUGMbr

Table 3-2. 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

If **cvlistcontained** is successful, it prints the command name followed by the child objects that the Provisioning Server returns. It prints one line for each child object and includes attributes and values.

Example

The following **cvlistcontained** command lists all LPorts on a given switch by their Names:

```
cvlistcontained switch.1.1.1.2 lport -Name
```

If successful, the command returns the following text:

```
cvlistcontained Switch.1.1.1.2.card.4.pport.2.lport.1 -Name  lport1
cvlistcontained Switch.1.1.1.2.card.4.pport.1.lport.1 -Name  lport2
cvlistcontained Switch.1.1.1.2.card.4.pport.3.lport.1 -Name  lport3
```

cvmodify

Purpose

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

Command Syntax

cvmodify *object-name* {*-attribute-name value* } **. . .**

Parameters

object-name specifies the object to be modified. The object is specified by its object ID, based on the containment hierarchy (for information, see "Managed Objects" on page 1-12).

-attribute-name value specifies an attribute and its value to be modified. The attribute is specified by its argument name. The value uses a data type appropriate for the argument.

Specify only those attributes and values appropriate for the object type. You can specify any attribute except those with either the Read-Only or Create-Only access restriction.

Notes

For a list of object types that you can use with this command, see Table 1-4 on page 1-39.

You can use this command to create a card or PPort. The NavisCore database automatically populates each switch with cards of type "empty". Use **cvmodify** to change the card's type from "empty" to the specified type. Likewise, once a card has been configured, NavisCore automatically populates the card with all necessary Physical Ports. Use **cvmodify** 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 **cvmodify** to change the channel specifications.

You can use this command to remove a card. Use **cvmodify** to change the card's type to "empty".

If **cvmodify** is successful, it prints the command name followed by the arguments that the Provisioning Server returns. You can use this output to verify that the arguments are the same as those specified in the original request.

Example

The following **cvmodify** command creates a card by specifying its type and administrative status:

```
/opt/ProvServ/bin/cvmodify Switch.1.1.1.1.card.4 -DefinedType
1PortAtmDs3Uni -AdminStatus Up
```

If successful, the command returns the following text:

```
/opt/ProvServ/bin/cvmodify Switch.1.1.1.1.card.4
-DefinedType 1PortAtmDs3Uni
-AdminStatus Up
```

cvstartdiag

Purpose

Starts diagnostics on an object in the network.

Command Syntax

cvstartdiag *object-name* {*-attribute-name* } **. . .**

Parameters

object-name specifies the object whose attributes are to be retrieved. The object is specified by its object ID, based on the containment hierarchy (for information, see "Managed Objects" on page 1-12).

-attribute-name specifies an attribute to be retrieved. The attribute is specified by its argument name. The value uses a data type appropriate for the argument.

Specify only those attributes appropriate for the object type.

Notes

For a list of object types that you can use with this command, see Table 1-4 on page 1-39.

If **cvstartdiag** is successful, it prints the command name followed by the arguments that the Provisioning Server returns. You can use this output to verify that the arguments are the same as those specified in the original request.

Examples

The following **cvstartdiag** command starts diagnostics:

```
/opt/ProvServ/bin/cvstartdiag
Switch.1.1.1.1.card.4.pport.1 -TestType internal
```

cvstopdiag

Purpose

Stops diagnostics on an object in the network.

Command Syntax

cvstopdiag *object-name*

Parameters

object-name specifies the object whose attributes are to be retrieved. The object is specified by its object ID, based on the containment hierarchy (for information, see "Managed Objects" on page 1-12).

Notes

For a list of object types that you can use with this command, see Table 1-4 on page 1-39.

If **cvstopdiag** is successful, it prints the command name followed by the arguments that the Provisioning Server returns. You can use this output to verify that the arguments are the same as those specified in the original request.

Examples

The following **cvstopdiag** command stops diagnostics:

```
/opt/ProvServ/bin/cvstopdiag
Switch.1.1.1.1.card.4.pport.1
```

cvupdatediag

Purpose

Modifies diagnostic parameters on the switch for an object being diagnosed on the network.

Command Syntax

cvupdatediag *object-name* {*-attribute-name* } **. . .**

Parameters

object-name specifies the object whose attributes are to be retrieved. The object is specified by its object ID, based on the containment hierarchy (for information, see "Managed Objects" on page 1-12).

-attribute-name specifies an attribute fand its value to be used for setting up the diagnostics.. The attribute is specified by its argument name. The value uses a data type appropriate for the argument.

Specify only those attributes appropriate for the object type.

Notes

For a list of object types that you can use with this command, see Table 1-4 on page 1-39.

If **cvupdatediag** is successful, it prints the command name followed by the arguments that the Provisioning Server returns. You can use this output to verify that the arguments are the same as those specified in the original request.

Examples

The following **cvupdatediag** command modifies diagnostic parameters on the switch:

```
/opt/ProvServ/bin/cvupdatediag
Switch.1.1.1.1.card.4.pport.1 -TestType clearcounter
```

CLI Examples

This section provides usage examples of each of the managed objects. Use these examples as guidelines for syntax and usage.

Sample CLI Format

Conventions used in the samples are as follows:

<ip_address> — Represents an IP address, such as 130.2.20.1.

<network_no> — Represents a network number.

<id> — Represents any numeric number representation, such as card number, PPort number, LPort number, channel number, DLCI number, VPI number, VCI number, PMPSpvcLeaf number, country code number, and so on. There is no relationship among the values for these numbers.

<name> — Represents a name string, such as the customer name string, Traffic Descriptor name string, VPN name string, switch name, and so on. If the string name contains a special character (such as a period or a blank character), enclose the entire string within /" characters. For example:

/"my switch/"

<svc_string> — Represents an address string that conforms to the convention for SVC addresses.

<svccug_string> — Represents an SVC CUG string.

<rule_string> — Represents an SVC CUG member rule string.

<peer_group_string> — Represents an Peer Group string.

{-Attribute value}* — Represents the applicable attribute-value pair. An asterisk (*) indicates that you can specify multiple attribute-value pairs.

CVT_APS

There is no identifier for APS.

cvlistcontained switch.<ip_address>.card.<id>.pport.<id> aps

cvget switch.<ip_address>.card.<id>.pport.<id>.aps {-Attribute}*

cvmodify switch.<ip_address>.card.<id>.pport.<id>.aps {-Attribute value}*

CVT_AssignedSvcSecScn

cvlistcontained switch.<ip_address>.card.<id>.pport.<id>.lport.<id> assignedsvcsecscn

CVT_Card

cvlistcontained switch.<ip_address> card

cvmodify switch.<ip_address>.card.<id> {-Attribute value}*

CVT_CardTca

cvmodify switch.<ip_address>.card.<id>.cardtca {-Attribute value}*

CVT_Channel

cvget switch.<ip_address>.card.<id>.pport.<id>.channel.<id> {-Attribute }*

CVT_Circuit

Circuits are always identified by their endpoints. An endpoint can be an LPort or a ServiceName; the object ID representation differs accordingly.

In the case of LPorts, endpoints are represented differently according to different service types for the containing LPort. For Frame Relay, an endpoint is identified by DLCI number; for ATM, an endpoint is identified by the VPI, VCI pair. Specify the first endpoint as the main object identifier in the CLI command. Specify the second endpoint as an mandatory attribute to the first endpoint (using “-Endpoint2”).

In the case of ServiceName, the endpoint is identified by the network number, the name of the ServiceName binding, and the VPI/VCI pair or DLCI number (depending on endpoint type). As with LPorts, the second endpoint is represented as an mandatory attribute to the first endpoint using “-Endpoint2.”

On a GX 550 switch, PPort IDs are fixed on a subcard. If you are provisioning this switch model, be sure to specify the fixed PPort ID. See “GX 550 Support” in *Software Release Notice for NavisXtend Provisioning Server* included with this release for information about creating Circuits on the GX 550 switch.

ServiceName Endpoints

cvadd network.<network_no>.servicename.<name>.vpi.<id>.vci.<id> {-Attribute value}*

cvadd network.<network_no>.servicename.<name>.dlci.<id> {-Attribute value}*

LPort Endpoints

ATM - ATM circuit:

cvadd switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.vpi.<id>.vci.<id> {-Attribute value}*
switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.vpi.<id>.vci.<id>

ATM - Frame Relay circuit:

cvadd switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.vpi.<id>.vci.<id> {-Attribute value}*
switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.dlci<id>

```
cvadd Switch.100.100.100.5.card.13.pport.5.lport.1.vpi.4.vci.76
   -Name fr_to_atm -FwdRateEnfScheme Simple -RevRateEnfScheme Jump
   -FwdZeroCIR Off -RevZeroCIR Off -FwdQOSClass VBRNonRealTime
   -FwdTrafficDescType PcrClp01ScrClp0MbsClp0Tag
   -RevQOSClass VBRRealTime -NdcEnable1 On
   -TrafficMgmtCtdStatus Enabled
   -Alias fr_to_atm -FwdParam1 100 -FwdParam2 100 -FwdParam3 100
   -Cir2 128000 -Bc2 128000 -Be2 64000 -Priority 1
   -RevPriority 1 -GracefulDiscard On -RevGracefulDiscard On -RevDeltaBc 1024
   -RevDeltaBe 2048 -AdminStatus Up -Loopback2 Normal -RerouteBalance Enabled
   -Endpoint2
Switch.100.100.100.8.card.3.pport.1.channel.2.lport.1.dlci.57
   -BandwidthPriority 0 -BumpingPriority 0 -FwdFcpDiscard CLP1
-RevFcpDiscard CLP1
   -UpcFunction Enabled -CdvTolerance 600 -OamAlarms Enabled
-TranslationType 1483and1490
   -CLP fr_de -DE atm_clp -RevRedFramePercent 100 -VpnName Public
   -CustomerName Public -PrivNetOverflow Public -Clp0CellThresh1 150135
   -Clp1CellThresh1 150135 -AcctChrgPartyId1 35 -AcctUsageMeasure1 Enabled
   -AcctPvcControl1 Enabled -FrPvcParamRecording2 Disabled
-FrAcctPvcControl2 Enabled
   -FrAcctChrgPartyId2 35 -FrAcctUsageMeasure2 All -TrafficMgmtCtd 11
   -FrToAtmEFCI Fr-Fecn -IsMgmtCkt True -AdminCost 10
```

Frame Relay - Frame Relay circuit:

cvadd switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.dlci.<id> {-Attribute value}*
switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.dlci<id>

cvlistcontained switch.<ip_address>.card.<id>.pport.<id>.lport.<id> circuit

```
cvadd Switch.100.100.100.8.card.3.pport.1.channel.2.lport.1.dlci.456
   -Name  fr_to_fr   -FwdRateEnfScheme Simple  -RevRateEnfScheme Jump
   -FwdZeroCIR On  -RevZeroCIR Off  -FwdQOSClass UBR
   -RevQOSClass VBRNonRealTime -TrafficMgmtCtdStatus Enabled -Alias  fr_to_fr
   -Cir2 256000  -Bc2 256000  -Be2 16000  -RevPriority 1
   -GracefulDiscard On  -RevGracefulDiscard On  -RevDeltaBc 65528
   -RevDeltaBe 65528  -AdminStatus Up  -Loopback1 Normal
   -Loopback2 Normal  -RerouteBalance Enabled
   -Endpoint2
Switch.100.100.100.8.card.3.pport.1.channel.2.lport.1.dlci.475
   -BandwidthPriority 15  -BumpingPriority 7  -FwdFcpDiscard EPD
   -RevFcpDiscard CLP1  -RevRedFramePercent 75  -VpnName  Public
```

-CustomerName Public -PrivNetOverflow Public -FrPvcParamRecording1 Disabled
-FrPvcParamRecording2 Disabled -FrAcctPvcControl1 Enabled -FrAcctPvcControl2 Enabled
-FrAcctChrgPartyId1 45 -FrAcctChrgPartyId2 45 -FrAcctUsageMeasure1 FramesAndDeBytes
-FrAcctUsageMeasure2 BytesAndDeBytes -TrafficMgmtCtd 10 -IsMgmtCkt False
-AdminCost 100

cvadd Switch.100.100.100.5.card.13.pport.5.lport.1.vpi.3.vci.234
-Name atm_to_atm -FwdQOSClass CBR -FwdTrafficDescType PcrClp0PcrClp01Tag
-RevQOSClass ABR -RevTrafficDescType PcrClp0McrClp0 -NdcEnable1 On
-NdcEnable2 On -TrafficMgmtCtdStatus Disabled -TrafficMgmtFwdCdvStatus Enabled
-TrafficMgmtFwdClrStatus Enabled -Alias atm_to_atm -FwdParam1 100
-FwdParam2 100 -RevParam1 100 -RevParam2 100 -RevPriority 1
-AdminStatus Up -RerouteBalance Enabled
-Endpoint2 Switch.100.100.100.5.card.13.pport.5.lport.1.vpi.7.vci.432
-BandwidthPriority 0 -BumpingPriority 0 -FwdFcpDiscard CLP1
-RevFcpDiscard CLP1 -UpcFunction Enabled -CdvTolerance 600
-OamAlarms Enabled -VpnName Public -CustomerName Public
-PrivNetOverflow Public -Clp0CellThresh1 300270 -Clp1CellThresh1 300270
-Clp0CellThresh2 150135 -Clp1CellThresh2 300270 -AcctChrgPartyId1 56
-AcctUsageMeasure1 Egress -AcctPvcControl1 Disabled -AcctChrgPartyId2 56
-AcctUsageMeasure2 Ingress -AcctPvcControl2 Study -TrafficMgmtFwdCdv 20
-TrafficMgmtFwdClr 5 -IsMgmtCkt True -FwdFrameDiscardStatus Enabled
-RevFrameDiscardStatus Enabled -AdminCost 0

CVT_Customer

cvlistcontained network.< ip_address> customer

cvget network.<ip_address>.customer.<name> {-Attribute }*

CVT_DefinedPath

cvget switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.dlci<id>.definedpath {-Attribute }*

cvmodify switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.vpi<id>.vci<id>.definedpath {-Attribute }*

CVT_LPort

Normal LPort type:

cvadd switch.<ip_address>.card.<id>.pport.<id>.lport.<id> {-Attribute value}*

cvmodify switch.<ip_address>.card.<id>.pport.<id>.lport.<id> {-Attribute value}*

cvget switch.<ip_address>.card.<id>.pport.<id>.lport.<id> {-Attribute }*

ATM Virtual UNI LPort type:

The LPort number is generated automatically from the Start VPI number and the LPort interface number. Therefore, during creation, you do not need to provide an LPort number. To retrieve information for the LPort, you must specify its LPort number. To obtain this number, use **cvlistcontained**.

cvadd switch.<ip_address>.card.<id>.pport.<id>.startvpi.<id> {-Attribute value}*

cvmodify switch.<ip_address>.card.<id>.pport.<id>.startvpi.<id> {-Attribute value}*

cvget switch.<ip_address>.card.<id>.pport.<id>.startvpi.<id> {-Attribute }*

cvlistcontained switch.<ip_address>.card.<id>.pport.<id> lport

ATM Network Interworking for Frame Relay NNI LPort type:

The LPort number is identified by VPI/VCI pair.

cvadd switch.<ip_address>.card.<id>.pport.<id>.vpi.<id>.vci.<id> {-Attribute value}*

cvmodify switch.<ip_address>.card.<id>.pport.<id>.vpi.<id>.vci.<id> {-Attribute value}*

cvget switch.<ip_address>.card.<id>.pport.<id>.vpi.<id>.vci.<id> {-Attribute }*

cvlistcontained switch.<ip_address>.card.<id>.pport.<id> lport

MLFRBundle LPort type:

The LPort is identified by card, not by PPort.

cvadd switch.<ip_address>.card.<id>.lport.<id> {-Attribute value}*

cvmodify switch.<ip_address>.card.<id>.lport.<id> {-Attribute value}*

The following command lists all the MLFRMember LPorts bound to the specified MLFRBundle LPort:

cvlistcontained switch.<ip_address>.card.<id>.lport.<id> lport

MLFRMember LPort type:

The LPort is identified by PPort.

cvadd switch.<ip_address>.card.<id>.pport.<id>.lport.<id> {-Attribute value}*

cvmodify switch.<ip_address>.card.<id>.pport.<id>.lport.<id> {-Attribute value}*

The following command binds a member to a bundle LPort, where both LPort are on the same PPort of the same card:

cvaddmember switch.<ip_address>.card.<id>.lport.<id>
switch.<ip_address>.card.<id>.pport.<id>.lport.<id>

The following command unbinds a member from a bundle LPort:

cvdeletemember switch.<ip_address>.card.<id>.lport.<id>
switch.<ip_address>.card.<id>.pport.<id>.lport.<id>

On a GX 550 switch, PPort IDs are fixed on a subcard. If you are provisioning this switch model, be sure to specify the fixed PPort ID. See “GX 550 Support” in *Software Release Notice for NavisXtend Provisioning Server* included with this release for information about creating LPorts on the GX 550 switch.

CVT_NetCac

There is no identifier for NetCac.

cvlistcontained network.<ip_address> netcac

cvmodify network.<ip_address>.netcac {-Attribute value}*

CVT_PerformanceMonitor

There is no identifier for PerformanceMonitor.

cvlistcontained switch.<ip_address>.card.<id>.pport.<id> pm

cvget switch.<ip_address>.card.<id>.pport.<id>.pm {-Attribute }*

cvmodify switch.<ip_address>.card.<id>.pport.<id>.pm {-Attribute value}*

CVT_PFdl

There is no identifier for PFdl.

cvlistcontained switch.<ip_address>.card.<id>.pport.<id> fdl

cvget switch.<ip_address>.card.<id>.pport.<id>.fdl {-Attribute }*

cvmodify switch.<ip_address>.card.<id>.pport.<id>.fdl {-Attribute value}*

CVT_PMPCkt

A PMP circuit leaf can be added only when a PMPCktRoot exists. The circuit type of a leaf must be the same as that of the root. For example, if the PMPCktRoot has been created without specifying the VCI value, all the leaves to be added to that particular root should not have their VCI value specified.

cvadd switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.pmpcktleaf.vpi.<id>.vci.<id>
{-Attribute value}*

cvlistcontained switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.pmpcktroot.vpi.<id>.vci.<id>
pmpcktleaf

CVT_PMPCktRoot

For VCC circuit type:

cvadd switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.pmpcktroot.vpi.<id>.vci.<id> {-Attribute value}*

For VPC circuit type:

cvadd switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.pmpcktroot.vpi.<id> {-Attribute value}*

cvlistcontained switch.<ip_address>.card.<id>.pport.<id>.lport.<id> pmpcktroot

CVT_PMPSpvcLeaf

A PMPSpvc circuit leaf can be added only when a PMPSpvcRoot exists.

For PMPSpvcLeaf objects, specify the Root parent as part of the object ID representation. For example:

cvadd switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.pmpspvcroot.vpi.<id>[.vci.<id>].pmpspvcleaf.<id> {-Attribute value}*

You no longer need to specify the Root object as one of the attributes.

You must specify the correct instance number when you perform a **cvadd**, **cvget**, **cvmodify**, or **cvdelete**. To retrieve the correct instance number from the database, use the attribute CVA_PMPSpvcRootNextAvailableLeafNo.

cvlistcontained switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.pmpspvcroot.vpi.<id>.vci.<id> pmpspvcleaf

CVT_PMPSpvcRoot

For VCC circuit type:

cvadd switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.pmpspvcroot.vpi.<id>.vci.<id> {-Attribute value}*

For VPC circuit type:

cvadd switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.pmpspvcroot.vpi.<id> {-Attribute value}*

cvlistcontained switch.<ip_address>.card.<id> pmpspvcroot

CVT_PnniNode

cvadd switch.<ip_address>.pnninode.<peer_group_string> {-Attribute value}*

cvget switch.<ip_address>.pnninode.<peer_group_string> {-Attribute }*

cvmodify switch.<ip_address>.pnninode.<peer_group_string> {-Attribute value}*

CVT_PPort

cvlistcontained switch.<ip_address>.card.<id> pport

cvget switch.<ip_address>.card.<id>.pport.<id> {-Attribute }*

CVT_PPortTca

cvmodify switch.<ip_address>.card.<id>.pport.<id>.pporttca {-Attribute value}*

CVT_RefTimeServer

cvadd switch.<ip_address>.reftimeserver.<ip_address> {-Attribute value}*

cvmodify switch.<ip_address>.reftimeserver.<ip_address> {-Attribute value}*

cvlistcontained switch.<ip_address> reftimeserver

CVT_ServiceName

cvadd network.<ip_address>.servicename.<name> {-Attribute value}*

cvmodify network.<ip_address>.servicename.<name> {-Attribute value}*

cvlistcontained network.<ip_address> servicename

CVT_SmdsAddressPrefix

cvlistcontained switch.<ip_address> addressprefix

CVT_SmdsAlienGroupAddress

cvlistcontained switch.<ip_address> aliengroupaddress

CVT_SmdsAlienIndividualAddress

cvlistcontained switch.<ip_address> alienindividualaddress

CVT_SmdsCountryCode

cvadd network.<ip_address>.countrycode.<id> {-Attribute value}

CVT_SmdsGroupScreen

cvlistcontained switch.<ip_address>.card.<id>.pport.<id>.lport.<id> groupscreen

CVT_SmdsIndividualScreen

cvlistcontained switch.<ip_address>.card.<id>.pport.<id>.lport.<id> individualscreen

CVT_SmdsLocalIndividualAddress

cvlistcontained switch.<ip_address>.card.<id>.pport.<id>.lport.<id> localindividualaddress

CVT_SmdsNetwideGroupAddress

cvlistcontained network.<ip_address> netwidegroupaddress

CVT_SmdsSwitchGroupAddress

cvlistcontained switch. <ip_address> switchgroupaddress

CVT_Spvc

For VCC circuit type:

cvadd switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.spvc.vpi.<id>.vci.<id> {-Attribute value}*

For VPC circuit type:

cvadd switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.spvc.vpi.<id> {-Attribute value}*

cvlistcontained switch.<ip_address>.card.<id> spvc

CVT_SvcAddress

An SvcAddress is represented by a string that conforms to the convention used to specify SVC addresses. The format of the **cvadd**, **cvmodify**, **cvget**, or **cvdelete** command depends on the format of the SVC address.

cvadd switch.<ip_address>.card.<id>.pport.<id>.lport.<id>. svcaddress.<svc_string> {-Attribute value}*

cvlistcontained switch.<ip_address>.card.<id>.pport.<id>.lport.<id> svcaddress

See the CVT_LPort object description for sample formats that specify other LPort types.

CVT_SvcConfig

cvmodify switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.svcconfig {-Attribute value}*

CVT_SvcCUG

No attributes are needed for addition.

cvadd network.<ip_address>.svccug.<svccug_string>

cvlistcontained network. <ip_address> svccug

CVT_SvcCUGMbr

cvadd network.<ip_address>.svccug.<svccug_string>.svccugmbr.<rule_string> {-Attribute value}*

cvlistcontained network.<ip_address>.svccug.<svccug_string> svccugmbr

CVT_SvcCUGMbrRule

cvadd network.<ip_address>.svccugmbrrule.<rule_string> {-Attribute value}*

cvlistcontained network.<ip_address>.svccugmbrrule

CVT_SvcNetworkId

Normal LPort type:

cvadd switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.svcnetworkid.<svc_str> {-Attribute value}*

cvmodify switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.svcnetworkid.<svc_str>
{-Attribute value}*

cvget switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.svcnetworkid.<svc_str> {-Attribute }*

ATM Virtual UNI LPort type:

The LPort number is generated automatically from the Start VPI number and the LPort interface number. Therefore, during creation, you do not need to provide an LPort number. To retrieve information for the LPort, you must specify its LPort number. To obtain this number, use **cvlistcontained**.

cvadd switch.<ip_address>.card.<id>.pport.<id>.startvpi.<id>.svcnetworkid.<svc_str>
{-Attribute value}*

cvmodify switch.<ip_address>.card.<id>.pport.<id>.startvpi.<id>.svcnetworkid.<svc_str>
{-Attribute value}*

cvget switch.<ip_address>.card.<id>.pport.<id>.startvpi.<id>.svcnetworkid.<svc_str> {-Attribute }*

cvlistcontained switch.<ip_address>.card.<id>.pport.<id>.startvpi.<id>.svcnetworkid
{-Attribute value}*

See the CVT_LPort object description for sample formats that specify other LPort types.

CVT_SvcNodePrefix

An SvcNodePrefix is represented by a string that conforms to the convention used to specify SVC addresses. The format of the **cvadd**, **cvmodify**, **cvget**, or **cvdelete** command depends on the format of the SVC address.

cvadd switch.<ip_address>.svcnodeprefix.<svc_string> {-Attribute value}*

cvlistcontained switch.<ip_address> svcnodeprefix

CVT_SvcPrefix

An SvcPrefix is represented by a string that conforms to the convention used to specify SVC addresses. The format of the **cvadd**, **cvmodify**, **cvget**, or **cvdelete** command depends on the format of the SVC address.

cvadd switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.svcprefix.<svc_string> {-Attribute value}*

cvlistcontained switch.<ip_address>.card.<id>.pport.<id>.lport.<id> svcprefix

See the CVT_LPort object description for sample formats that specify other LPort types.

CVT_SvcSecScn

cvlistcontained network.<ip_address> svcsecscn

CVT_SvcSecScnActParam

cvlistcontained switch.<ip_address>.card.<id>.pport.<id>.lport.<id> svcsecscnactparam

CVT_SvcUserPart

An SvcUserPart is represented by a string that conforms to the convention used to specify SVC addresses. The format of the **cvadd**, **cvget**, or **cvdelete** command depends on the format of the SVC address.

cvadd switch.<ip_address>.card.<id>.pport.<id>.lport.<id>. svcuserpart.<svc_string>

cvlistcontained switch.<ip_address>.card.<id>.pport.<id>.lport.<id> svcuserpart

See the CVT_LPort object description for sample formats that specify other LPort types.

CVT_Switch

cvlistcontained network.<ip_address> switch

CVT_TrafficDesc

cvadd network.<ip_address>.trafficdesc.<name> {-Attribute value}*

cvlistcontained network.<ip_address> trafficdesc

CVT_TrafficShaper

cvlistcontained switch.<ip_address>.card.<id>.pport.<id> ts

cvget switch.<ip_address>.card.<id>.pport.<id>.ts.<id> {-Attribute }*

CVT_Trunk

cvget network.<ip_address>.trunkname.<name> {-Attribute }*

cvlistcontained network.<ip_address> trunk

In addition, the following command returns all the circuits configured on a trunk:

cvlistcontained network.<ip_address>.trunkname.<name> circuit

When adding a trunk, specify the LPort of each trunk endpoint:

cvadd network.<ip_address>.trunkname.<name>
-Lport1 switch.<ip_address>.card.<id>.pport.<id>.lport.<id>
-Lport2 switch.<ip_address>.card.<id>.pport.<id>.lport.<id> {-Attribute value}*

When adding a Multi-Link Frame Relay (MLFR) trunk, specify the LPort of each trunk endpoint without specifiying a PPort:

cvadd network.<ip_address>.trunkname.<name>
-Lport1 switch.<ip_address>.card.<id>.lport.<id>
-Lport2 switch.<ip_address>.card.<id>.lport.<id> {-Attribute value}*

To identify Direct Trunk LPort type (such as DirectLine Trunk, ATM Direct Trunk, and so on):

switch.<ip_address>.card.<id>.pport.<id>.lport.<id>

To identify Frame OPTimum Trunk LPort type:

switch.<ip_address>.card.<id>.pport.<id>.dlci.<id>

To identify ATM OPTimum Trunk LPort type (such as ATM OPTimum Cell Trunk, ATM OPTimum Frame Trunk, and so on):

switch.<ip_address>.card.<id>.pport.<id>.vpi.<id>.vci.<id>

To identify MLFR Trunk LPort type:

switch.<ip_address>.card.<id>.lport.<id>

CVT_VPCITable

cvadd switch.<ip_address>.card.<id>.pport.<id>.lport.<id>.vpcitable.<id> {-Attribute value}*

cvlistcontained switch.<ip_address>.card.<id>.pport.<id>.lport.<id> vpcitable

CVT_VPN

cvadd network.<ip_address>.vpn.<name>

4

Using the SNMP MIB

This chapter describes how to use the SNMP MIB to access the Provisioning Server. To understand the Provisioning Server object hierarchy, first read Chapter 1.

About the Enterprise-specific MIB

The enterprise-specific MIB interface provides SNMP access to the Provisioning Server. Use the Provisioning Server MIB to provision via SNMP instead of using a C or C++ program or the CLI.

The Provisioning Server MIB is different than the Ascend Enterprise MIB.

The Provisioning Server SNMP agent supports the SNMPv1 and SNMPv2c protocols. The following SNMP operations are supported:

- **get**
- **get-next**
- **set** (used for creating, modifying, and deleting)

The Provisioning Server does not generate or process SNMP traps, nor does it support the **getBulkRequest** and **InformRequest** SNMPv2 PDU types.

The Provisioning Server MIB is defined according to Structure of Management Information version 2 (SMIv2). You can view the MIB with an SMIv2-compliant MIB browser.

To compile the MIB, use an SMIv2-compliant compiler.

The MIB is defined in the file provserv.mib, which is installed in the directory /opt/ProvServ/snmp_mibs.

If you install the Provisioning Server on a separate machine from NavisCore, and you want to use the HP OpenView MIB browser to view the MIB, perform the following steps:

1. File transfer provserv.mib from the Provisioning Server machine to the directory opt/CascadeView/snmp_mibs on the NavisCore machine.
2. Load the MIB file from the NavisCore machine.

For a listing of the variables in the Provisioning Server MIB, see the *NavisXtend Provisioning Server Enterprise MIB Definitions*.

Community Strings

The Provisioning Server implements an SNMP agent as a separate entity within the server to service MIB interface requests. The community name provides a mechanism for authentication and access-control at the agent. The Provisioning Server supports two community names, one for Read-Only operations and another for Read-Write operations.

The community strings are defined using the environment variables CV_READONLY_COMMUNITY_STRING (default value ‘public’) and CV_READWRITE_COMMUNITY_STRING (default value ‘ascend’). If the environment variables are defined in the script start-server.sh, the specified strings take precedence. If they are not set in the script or if the server shell environment does not define the variables, the server assumes the default values.

If the community name is not valid when you issue an snmp_set request, the request exceeds the time-out period and fails. You can access the Command Error Table in the MIB to see if the source of the problem is an invalid community name. Specify the Read-Only community name when you access the table, as that community name is used for validation purposes.

When you make an snmp_get request, specify either the Read-Only or the Read-Write community name. If you use a different community name and you encounter an error, the error is not propagated to the Command Error Table.

For details on setting environment variables to configure how the Provisioning Server handles MIB requests, see “Setting Environment Variables” on page 2-11.

MIB Structure

The Provisioning Server MIB defines objects that a client can configure or read. The MIB is organized into logical groups by object (node, card, LPort, PPort, and so on). Each group contains table entries that map to the attributes of the API.

The various groups of the MIB are placed under the Provisioning Server object identifier (OID):

```
1.3.6.1.4.1.277.9.1
```

where the last term in the OID represents the version number of the MIB.

Each group can have one or more tables and/or scalar objects. The tables are two-dimensional, with each column representing an attribute and each row representing an object instance on a switch. Because a column contains rows for all possible object instances, many of which may not actually use that attribute, a table can contain *holes*. Holes are non-applicable elements of the matrix. For example, in the PPort table, the column that contains the attribute pportChannelIsInUse is sparse because it contains values only for PPort instances present on channelized cards.

Row instances in a table are uniquely identified by Index information. The Index represents the information you need to provide when issuing a command on a particular object. For example, to configure an LPort, you need to specify the IP address of the switch that contains the LPort and the ifIndex.

Segmented Information in Multiple Tables

LPorts are always identified by specifying the IP address of the switch that contains the LPort and the LPort's ifIndex. Because LPorts are complex objects, additional information (such as LPortId, DLCI number, or VPI/VCI pair) is required to obtain an ifIndex.

The MIB uses Translation Tables to convert the information required for a specific LPort type into the ifIndex value. The Translation Table provides a unique key to access a specific row in the Configuration Table (the table that contains the configuration attributes of the LPort). Table 4-1 lists the information required to create each type of LPort and which specific Translation Table to use.

Table 4-1. Information Required for Creating Specific LPorts

LPort Type	Information Required	Table to Use
ATM Direct Trunk ATM UNI DCE/DTE Direct Line Trunk Encapsulation FRAD FR NNI FR UNI DCE/DTE PPP to1490 Encapsulation SMDS DXI/SNI DTE/DCE SMDS OPT Trunk SMDS SSI DTE	Switch Card PPort LPort Id	lportIdIndexTransTable lportIdChannelIndexTransTable (for LPorts on the channelized DS3 card)

Table 4-1. Information Required for Creating Specific LPorts (Continued)

LPort Type	Information Required	Table to Use
FR OPT PVC Trunk	Switch Card PPort DLCI	dlciIndexTransTable dlciChannelIndexTransTable (for LPorts on the channelized DS3 card)
Virtual UNI DCE/DTE	Switch Card PPort VPI start number	vpiStartIndexTransTable
ATM OPT Cell Trunk	Switch Card PPort VPI (1-15)/ VCI 0	vpiVciIndexTransTable vpiVciChannelIndexTransTable (for LPorts on the channelized DS3 card)
ATM Network Interworking for FR NNI ATM OPT Frame Trunk	Switch Card PPort VPI (0-15)/ VCI (32-255)	vpiVciIndexTransTable vpiVciChannelIndexTransTable (for LPorts on the channelized DS3 card)

Circuits are segmented into several categories of tables, based on technology. You access various tables to configure circuit endpoints and configure the cross-connections between endpoints. Table 4-2 lists the information required to create each type of circuit endpoint and which specific endpoint table to use. The type of endpoint table to use depends on the type of services offered on the card on which the endpoint is created. Note that the ServiceName can be used with either or both endpoints.

Once the endpoints are created, use the CircuitCrossConnectTable.

Table 4-2. Information Required for Creating Specific Circuits

Circuit Type	Card Type	Information Required	Table to Use
FR-FR	All cards on B-STDX and STDX except: 1-port ATM IWU OC3 1-port ATM-CS/DS3 6-port DS3 Frame Relay card on CBX 500.	{switchIdIndex lportIfIndex DlciIdIndex}	frCircuitEndpointTable
FR-FR (with either endpoint using ServiceName)	All cards on B-STDX and STDX except: 1-port ATM IWU OC3 1-port ATM-CS/DS3 6-port DS3 Frame Relay card on CBX 500.	{switchIdIndex lportIfIndex DlciIdIndex } (for non-ServiceName based endpoint) {networkIdIndex networkServiceName Index dlciIdIndex} (for ServiceName based endpoint)	frCircuitEndpointTable (for non-ServiceName endpoint) frCircuitServiceNameEndpoint Table (for ServiceName based endpoint)
ATM-ATM	All cards on CBX/GX (except 6-port DS3 Frame Relay) and 1-port ATM IWU OC3 and 1-port ATM CS/DS3 card on B-STDX	{switchIdIndex lportIfIndex vpiIndex vciIndex}	atmCircuitEndpointTable
ATM-ATM (with either endpoint using ServiceName)	All cards on CBX/GX (except 6-port DS3 Frame Relay) and 1-port ATM IWU OC3 and 1-port ATM CS/DS3 card on B-STDX	{switchIdIndex lportIfIndex vpiIndex vciIndex} (for non-ServiceName based endpoint) {networkIdIndex networkServiceName Index vpiIndex vciIndex} (for ServiceName based endpoint)	atmCircuitEndpointTable (for non-ServiceName endpoint) atmCircuitServiceNameEndpoint Table for ServiceName based endpoint)

Table 4-2. Information Required for Creating Specific Circuits (Continued)

Circuit Type	Card Type	Information Required	Table to Use
ATM-ATM	One endpoint on any card that is one of the following (category A): All cards on CBX 500. 1-port ATM IWU OC3 card and 1- port ATM CS/DS3 card on the B-STDX. The other endpoint on any card that is one of the following (category B): All cards on B-STDX except 1-port ATM IWU OC3 card and 1-port ATM CS/DS3 card.	For category A or category B without ServiceNames use: {switchIdIndex lportIfIndex vpiIndex vciIndex} For category A or B with ServiceNames use: {networkIdIndex networkServiceName Index vpiIndex vciIndex}	For category A without ServiceName use atmCircuitEndpointTable. For category A with ServiceNames use atmCircuitServiceNameEndpoint Table. For category B without ServiceName use interworkingCircuitEndpointTable. Use interworkingCircuitServiceName EndpointTable.
ATM-ATM	Both endpoints exist on cards that belong to category B as explained above.	For category B without ServiceNames use: {switchIdIndex lportIfIndex vpiIndex vciIndex} For category B with ServiceNames use: {networkIdIndex networkServiceName Index vpiIndex vciIndex}	Without ServiceName, use interworkingCircuitEndpointTable. For category B with ServiceNames use interworkingCircuitServiceName EndpointTable.
FR-ATM Interworking	Does not depend on the card type of the endpoints.	For the FR endpoint: {switchIdIndex lportIfIndex dlciIdIndex} For the ATM endpoint: {switchIdIndex lportIfIndex vpiIndex vciIndex}	For the ATM endpoint, use interworkingCircuitEndpointTable For the FR endpoint use frCircuitEndpointTable

Table 4-2. Information Required for Creating Specific Circuits (Continued)

Circuit Type	Card Type	Information Required	Table to Use
FR-ATM Interworking with either endpoint using ServiceName	Does not depend on the card type of the endpoints.	For the ServiceName based FR endpoint: {networkIdIndex networkServiceName Index, dlciIdIndex} For the ServiceName based ATMendpoint: {networkIdIndex networkServiceName Index vpiIndex vciIndex}	For the FR endpoint use frCircuitServiceNameEndpoint Table For the ATM endpoint use interworkingCircuitServiceName EndpointTable

By segmenting information into separate tables based on technology or specific features, the MIB improves performance of **get-next** operations because it minimizes holes in matrices.

See the Provisioning Server MIB for details on the main groups of the MIB and the indexing scheme for each group.

Row Aliasing

For objects in the MIB that have attributes dispersed in several tables, some attributes are common to multiple tables. In the tables of the LPort and circuit groups, the following attributes are common attributes:

- RowStatus (see “RowStatus Attribute” on page 4-8)
- ModifyType (see “NumRetries Attribute” on page 4-9)
- lportIfIndex (for tables in the LPort group only)
- CircuitNumber (for tables in the circuit group only)
- NumRetries (for tables in the circuit group only) (see “NumRetries Attribute” on page 4-9)

The tables containing common attributes are considered linked. Thus, an operation on a common attribute in one linked table affects the common attribute in the other linked tables. For example, for a Frame Relay UNI DCE LPort, when the RowStatus attribute is modified in one table (such as the lportIdIndexTransTable), the value of that

attribute is updated in other linked tables (such as the lportAdminTable and lportFrTable). For a Frame Relay to Frame Relay circuit, when the RowStatus attribute is modified in the frCircuitEndpointTable, the value of the attribute is updated in the linked circuitCrossConnectTable.

This feature, known as *row aliasing*, gives the user the flexibility to set an attribute in only one table rather than set it in all related tables. Using row aliasing, the Provisioning Server reflects the same value for a common attribute for the same row across linked tables. Row aliasing assures that the status of a row and its common attributes are always the same irrespective of the table.

The lportIfIndex attribute is an attribute that is not directly set by the user. It is generated when the user sets the RowStatus attribute to the createAndWait state in a translation table (such as the lportIdIndexTransTable). Once lportIfIndex is generated, the attribute is updated in the linked LPort tables lportIdIndexTransTable, lportAdminTable, and lportFrTable.

The user does not directly set the CircuitNumber attribute. It is generated when the user generates circuit endpoints. Once CircuitNumber is generated, the attribute is updated in the linked circuit tables frCircuitEndpointTable and circuitCrossConnectTable.

Column Access Specifiers

Access specifiers for a table column are specified as Read-Only, Read-Write, or Not-Accessible. Because SNMP does not support the category Create-Only, attributes with this restriction are defined as Read-Write. These attributes are usually mandatory attributes that you provide when creating an object. See the *NavisXtend Provisioning Server Object Attribute Definitions* for attributes that are Create-Only.

For most tables, the index attributes are specified as Not-Accessible. Instead of accessing these index columns directly, you use the Translation Tables to convert required information into index attributes.

Additional Table Entries

Most table entries have the attributes RowStatus and ModifyType. These attributes are used in set operations. Circuit table entries have the NumRetries attribute, which specifies retry behavior in the event of a failed attempt to add, delete, or modify a circuit.

RowStatus Attribute

The RowStatus attribute specifies the state of the table entry at a given time. Valid values are as follows:

active (1) — Entry is active, such as when it has been created and definitions have been made to it.

notInService (2) — Entry is not in service, such as when modifications are being made to it.

notReady (3) — Entry is under creation.

createAndWait (5) — Entry is being created, and is waiting for definitions to be made to it. When you set the RowStatus attribute to 5, it gets set to 3.

destroy (6) — Entry has been removed.

You must include the RowStatus attribute when you:

- Create an object
- Modify an object by specifying the attribute modifications in multiple PDUs
- Destroy an object

ModifyType Attribute

The ModifyType attribute specifies the update method, as follows:

1 sends updates to both the network component and the database. The database is updated only if the network component updates successfully.

4 sends updates to the database only.

5 sends updates to the database only and sets a flag indicating that the object is out of synchronization in the database.

By default, updates are made to both the component and the database.

You must include the ModifyType attribute when you want an update to be made to the database only. The setting applies only to the current request. Subsequent requests revert to the default setting.

NumRetries Attribute

For requests to add, delete, or modify a circuit, use the NumRetries attribute to specify the retry control.

By default, when the Provisioning Server receives a request to add, delete, or modify a circuit, the server obtains card status for both circuit endpoints:

- If both cards are up, the Provisioning Server performs the add, delete, or modify request as normal.
- If either card is down or is not reachable (for example, because of an SNMP timeout), the server retries the request for card status as many times as specified by the retry control:

- If the card becomes reachable and is up, the Provisioning Server performs the circuit provisioning request.
- Once all retries have been issued, if the card is still not reachable or is still down, the provisioning request is not performed.

The attribute has the default value 0 and the maximum value 5. 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. Specify the NumRetries attribute for each MIB request.

NumRetries is a common attribute to the following tables:

- interworkingCircuitEndpointTable
- atmCircuitEndpointTable
- frCircuitEndpointTable
- circuitCrossConnectTable
- atmCircuitBillingTable
- atmCircuitNdcTable
- interworkingCircuitServiceNameEndpointTable
- atmCircuitServiceNameEndpointTable
- frCircuitServiceNameEndpointTable

This control prevents circuits from being partially provisioned and the database from becoming out of sync with the switch. However, it can increase the time it takes to provision a circuit, depending on how many card status checks occur.

Keep in mind that this control affects the retry behavior of circuit provisioning requests only. Other retry controls specified in cascadeview.cfg (CV_SNMP_MAX_RETRIES, CV_SNMP_RETRY_INTERVAL, and CV_SNMP_REQUEST_TIMEOUT) also apply to each request. Remember to consider these other retry controls when specifying retry behavior of a circuit request.

Using the MIB

This section describes how to use the MIB to list, create, and modify a given component on the network.

Using the SNMP Commands

The Provisioning Server supports the following SNMP commands:

get — Reads a single attribute of a row in a table.

get-next — Walks the MIB (similar to performing a ListContained command in the API or CLI). The command is based on a lexicographical ordering of the complete OID for various row instances. Thus, the command walks a table by reading all row values of the first column before starting the second column.

set — Creates a new object, or modifies or deletes an existing object.

Command Error Table

The Command Error Table supplies information about any errors you encounter during snmp_set operations to create or modify objects. This information is useful for troubleshooting problems.

Entries in the table contain the following information:

- IP address of the host machine where the MIB client's request originated.
- The request ID of the request sent to the server.
- UDP port number of the client.
- Error code encountered when the server executed the command.
- Error message string.
- The OID of the attribute (column) that is in error. If several attributes are in error, only the first one is reported.
- The timestamp at which the error occurred.

If several MIB clients use the same host, it can be difficult to distinguish the various entries in the table based on IP address. To determine uniqueness, use the request ID, UDP port number, or the timestamp of the entry.

The Provisioning Server purges entries in the Command Error Table based on the value of environment variable CV_SNMP_CMDERROR_CACHE_TIMEOUT. The default setting of this variable is 6000 seconds. For more information on this environment variable, see "Controlling MIB Cache" on page 2-19.

SNMPv2 uses a richer set of error codes than SNMPv1. Because the bi-lingual agent may be responding to SNMPv1 and SNMPv2 messages, it may need to map to the appropriate error code. Table 4-4 is the table the agent currently uses to map an SNMPv2 code to an SNMPv1 code.

Table 4-3. Error Code Mapping from SNMPv2 to SNMPv1

SNMPv2	SNMPv1
noError	noError
tooBig	tooBig
genErr	genErr
wrongValue	badValue
wrongEncoding	badValue
wrongType	badValue
wrongLength	badValue
inconsistentValue	badValue
noAccess	noSuchName
notWritable	noSuchName
noCreation	noSuchName
inconsistentName	noSuchName
resourceUnavailable	genErr
commitFailed	genErr
authorizationError	noSuchName
undoFailed	genErr

MIB Cache and Database Locking

The Provisioning Server implements a MIB cache that stores data in memory for a fixed time period. The server uses the cache to optimize performance of **get-next** requests and to store data to be committed to the database during transactions involving multiple PDUs. The caching behavior varies depending on which operation is being performed. For details, see “Row Creation”, “Row Modification”, and “get-next Operations” later in this section.

The object locking behavior for MIB objects in the database differs from the locking behavior of the Provisioning Server API, CLI, or NavisCore. For these interfaces, the steps associated with locking are transparent to the user. When an object is created or modified, its parent object gets locked. The user specifies all the information needed to create or modify the object in one request. Once the request is complete, the parent gets unlocked.

By contrast, in the case of the MIB, the information needed to create or modify an object may not be available in one PDU. As a result, the locks in the database must be held for a longer time. Thus, the steps associated with locking are not transparent to the user.

If the API, CLI, or NavisCore makes modifications to an object in the database at the same time that the object is present in MIB cache during a **get** or **get-next** request, the MIB values in cache become stale.

Row Creation

When an object is created, a new row is created in the database. During a successful row creation, you perform the following steps:

1. Initiate the transaction by setting the RowStatus attribute to the createAndWait state.

   The parent object gets locked.

2. Issue one or more snmp_set requests to assign values to other attributes of the row.

   The attribute values are stored in MIB cache.

3. Complete the transaction by setting the RowStatus attribute to the active state.

   When no errors are encountered, the changes are committed to the switch and to the database, the row is flushed from MIB cache, and the lock is released.

If an error is encountered, the row remains in cache and the lock remains in effect. You can correct the error by modifying the contents of the cache (by returning to step 2). Once you have corrected the error and set the RowStatus to the active state, the row creation is completed, the row is flushed from MIB cache, and the lock is released. Note that it can take several iterations before all the errors are corrected.

If a user initiates but does not complete a transaction to create an object, the partially-created row remains in MIB cache for the amount of time specified by the environment variable CV_SNMP_LOCK_TIMEOUT. And, the parent object remains locked for the time specified by the CV_SNMP_LOCK_TIMEOUT value, preventing other users from accessing the parent object. Thus, users should make sure to complete all transactions. Once the CV_SNMP_LOCK_TIMEOUT timer expires, the partially-created row is flushed out of cache and the lock is removed.

For more information on this environment variable, see "Controlling Object Locking" on page 2-19.

Row Modification

When an object is modified, a row is modified in the database. Before modifying an object, perform an snmp_get request on the RowStatus attribute to check if another user is currently accessing the entry. If the entry is in use, retry your request later.

Row modification can be performed with or without modifying the RowStatus attribute.

PDU Modification without Modifying RowStatus

Simple modifications do not require you to set the RowStatus to the notInService state; the RowStatus remains Active through the transaction. When you do not have to modify the RowStatus attribute, the locking and unlocking of the object becomes transparent.

If you want to modify only a few attributes, you can issue one PDU containing the appropriate values for the varbinds. Or, you can issue a PDU multiple times.

To maximize MIB efficiency, you should specify all varbinds in one PDU whenever possible.

PDU Modification by Modifying RowStatus

With complex modifications involving a number of attributes, you can issue multiple PDUs containing the appropriate values for the varbinds. However, to maximize MIB efficiency, you should specify all varbinds in one PDU whenever possible.

Because of attribute dependencies, you should first set the RowStatus to the notInService state before making the modifications.

During a complex modification, you perform the following steps:

1. Issue an snmp_get request on the RowStatus attribute to make sure that no other user is currently accessing the object.
2. Initiate the transaction by setting the RowStatus attribute to the notInService state.

   The object gets locked.

3. Issue one or more snmp_set requests to assign values to other attributes of this row.

   The attribute values are stored in MIB cache.

4. Complete the transaction by setting the RowStatus attribute to the active state.

   When no errors are encountered, the changes are committed to the switch and to the database, and the lock is released.

If an error is encountered during modification, the lock remains in effect. You can correct the error by modifying the contents of the cache (by returning to step 3). Once you have corrected the error and set the RowStatus to the active state, the row creation is completed and the lock is released.

If a user initiates (but does not complete) a transaction to modify an object, the partially-modified row remains in MIB cache for the amount of time specified by the environment variable CV_SNMP_LOCK_TIMEOUT. And, the object remains locked for the time specified by the CV_SNMP_LOCK_TIMEOUT value, preventing other users from accessing the object. Thus, users should make sure to complete all transactions. Once the CV_SNMP_LOCK_TIMEOUT timer expires, the partially-created row is flushed out of cache and the lock is removed.

get-next Operations

You can perform a **get-next** request starting at any location in the MIB (including the top of the MIB), at any group of the MIB, any column of a table, or a specific column of an instance.

get-next requests are performance-intensive operations. The Provisioning Server uses MIB cache to cache objects (rows), thus optimizing performance of **get-next** requests. When the objects are initially loaded into cache from the database, the response to a **get-next** request may be slow. However, once the caching is complete, the response becomes significantly faster.

Be aware that using **get-next** operations on tables with many entries in the database may take some time to retrieve. These operations can significantly affect performance of the server. Although a **get-next** operation will not block other requests, it can slow the response to the other requests.

The Provisioning Server purges entries in MIB cache resulting from a **get-next** operation based on the value of environment variable CV_SNMP_ROWENTRY_TIMEOUT. The default setting of this variable is 900 seconds. For more information on this environment variable, see “Controlling MIB Cache” on page 2-19.

Specifying the Object Identifier

When you want to access a specific variable from a MIB group, you enter an OID that uses the following format:

```
{Provisioning Server OID}.{Group}.{Sub-group}.{Table}.{Entry}.{Column}.{Index}
```

Complex objects, such as LPorts and circuits, require a sub-group; simple objects do not.

The Provisioning Server OID is:

```
1.3.6.1.4.1.277.9.1
```

where the last term in the OID represents the version number of the MIB.

Example 1: get Command

To find out what type of card is located in a particular slot of a switch, use the following steps to determine the OID of the command you want to issue:

1. Determine the group value by locating the Card Group in the beginning of the MIB document. The following line indicates that the group value is **4**:

   ```
   card       OBJECT IDENTIFIER ::= { psMibRev2 4 }
   ```

   Cards are simple objects that do not require a sub-group name.

2. Determine the Table value by locating the Table index, **cardTable**. The line **::= { card 1 }** indicates that the Table value is **1**:

   ```
   cardTable OBJECT-TYPE
       SYNTAX SEQUENCE OF CardEntry
       MAX-ACCESS not-accessible
       STATUS current
       DESCRIPTION
              "Table representing information about all cards in the network"
       ::= { card 1}
   ```

3. Determine the Entry value by locating the Entry index, **cardEntry**. The line **::= { cardTable 1 }** indicates that the Entry value is **1**:

   ```
   cardEntry OBJECT-TYPE
       SYNTAX CardEntry
       MAX-ACCESS not-accessible
       STATUS current
       DESCRIPTION
              "Entry representing information about one card"
       INDEX  { switchIdIndex, slotIdIndex }
       ::= { cardTable 1 }
   ```

4. Determine the Column value for the MIB variable you want to access. To retrieve a card's type, you need to access the variable **cardDefinedType**. The line **::= { cardEntry 1 }** indicates that the Column value is **1**.

5. Determine the Index items by locating them in the cardEntry variable you located in step 3. The line **INDEX { switchIdIndex, slotIdIndex }** indicates the index items you need to provide to complete this command.

The **switchIdIndex** represents the IP address of the switch. The **slotIdIndex** represents the slot where the card is located. If the switch that contains the card has IP address 152.148.10.19 and the card for which you are requesting information is in slot 8, then the index is 152.148.10.19.8.

6. Enter the following command to retrieve the card type for the card (this example uses MIT SNMP Tools command syntax):

```
snmpget -h <server-machine-name> -p<server-port> -c<community-name>
1.3.6.1.4.1.277.9.1.4.1.1.1.152.148.10.19.8
```

where {Provisioning Server OID = 1.3.6.1.4.1.277.9.1}.{Group = 4}.
{Table = 1}.{Entry = 1}.{Column = 1}.{Index = 152.148.10.19.8}

The system responds by displaying the command as the full MIB tree index, 1.3.6.1.4.1.277.9.1.4.1.1.1.152.148.10.19.8, and retrieves an integer that represents the type of the card. See the cardDefinedType variable to interpret this integer.

Example 2: get-next Command

To retrieve the Admin status for all LPorts on a switch, use the following steps to determine the OID of the command you want to issue:

1. Determine the group value by locating the LPort Group in the beginning of the MIB document. The following line indicates that the group value is **6**:

```
lport      OBJECT IDENTIFIER ::= { psMibRev2 6 }
```

2. Admin status is a configuration attribute. To determine the Sub-group value, locate the LPortConfiguration table in the beginning of the MIB document. The following line indicates that the Sub-group value is **2**.

```
lportConfigurationOBJECT IDENTIFIER ::= { lport 2 }
```

3. Determine the Table value by locating the Table index, **lportAdminTable**. The line **::= { lportConfiguration 1 }** indicates that the Table value is **1**:

```
lportAdminTable OBJECT-TYPE
    SYNTAX SEQUENCE OF LportAdminEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION  "List of logical port common attribute entries."
    ::= { lportConfiguration 1 }
```

4. Determine the Entry value by locating the Entry index, **lportAdminEntry**. The line **::= { lportAdminTable 1}** indicates that the Entry value is **1**:

```
lportAdminEntry OBJECT-TYPE
    SYNTAX LportAdminEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
           "Logical Port Configuration Entry"
    INDEX  { switchIdIndex, lportIfIndex }
    ::= { lportAdminTable 1 }
```

5. Determine the Column value for the MIB variable you want to access. To retrieve the Admin status, you need to access the variable **lportAdminAdminStatus**. The line **::= { lportAdminEntry 19 }** indicates that the Column value is **19**:

```
lportAdminAdminStatus OBJECT-TYPE
    SYNTAX INTEGER {
      up(1),
      down(2),
      testing(3)
    }
    MAX-ACCESS read-write
    STATUS current
    DESCRIPTION "LPort Administrative Status. This attribute is mandatory
    for lport creation"
    ::= { lportAdminEntry 19}
```

6. Enter the following command to request Admin status of all LPort instances in the table (this example uses MIT SNMP Tools command syntax):

```
snmpget -h <server-machine-name> -p<server-port> -c<community-name>
1.3.6.1.4.1.277.9.1.6.2.1.1.19
```

   where {Provisioning Server OID = 1.3.6.1.4.1.277.9.1}.{Group = 6}.{Sub-group = 2}.{Table = 1}.{Entry = 1}.{Column = 19}

   Index items are omitted, because the request is for Admin status of *all* LPort instances in the table.

   For each LPort instance in the network, the system responds by displaying the command as the full MIB tree index, 1.3.6.1.4.1.277.9.1.6.2.1.1.19, and retrieves an integer that represents the Admin status of the LPort. If the value is 1, the Admin Status of an LPort is up; if the value is 2, the Admin Status is down.

The following examples illustrate how to use the Provisioning Server MIB to create, modify, and delete objects. Several examples involve ATM objects. You would use a similar approach to manage Frame Relay objects, except that you access different tables in the MIB. For example, to manage a Frame Relay LPort, you use the appropriate LPort Translation Table, the lportAdminTable, and the lportFrTable.

Example 3: set Command to Create an ATM LPort

To create an ATM LPort, you use the lportIdIndexTransTable to map between the card, PPort, and LPort ID and the LPort interface number. You must specify the LPort ID and request an interface number for it.

To create an ATM LPort, use the following steps:

1. Issue an snmp_set request to obtain an LPort interface number based on the LPort ID. Set the lportIdIndexTransRowStatus to the createAndWait state, specifying the switchIdIndex 1.1.1.1, slotIdIndex 7, pportIdIndex 8, and lportIdIndex 1.

   The SNMP agent processes the request and returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2).

2. Issue an snmp_get request to obtain the interface number (lportIfIndex) that will be used to create a new entry in the lportAdminTable and the lportAtmTable. Issue the request on the lportIdIndexTransIfIndex, specifying the switchIdIndex, slotIdIndex, pportIdIndex, and lportIdIndex values.

   The SNMP agent processes the request and returns an snmp_getResponse (SNMPv1) or an snmp_Response (SNMPv2) with the lportIfIndex 7.

3. Issue a series of snmp_set requests that assign values to the attributes of the LPort in both the lportAdminTable and the lportAtmTable.

   The SNMP agent processes the requests by storing the values in MIB cache. Then, the agent returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2).

4. Issue an snmp_set request to commit the new entry. Set the lportAdminRowStatus to the active state, specifying the switchIdIndex 1.1.1.1 and the lportIfIndex 7. This command automatically sets the lportIdIndexTransRowStatus to the active state.

   The SNMP agent processes the request by committing the new entry to the switch and to the NavisCore database.

   On receipt of noError messages from the switch and database, the SNMP agent returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2) to the MIB client.

Figure 4-1 shows the request-response message flow between the MIB client, the SNMP agent, and the database when adding the LPort.

Figure 4-1 uses SNMPv1 syntax. If you are using SNMPv2, the syntax used in messages from the agent to the MIB client will differ. For example, when a v1 agent sends a message beginning with snmp_setResponse, a v2 agent uses snmp_Response. When a v1 agent sends snmp_getResponse, a v2 agent sends snmp_Response.

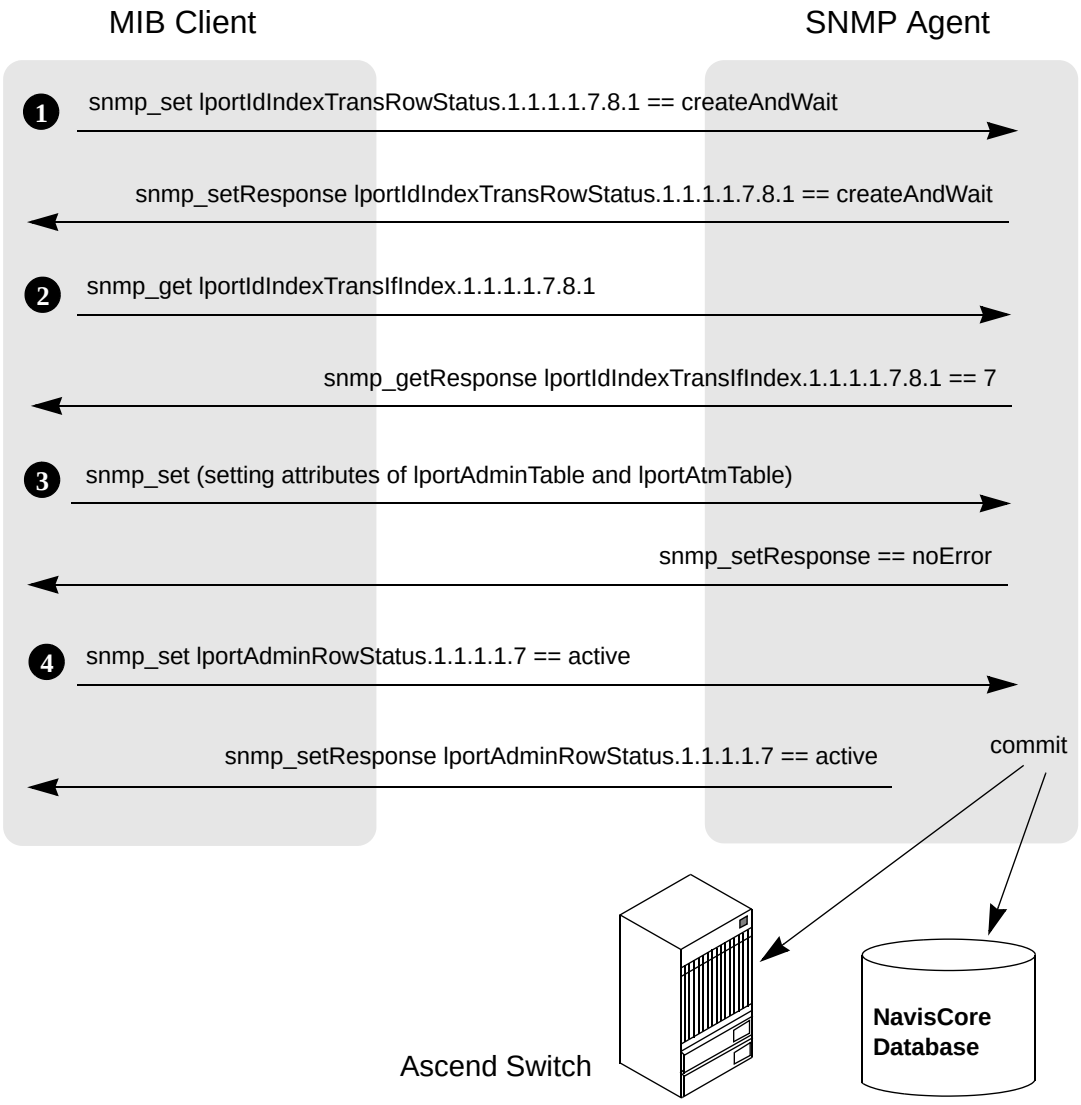


Figure 4-1. Creating an ATM LPort

Example 4: set command to Modify an ATM LPort

You can modify an LPort using either of the following methods:

- Specifying the interface number of the LPort
- Specifying the LPort's VPI/VCI pair

Before modiying any attribute, perform an snmp_get request on the RowStatus attribute to check if another user is currently accessing the entry. If the entry is in use, retry your request later.

Before modifying the LPort attributes, set the lportIdIndexTransRowStatus to the notInService state. You can skip this step if you specify the attribute modifications in a single PDU.

To modify attributes of an ATM LPort for which you do not know the interface number, use the following steps:

1. Issue an snmp_set request to set the LPort to the notInService state, based on the LPort ID. Set the lportIdIndexTransRowStatus to the notInService state, specifying the switchIdIndex 1.1.1.1, slotIdIndex 7, pportIdIndex 8, and lportIdIndex 1.

   The SNMP agent processes the request and returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2).

2. Issue an snmp_get request to obtain the interface number (lportIfIndex) that will be used to modify the entry in the lportAdminTable and the lportAtmTable. Issue the request on the lportIdIndexTransIfIndex, specifying the switchIdIndex, slotIdIndex, pportIdIndex, and lportIdIndex values.

   The SNMP agent processes the request and returns an snmp_getResponse (SNMPv1) or an snmp_Response (SNMPv2) with the lportIfIndex 7.

3. Issue a series of snmp_set requests that modify values of the attributes of the LPort in both the lportAdminTable and the lportAtmTable.

   The SNMP agent processes the requests by storing the values in MIB cache. Then, the agent returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2).

4. Issue an snmp_set request to commit the modified entry. Set the lportAdminRowStatus to the active state, specifying the switchIdIndex 1.1.1.1 and the lportIfIndex 7. This command automatically sets the lportIdIndexTransRowStatus to the active state.

   The SNMP agent processes the request by committing the modified entry to the switch and to the NavisCore database.

   On receipt of noError messages from the switch and database, the SNMP agent returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2).

Figure 4-2 shows the request-response message flow between the MIB client, the SNMP agent, and the database when modifying attributes of the LPort.

> Figure 4-2 uses SNMPv1 syntax. If you are using SNMPv2, the syntax used in messages from the agent to the MIB client will differ. For example, when a v1 agent sends a message beginning with `snmp_setResponse`, a v2 agent uses `snmp_Response`. When a v1 agent sends `snmp_getResponse`, a v2 agent sends `snmp_Response`.

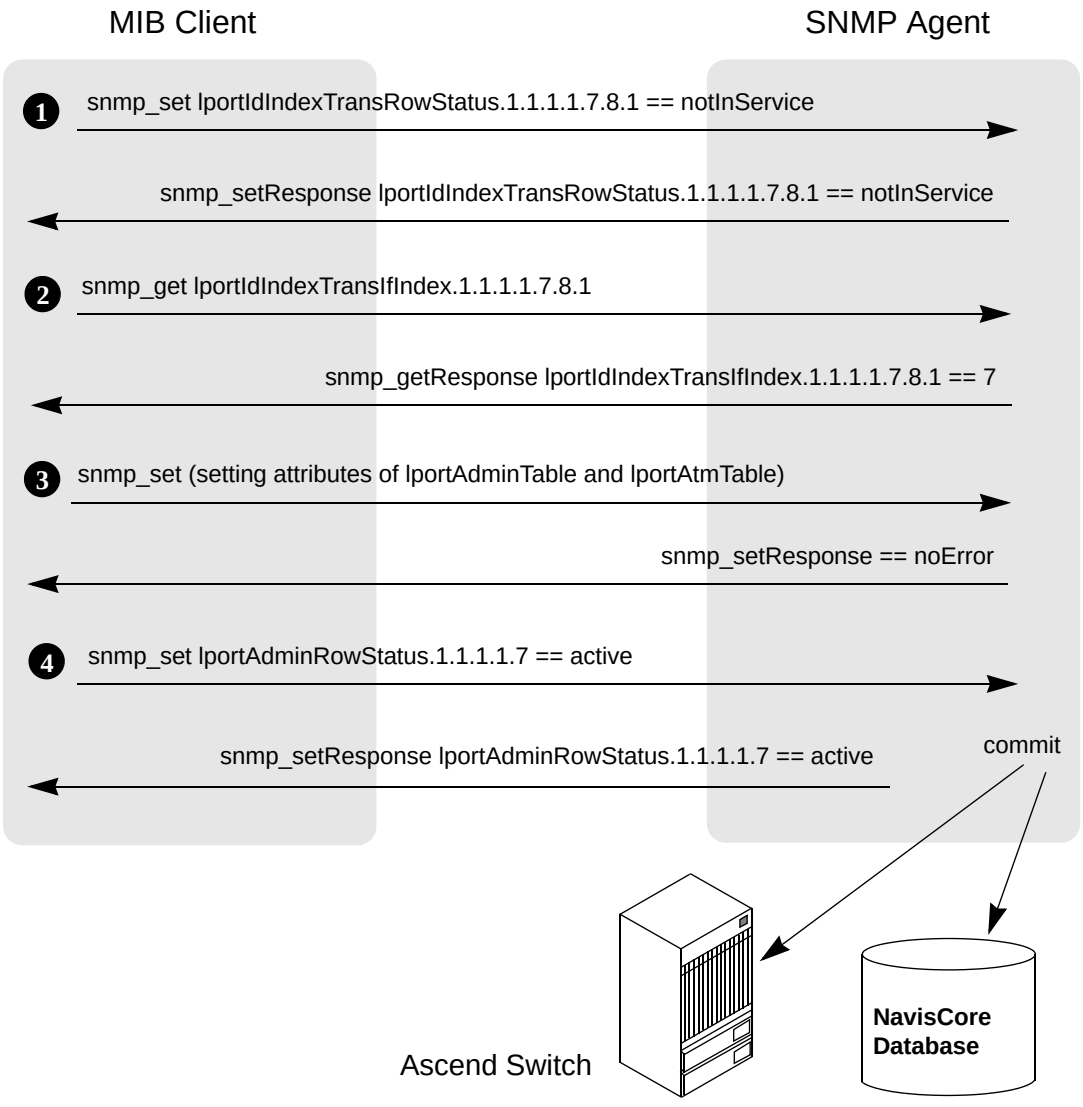


Figure 4-2. Modifying an ATM LPort

Example 5: set Command to Delete an ATM LPort

You can delete an LPort using either of the following methods:

- Specifying the interface number of the LPort
- Specifying the LPort ID

Before deleting an object, perform an snmp_get request on the RowStatus attribute to check if another user is currently accessing the object. If the object is in use, retry your request later.

To delete an ATM LPort for which you do not know the interface number, use the following step:

1. Issue an snmp_set request to delete an LPort based on the LPort ID. Set the lportIdIndexTransRowStatus to the destroy state, specifying the switchIdIndex 1.1.1.1, slotIdIndex 7, pportIdIndex 8, and lportIdIndex 1. (As an alternative, you could set the lportAdminRowStatus to the destroy state, as these attributes are linked by aliasing.)

   The SNMP agent processes the request by committing the modified entry to the switch and to the NavisCore database.

   On receipt of noError messages from the switch and database, the SNMP agent returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2).

Figure 4-3 shows the request-response message flow between the MIB client, the SNMP agent, and the database when deleting the LPort.

Figure 4-3 uses SNMPv1 syntax. If you are using SNMPv2, the syntax used in messages from the agent to the MIB client will differ. For example, when a v1 agent sends a message beginning with `snmp_setResponse`, a v2 agent sends `snmp_Response`.

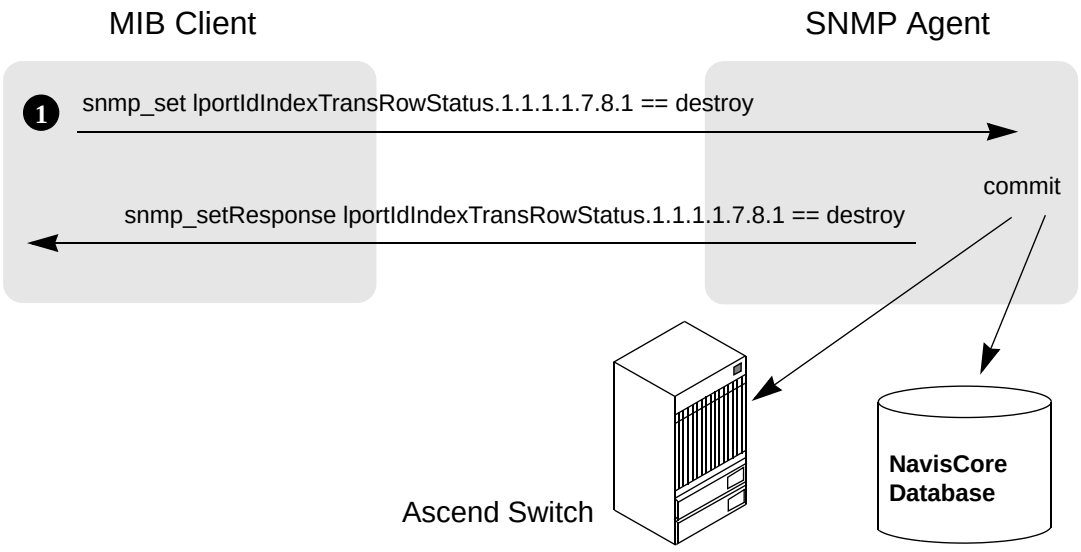


Figure 4-3. Deleting an ATM LPort Using its VPI/VCI Pair

To delete an ATM LPort for which you know the interface number, use the following step:

1. Issue an snmp_set request to delete an LPort based on the LPort's interface number. Set the lportAdminRowStatus to the destroy state, specifying the switchIdIndex 1.1.1.1 and the lportIfIndex 7.

   The SNMP agent processes the request by committing the modified entry to the switch and to the NavisCore database.

   On receipt of noError messages from the switch and database, the SNMP agent returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2).

Figure 4-4 shows the request-response message flow between the MIB client, the SNMP agent, and the database when deleting the LPort.

Figure 4-4 uses SNMPv1 syntax. If you are using SNMPv2, the syntax used in messages from the agent to the MIB client will differ. For example, when a v1 agent sends a message beginning with `snmp_setResponse`, a v2 agent sends `snmp_Response`.

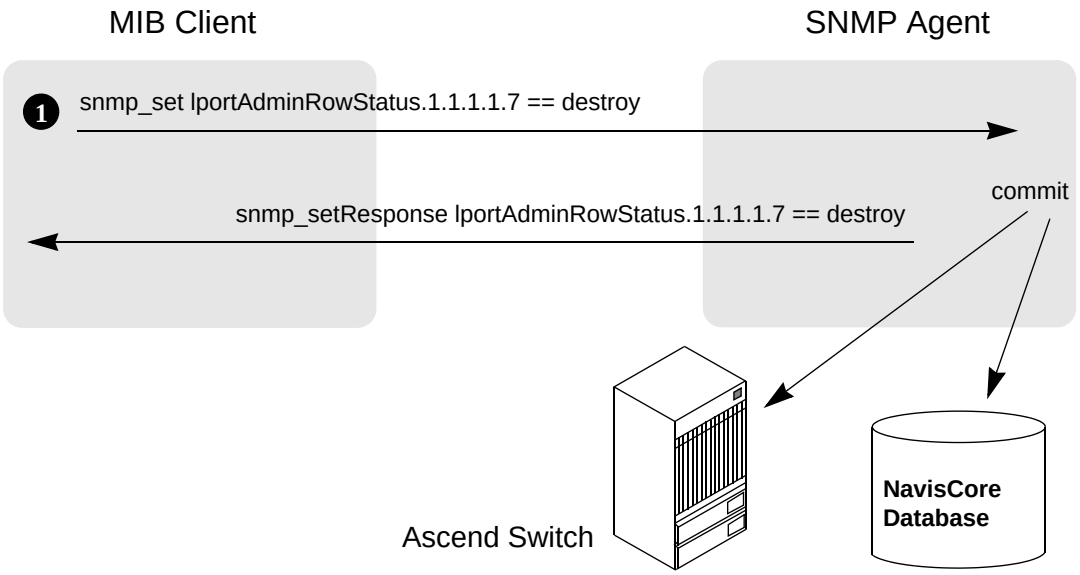


Figure 4-4. Deleting an ATM LPort Using its Interface Number

Example 6: set Command to Create an ATM Circuit

To create an ATM circuit, you define the two circuit endpoints using the atmCircuitEndpointTable and establish their interconnection using the circuitCrossConnectTable (see Table 4-2).

To create an ATM circuit, use the following steps:

1. Issue an snmp_set request to define the two circuit endpoints and establish their interconnection. Set the atmCircuitEndpointRowStatus to the createAndWait state, specifying both endpoint 1 (switchIdIndex 1.1.1.1, lportIfIndex 10, vpiIdIndex 8, and vciIdIndex 34) and endpoint 2 (switchIdIndex 2.2.2.2, lportIfIdIndex 4, vpiIdIndex 4, and vciIdIndex 54) in a single PDU.

   The SNMP agent processes the request and returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2).

2. Issue a series of snmp_set requests that assign values to the attributes of the circuit endpoints in the atmCircuitEndpointTable.

   The SNMP agent processes the requests by storing the values in MIB cache. Then, the agent returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2).

3. Issue an snmp_get request to obtain the circuit number that will be used to create a new entry in the circuitCrossConnectTable. Specify the switchIdIndex, lportIfIndex, vpiIdIndex, and vciIdIndex values for one of the endpoints (the circuit number is the same for both endpoints).

The SNMP agent processes the request and returns an snmp_getResponse (SNMPv1) or an snmp_Response (SNMPv2) with the atmCircuitEndpointCircuitNumber 10.

4. Issue a series of snmp_set requests that assign values to the attributes of the circuit interconnection in the circuitCrossConnectTable.

   The SNMP agent processes the requests by storing the values in MIB cache. Then, the agent returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2).

5. Issue an snmp_set request to commit the new entry. Set the circuitCrossConnectRowStatus to the active state, specifying the atmCircuitEndpointCircuitNumber 10. This command automatically sets the atmCircuitEndpointRowStatus of the two endpoints to the active state.

   The SNMP agent processes the request by committing the new entry to the switch and to the NavisCore database.

   On receipt of noError messages from the switch and database, the SNMP agent returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2).

Figure 4-5 shows the request-response message flow between the MIB client, the SNMP agent, and the database when adding the ATM circuit.

Figure 4-5 uses SNMPv1 syntax. If you are using SNMPv2, the syntax used in messages from the agent to the MIB client will differ. For example, when a v1 agent sends a message beginning with `snmp_setResponse`, a v2 agent sends `snmp_Response`. When a v1 agent sends `snmp_getResponse`, a v2 agent sends `snmp_Response`.

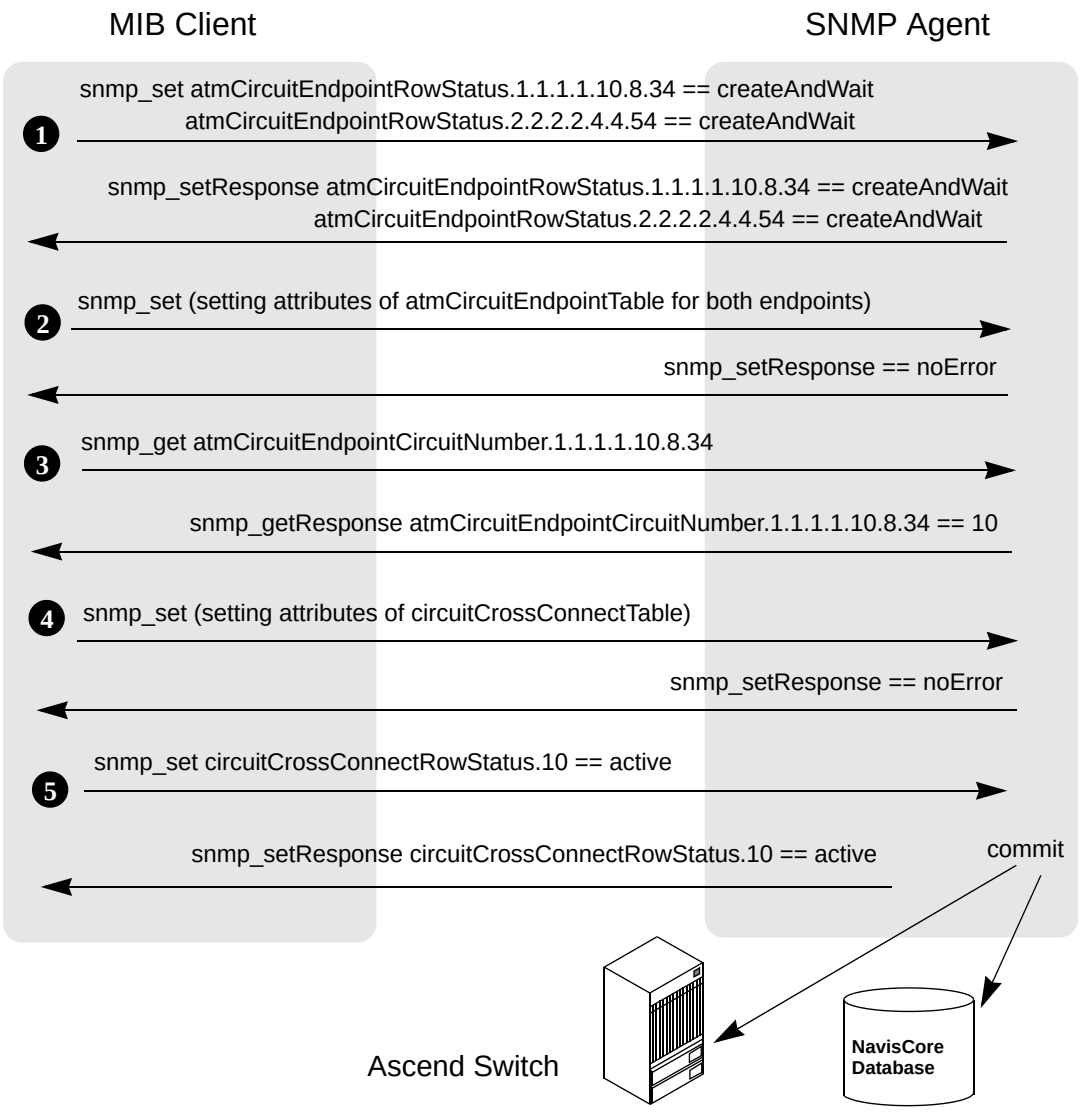


Figure 4-5. Creating an ATM Circuit

Example 7: set Command to Modify an ATM Circuit

You can modify a circuit using either of the following methods:

- Specifying the circuit number
- Specifying the circuit's endpoints

Before performing a modification on any attribute, perform an snmp_get request on the RowStatus attribute to check if another user is currently accessing the entry. If the entry is in use, retry your request later.

Before modifying the circuit attributes, set the circuitCrossConnectRowStatus to the notInService state. You can skip this step if you specify the attribute modifications in a single PDU.

To modify attributes of a circuit, use the following steps:

1. Issue an snmp_get request to obtain the circuit number in the appropriate Circuit Endpoint Table. Specify the switch IP address, lportIfIndex, vpiIdIndex, and vciIdIndex values for one of the endpoints (the circuit number is the same for both endpoints).

   If you know the circuit number, skip to step 2.

2. Issue an snmp_set request to set the circuit to the notInService state, based on the circuit number. Set the circuitCrossConnectRowStatus to the notInService state, specifying the circuit number 10.

   The SNMP agent processes the request and returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2).

3. Issue a series of snmp_set requests that modify values of the attributes of the circuit. Modifications are made to the circuitCrossConnectTable and the atmCircuitEndpointTable.

   The SNMP agent processes the requests by storing the values in MIB cache. Then, the agent returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2).

4. Issue an snmp_set request to commit the modified entry. Set the circuitCrossConnectRowStatus to the active state, specifying the circuit number 10. This command automatically sets the atmCircuitEndpointRowStatus to the active state.

   The SNMP agent processes the request by committing the modified entry to the switch and to the NavisCore database.

   On receipt of noError messages from the switch and database, the SNMP agent returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2).

Figure 4-6 shows the request-response message flow between the MIB client, the SNMP agent, and the database when modifying attributes of the circuit.

> Figure 4-6 uses SNMPv1 syntax. If you are using SNMPv2, the syntax used in messages from the agent to the MIB client will differ. For example, when a v1 agent sends a message beginning with `snmp_setResponse`, a v2 agent sends `snmp_Response`.

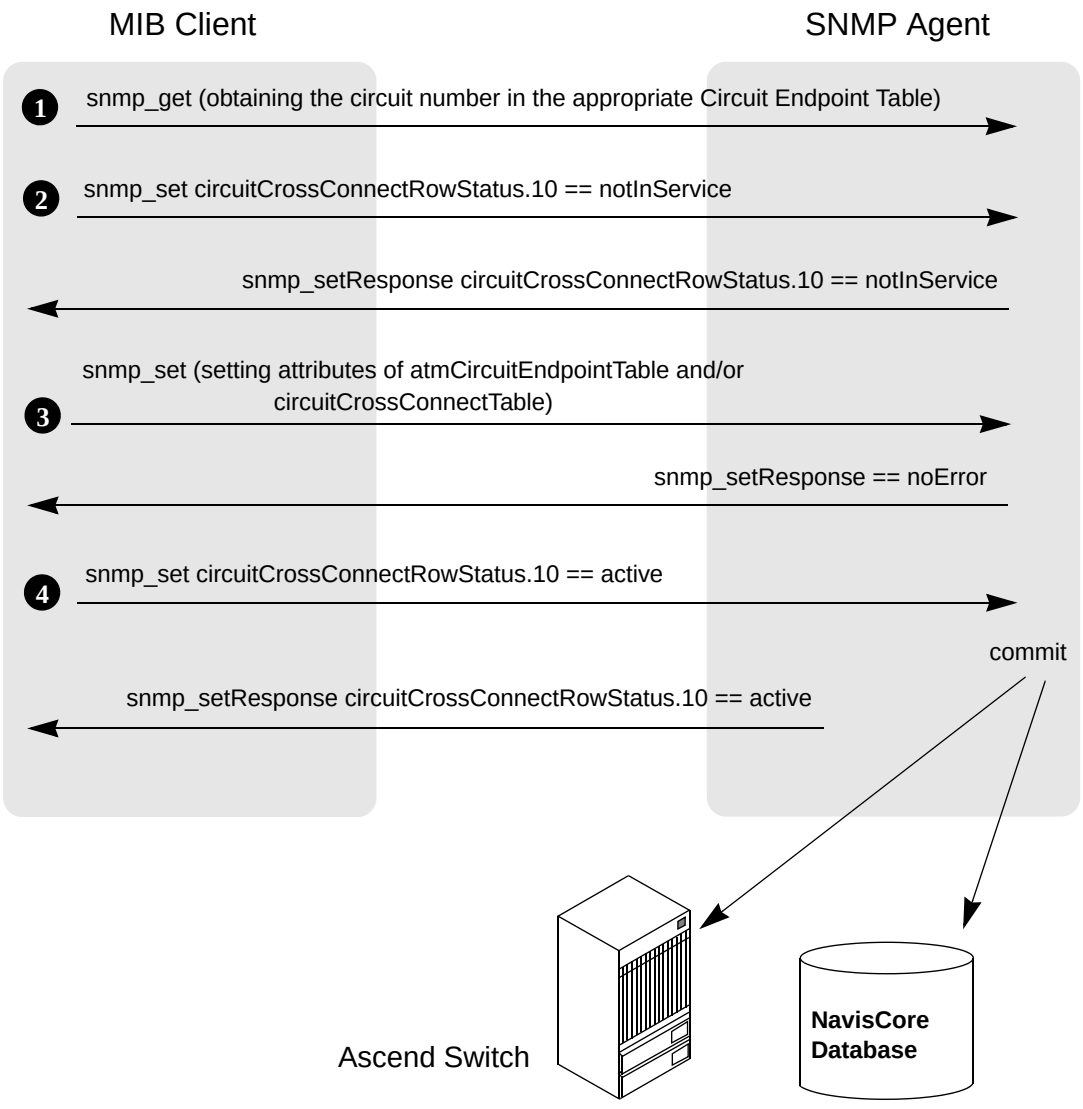


Figure 4-6. Modifying an ATM Circuit Using its Circuit Number

Example 8: set Command to Delete an ATM Circuit

You can delete a circuit using either of the following methods:

- Specifying the circuit number
- Specifying the circuit's endpoints

Before deleting an object, perform an snmp_get request on the RowStatus attribute to check if another user is currently accessing the object. If the object is in use, retry your request later.

To delete an ATM circuit for which you know the circuit number, use the following step:

1. Issue an snmp_set request to delete a circuit based on the circuit number. Set the circuitCrossConnectRowStatus to the destroy state, specifying the circuit number 10.

   The SNMP agent processes the request by committing the modified entry to the switch and to the NavisCore database.

   On receipt of noError messages from the switch and database, the SNMP agent returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2).

Figure 4-7 shows the request-response message flow between the MIB client, the SNMP agent, and the database when deleting the circuit.

Figure 4-7 uses SNMPv1 syntax. If you are using SNMPv2, the syntax used in messages from the agent to the MIB client will differ. For example, when a v1 agent sends a message beginning with `snmp_setResponse`, a v2 agent sends `snmp_Response`.

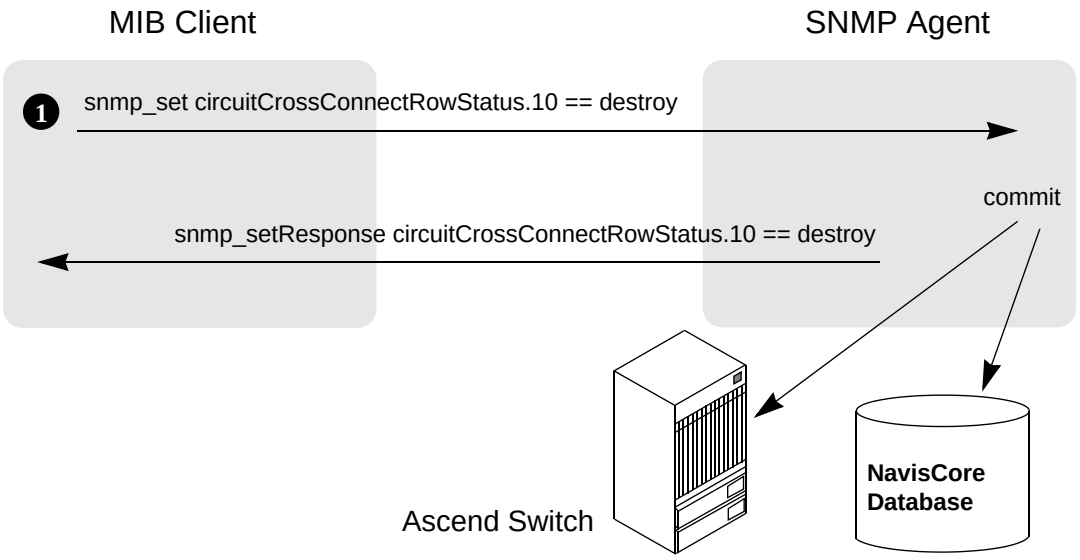


Figure 4-7. Deleting an ATM Circuit Using its Circuit Number

Example 9: set Command to Create a VPN Indexed by Name

To specify a string value when you create an object, you specify the length of the string and the ASCII representation of each of the characters in the string.

To create a VPN indexed by name, use the following steps:

1. Issue an snmp_set request to set the VPN name “abc”. Set the vpnRowStatus to the createAndWait state, specifying the networkIdIndex 100.100.0.0, the length of the name (3 characters), and the ASCII values of each letter in the name (97, 98, and 99, respectively).

   The SNMP agent processes the request and returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2).

2. Issue a series of snmp_set requests that assign values to the attributes of the VPN in the vpnTable.

   The SNMP agent processes the requests by storing the values in MIB cache. Then, the agent returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2).

3. Issue an snmp_set request to commit the new entry. Set the vpnRowStatus to the active state, specifying the networkIdIndex 100.100.0.0, the length of the name, and the ASCII values of each letter in the name.

   The SNMP agent processes the request by committing the new entry to the switch and to the NavisCore database.

On receipt of noError messages from the switch and database, the SNMP agent returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2) to the MIB client.

Figure 4-8 shows the request-response message flow between the MIB client, the SNMP agent, and the database when adding the VPN.

> Figure 4-8 uses SNMPv1 syntax. If you are using SNMPv2, the syntax used in messages from the agent to the MIB client will differ. For example, when a v1 agent sends a message beginning with `snmp_setResponse`, a v2 agent sends `snmp_Response`.

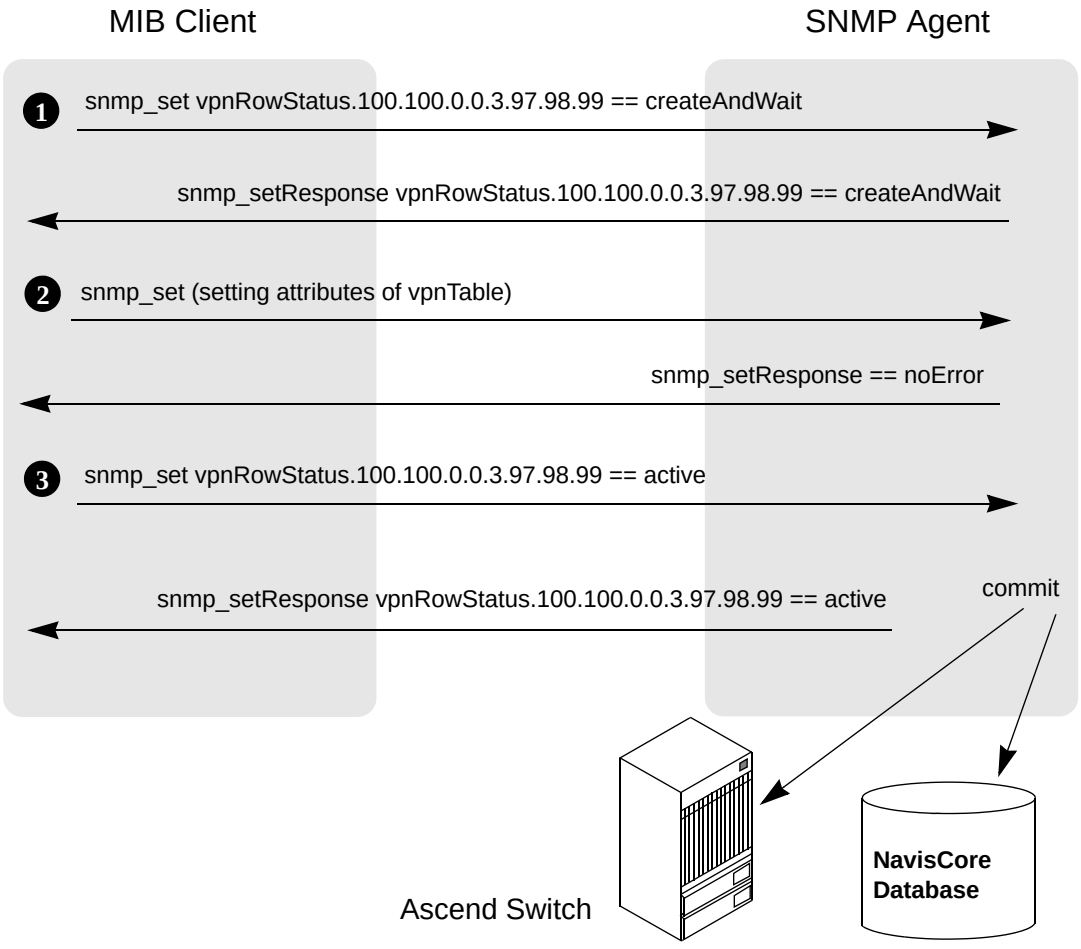


Figure 4-8. Creating a VPN Indexed by Name

Example 10: set Command to Create a ServiceName Indexed by Name

To specify a string value when you create an object, you specify the length of the string and the ASCII representation of each of the characters in the string.

When you create a ServiceName, the first PDU should contain the networkServiceNameRowStatus as the first varbind and the networkServiceNamePrimaryLPort as the second varbind.

To create a ServiceName indexed by the name “abc”, use the following steps:

1. Issue an snmp_set request to define the primary ServiceName binding. In a single PDU, set the networkServiceNameRowStatus to the createAndWait state and set the networkServiceNamePrimaryLPort to the objectId (lportAdminIfIndex) of the LPort. Specify the networkIdIndex 100.100.0.0, the length of the name (3 characters), and the ASCII values of each letter in the name (97, 98, and 99, respectively).

   The SNMP agent processes the request and returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2).

2. Issue a series of snmp_set requests that assign values to the attributes of the ServiceName in the networkServiceNameTable.

Do not set the networkServiceNameBackupLPort attribute in an add request. Otherwise, an error will be reported when the new entry is committed to the database.

   The SNMP agent processes the requests by storing the values in MIB cache. Then, the agent returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2).

3. Issue an snmp_set request to commit the new entry. Set the networkServiceNameRowStatus to the active state, specifying the networkIdIndex 100.100.0.0, the length of the name, and the ASCII values of each letter in the name.

   The SNMP agent processes the request by committing the new entry to the switch and to the NavisCore database.

   On receipt of noError messages from the switch and database, the SNMP agent returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2) to the MIB client.

Figure 4-9 shows the request-response message flow between the MIB client, the SNMP agent, and the database when adding the ServiceName binding.

Figure 4-9 uses SNMPv1 syntax. If you are using SNMPv2, the syntax used in messages from the agent to the MIB client will differ. For example, when a v1 agent sends a message beginning with `snmp_setResponse`, a v2 agent sends `snmp_Response`.

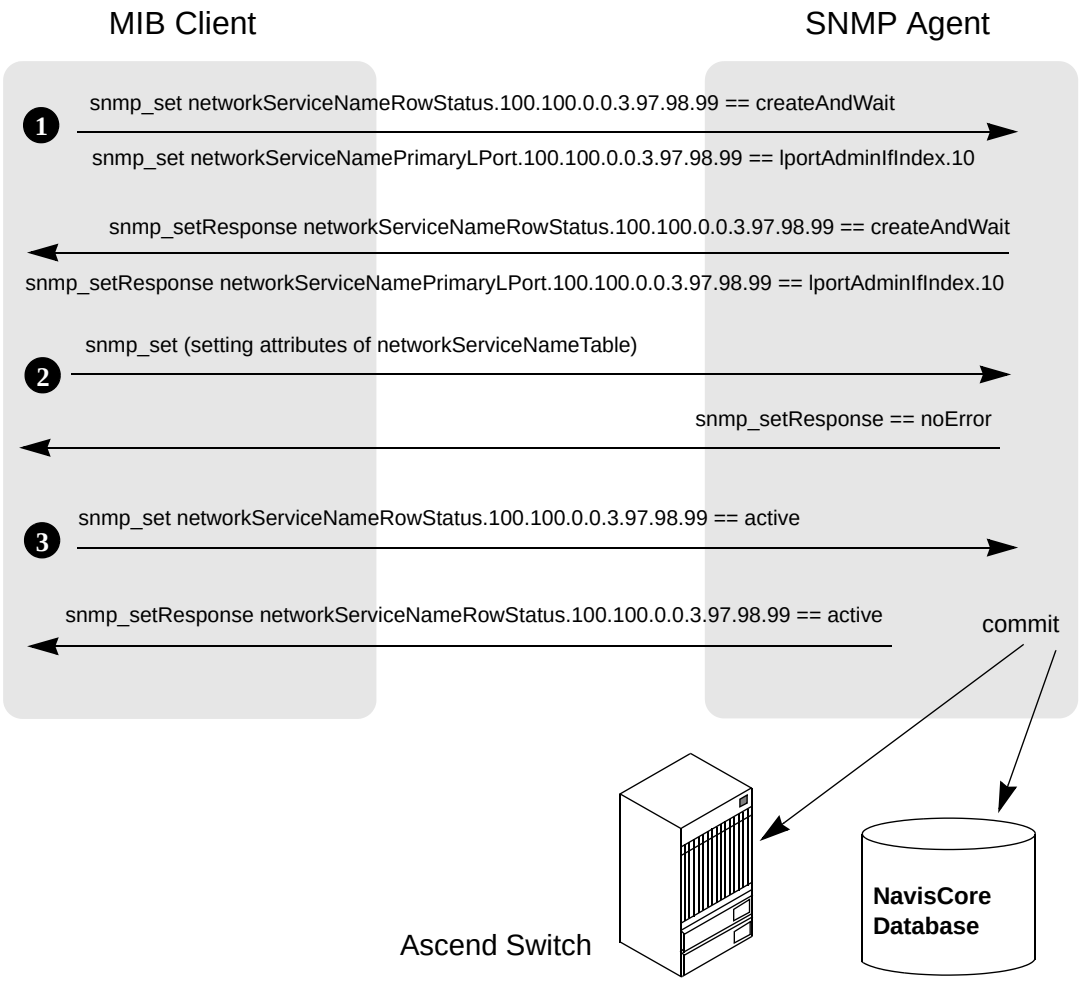


Figure 4-9. Creating a ServiceName Indexed by Name

Example 11: set command to Modify a ServiceName Indexed by Name

Before performing a modification on any attribute, perform an snmp_get request on the RowStatus attribute to check if another user is currently accessing the entry. If the entry is in use, retry your request later.

Before modifying the ServiceName attributes, set the networkServiceNameRowStatus to the notInService state. You can skip this step if you specify the attribute modifications in a single PDU.

To modify attributes of a ServiceName indexed by the name “abc”, use the following steps:

1. Issue an snmp_set request to set the ServiceName “abc” to the notInService state. Set the networkServiceNameRowStatus to the notInService state, specifying the networkIdIndex 100.100.0.0, the length of the name (3 characters), and the ASCII values of each letter in the name (97, 98, and 99, respectively).

   The SNMP agent processes the request and returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2).

2. Issue a series of snmp_set requests that assign values to the attributes of the ServiceName in the networkServiceNameTable.

   The SNMP agent processes the requests by storing the values in MIB cache. Then, the agent returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2).

3. Issue an snmp_set request to commit the new entry. Set the networkServiceNameRowStatus to the active state, specifying the networkIdIndex 100.100.0.0, the length of the name, and the ASCII values of each letter in the name.

   The SNMP agent processes the request by committing the new entry to the switch and to the NavisCore database.

   On receipt of noError messages from the switch and database, the SNMP agent returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2) to the MIB client.

Figure 4-10 shows the request-response message flow between the MIB client, the SNMP agent, and the database when modifying the ServiceName binding.

Figure 4-10 uses SNMPv1 syntax. If you are using SNMPv2, the syntax used in messages from the agent to the MIB client will differ. For example, when a v1 agent sends a message beginning with `snmp_setResponse`, a v2 agent sends `snmp_Response`.

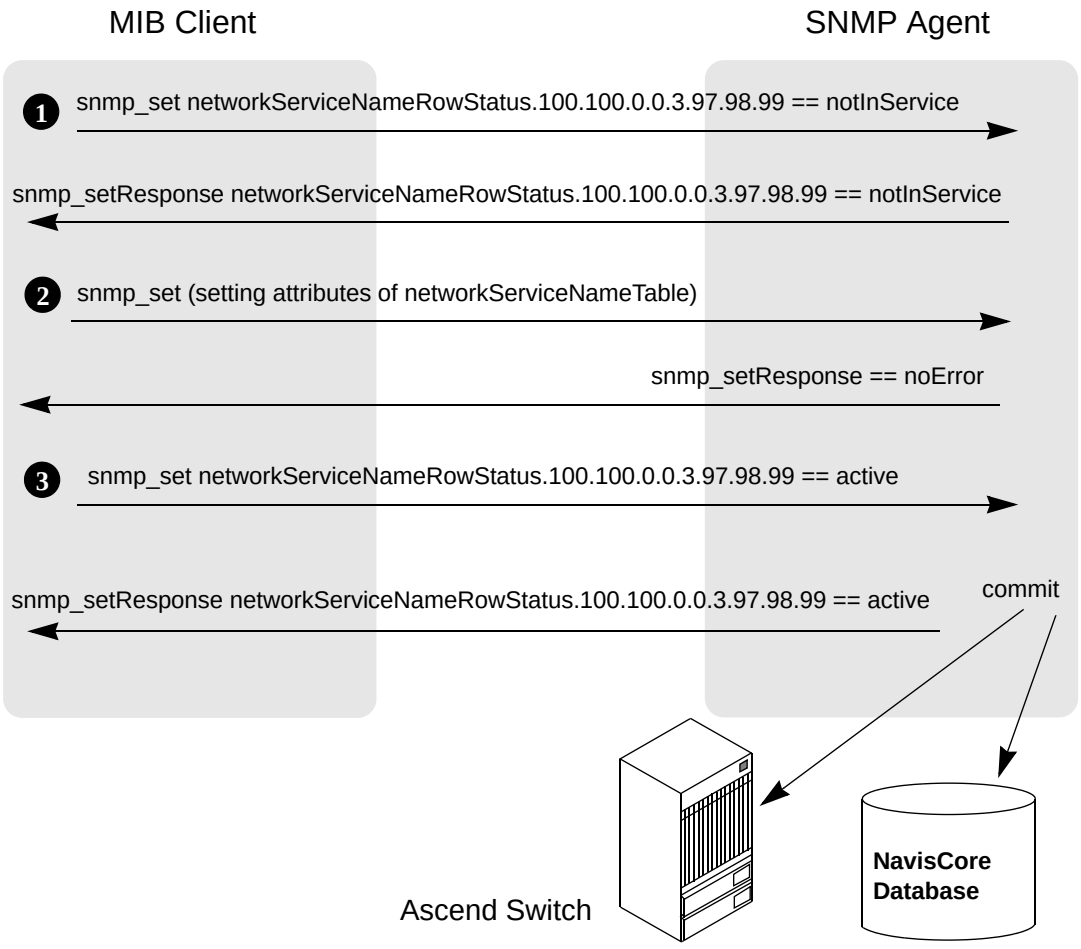


Figure 4-10. Modifying a ServiceName Indexed by Name

Example 12: set command to Delete a ServiceName Indexed by Name

Before deleting an object, perform an snmp_get request on the RowStatus attribute to check if another user is currently accessing the object. If the object is in use, retry your request later.

To delete a ServiceName indexed by the name “abc”, use the following steps:

1. Issue an snmp_set request to set the ServiceName “abc” to the destroy state. Set the networkServiceNameRowStatus to the destroy state, specifying the networkIdIndex 100.100.0.0, the length of the name (3 characters), and the ASCII values of each letter in the name (97, 98, and 99, respectively).

   The SNMP agent processes the request by committing the modified entry to the switch and to the NavisCore database.

   On receipt of noError messages from the switch and database, the SNMP agent returns a successful snmp_setResponse (SNMPv1) or an snmp_Response (SNMPv2).

Figure 4-11 shows the request-response message flow between the MIB client, the SNMP agent, and the database when deleting the ServiceName.

Figure 4-11 uses SNMPv1 syntax. If you are using SNMPv2, the syntax used in messages from the agent to the MIB client will differ. For example, when a v1 agent sends a message beginning with `snmp_setResponse`, a v2 agent sends `snmp_Response`.

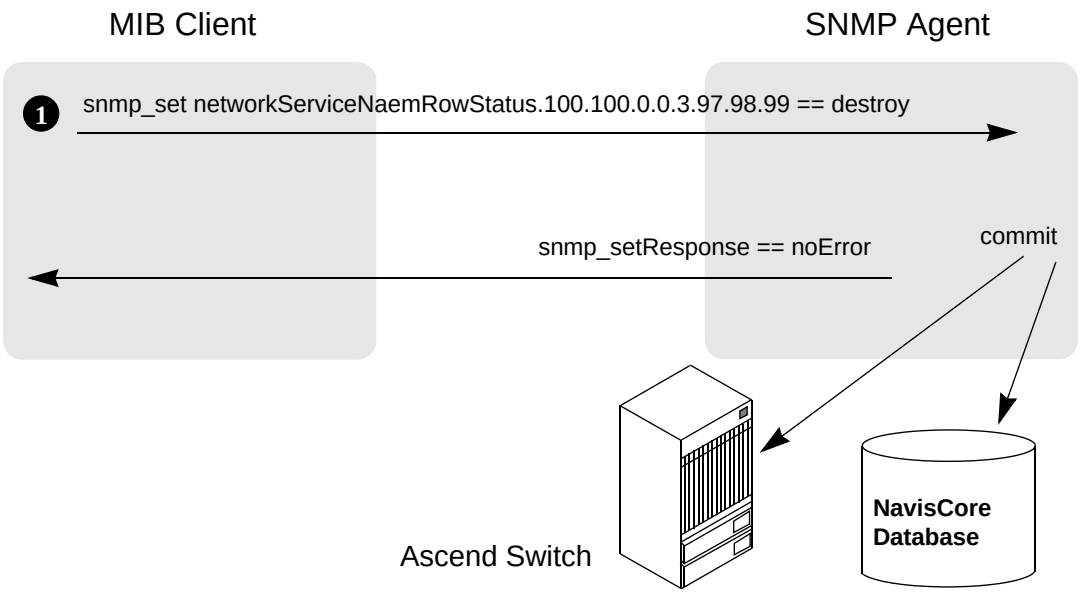


Figure 4-11. Deleting a ServiceName Indexed by Name

Index

A

C

D

E

F

H

I

L

M

N

O

P

R

S

T

U

V

W