Cisco CDS Video Navigator 2.2 User Guide

March 2011

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

Text Part Number: OL-22179-01

CONTENTS

Preface

This preface describes the use and organization of the *Cisco CDS Video Navigator 2.2 User Guide*. The preface also outlines the document conventions and support information.

This preface contains the following sections:

Document Revision History

The following Document Revision History table records technical changes to this document.

Document	Date	Change Summary
OL-22179-01	March 2011	Initial release

Objectives

This document describes the Cisco Content Delivery System (CDS) Video Navigator application, Release 2.2, and explains how to configure Video Navigator and other related software.

Document Organization

This guide uses the following sections:

Chapter	Description
Chapter 1, "Introduction to Cisco CDS Video Navigator"	Provides an introduction to Cisco CDS Video Navigator, Cisco CDS Poster Art Server, and Cisco Content Storage Interface.
Chapter 2, "Getting Started with CDS Video Navigator"	Explains how to perform the initial configuration tasks needed to get Video Navigator and Poster Art Server running.
Chapter 3, "Understanding and Using the Cisco CDS Poster Art Server"	Provides a detailed understanding of the operation and configuration of Poster Art Server.
Chapter 4, "Using the Cisco Content Storage Interface"	Provides details of the Content Storage Interface API and schema.
Chapter 5, "Troubleshooting CDS Video Navigator and Poster Art Server"	Describes how to identify and remedy problems related to Video Navigator and Poster Art Server.

Document Conventions

This document uses the following conventions:

Convention	Description
boldface font	Commands and keywords are in **boldface**.
italic font	Arguments for which you supply values are in *italics*.
[ ]	Elements in square brackets are optional.
{*x* | *y* | *z*}	Alternative, mutually exclusive, keywords are grouped in braces and separated by vertical bars.
[*x* | *y* | *z*]	Optional alternative keywords are grouped in brackets and separated by vertical bars.
string	A nonquoted set of characters. Do not use quotation marks around the string or the string will include the quotation marks.
`screen` font	Terminal sessions and information the system displays are in `screen` font.
`boldface screen` font	Information you must enter is in **`boldface screen`** font.
`italic screen` font	Arguments for which you supply values are in *`italic screen`* font.
^	The symbol ^ represents the key labeled Control—for example, the key combination ^D in a screen display means hold down the Control key while you press the D key.
< >	Nonprinting characters, such as passwords, are in angle brackets in contexts where italics are not available.
!, #	An exclamation point (!) or a pound sign (#) at the beginning of a line of code indicates a comment line.

Note Means *reader take note*. Notes contain helpful suggestions or references to materials not contained in this publication.

Tip Means the following information might help you solve a problem.

Caution Means *reader be careful*. In this situation, you might do something that could result in equipment damage or loss of data.

Related Documentation

Refer to the following documents for additional information about Cisco CDS Video Navigator and the Cisco CDE110 appliance:

- *Release Notes for Cisco CDS Video Navigator Application, Release 2.2*

  http://www.cisco.com/en/US/docs/video/cds/cda/vn/2_2/release_notes/vn_notes2_2.html
- *Cisco Content Delivery Engine 110 Hardware Installation Guide*

  http://www.cisco.com/en/US/docs/video/cds/cde/cde110/installation/guide/cde110_install.html
- *Regulatory Compliance and Safety Information for the Cisco Content Delivery Engine 110*

  http://www.cisco.com/en/US/docs/video/cds/cde/regulatory/compliance/cde110_rcsi.pdf
- *Open Source Used in CDS VN 2.2*

  http://www.cisco.com/en/US/products/ps7127/products_licensing_information_listing.html

Obtaining Documentation and Submitting a Service Request

For information on obtaining documentation, submitting a service request, and gathering additional information, see the monthly *What's New in Cisco Product Documentation*, which also lists all new and revised Cisco technical documentation, at:

http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html

Subscribe to the *What's New in Cisco Product Documentation* as a Really Simple Syndication (RSS) feed and set content to be delivered directly to your desktop using a reader application. The RSS feeds are a free service and Cisco currently supports RSS Version 2.0.

CHAPTER 1

Introduction to Cisco CDS Video Navigator

This chapter provides an introduction to Cisco CDS Video Navigator, Release 2.2. This release includes the Cisco CDS Poster Art Server application, which supports the storage, management, and delivery of poster art for the end user, while providing an interface that allows third-party applications to access all posters. In addition, a Cisco Content Storage Interface (CSI) application programming interface (API) is provided; this API supports interactions between Video Navigator or Poster Art Server and the video backoffice (VBO) for the management and ingestion of video assets.

The following major topics are presented:

- Cisco CDS Video Navigator Overview, page 1-1
- Cisco CDS Poster Art Server Overview, page 1-4
- Cisco Content Storage Interface Overview, page 1-5
- Support for Resiliency with Cisco ACE, page 1-6

Cisco CDS Video Navigator Overview

Cisco CDS Video Navigator is a navigation application server that provides information about available video on demand (VOD) content to the on-demand client application running on a set-top box (STB). A web services application, Video Navigator is required for VOD navigation. Because of resource constraints in the STB, in particular insufficient memory to support a VOD catalog of arbitrary size, the catalog metadata (title, genre, rating, length, description, price, and so forth) must reside on an application server—Video Navigator.

Note Beginning with this release, exclusive support is provided for the Cisco Real Time Streaming Protocol (RTSP) network control protocol to communicate with streaming media servers, establishing and controlling media sessions between end points. RTSP enables support for both IP and cable STBs. For the STBs that are supported in this release, see *Release Notes for Cisco CDS Video Navigator 2.2*.

For implementations that require the Interactive Services Architecture (ISA) protocol, Cisco CDS Video Navigator, Release 2.1, is required.

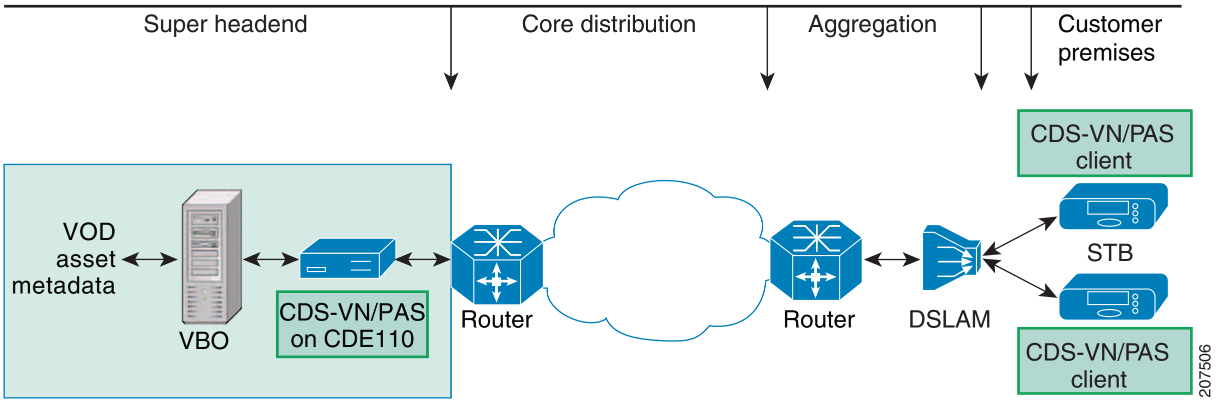
Video Navigator retrieves from the VBO the metadata describing VOD services and offerings, along with subscriber rental history. Because the VBO is not designed to support real-time queries from thousands of STBs simultaneously, Video Navigator caches the metadata and presents the information to the on-demand client application on the STB. The client application uses the metadata and subscriber data to determine which VOD content to display for the subscriber.

Note Unless each application is addressed explicitly, Video Navigator 2.2 is also used to refer to Cisco Video Navigator with Poster Art Server. Both are shipped preinstalled on the Cisco CDE110. For complete information on the Cisco CDE110, see the *Cisco Content Delivery Engine 110 Hardware Installation Guide*.

CDS Video Navigator (CDS-VN) is located between the VBO and the STB. The major components related to Video Navigator Release 2.2 are shown in Figure 1-1:

- CDS-VN and CDS Poster Art Server (PAS) applications on the CDE110—Applications on the Cisco Content Delivery Engine 110 (CDE110) respond to client requests that are made as the subscriber navigates the VOD menu.
- Video Navigator and Poster Art Server clients—On-demand applications on the STB request metadata from the Video Navigator or Poster Art Server and display the VOD catalog or poster art to the subscriber.

Figure 1-1 Video Navigator Release 2.2 Components



Using the Cisco Content Storage Interface (CSI) to support the ingestion, removal, and management of poster data by the VBO (see Cisco Content Storage Interface Overview, page 1-5), Video Navigator retrieves the catalog data from the VBO. It creates and stores an XML data structure representing the hierarchy of the VOD content menu in a format optimized for fast access by the STB client. To minimize the need for the dynamic processing of catalog data requested by the client, Video Navigator pregenerates and caches various lists of sorted and filtered offerings. The pregenerated lists may include such items as Titles (A–M), Titles (N–Z), Titles by Genre, and Titles by Studio/Network.

As the subscriber navigates the VOD program guide on the STB by browsing and selecting shows and movies to view, Video Navigator presents the STB client with XML data describing each currently displayed part of the guide. When the subscriber session initiates a request for noncatalog data, such as poster art or trailers, Video Navigator redirects the request to the URL or URI specified in the metadata.

A subscriber may purchase new services or cancel existing ones, either by calling the service operator (who manually provisions them), or by ordering the services directly (by interacting with the Video Navigator client on the STB). The Video Navigator client transacts all purchases directly through the VBO server, which interfaces with the subscriber management, traffic, and billing systems of the operator. Video Navigator has no direct role in the purchase of any VOD service or offering.

The VBO stores subscriber purchases for billing purposes. Video Navigator also caches the history about subscriber purchases and bookmarks that occur during each VOD navigation session, so that the transaction state can be reflected in the guide on the STB. Service listings are correlated with the status of the subscriber's account and are flagged in the XML data structure, allowing the STB client to differentiate those services to which the user has subscribed from those to which the user has not subscribed. This differentiation enables the promotion and upselling of additional services.

A single Video Navigator server is expected to support a population of 75,000 STBs with an assumed concurrency rate of 2 percent (that is, 1500 simultaneous sessions with Video Navigator clients where users are engaged in any arbitrary combination of actions, such as catalog browsing, checking of rental history, and so forth). The actual capacity of a Video Navigator server may vary, depending on the VOD consumption patterns of the subscriber base. For example, how frequently the VOD guide is accessed and how long the average subscriber stays within the guide are affected by how compelling an operator's VOD service is and how aggressively it is marketed.

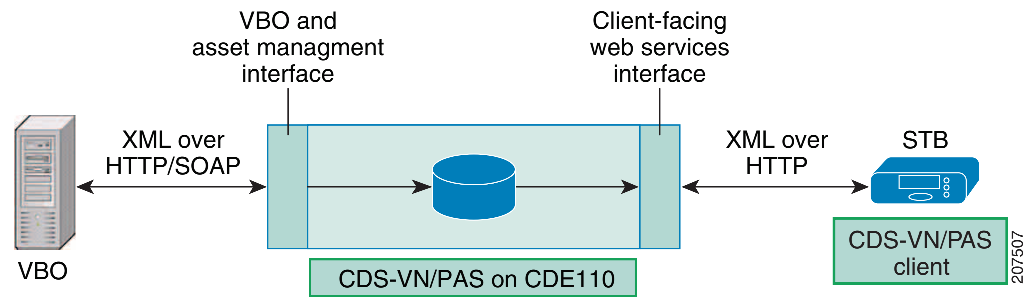
Video Navigator Logical Interfaces

Video Navigator is a web services application that acts similar to a web proxy between the metadata sources for VOD services (the VBO) and the metadata consumers (the Video Navigator clients on the STBs). Cisco Video Navigator Release 2.2 supports two major logical interfaces:

- VBO and asset management interface
- Client-facing web services interface

Figure 1-2 shows these two interfaces and their associated data flows.

Figure 1-2 Video Navigator Logical Interfaces and Data Flows



VBO and Asset Management Interface

The VBO and asset management interface (AMI) is responsible for extracting asset metadata and subscriber data from the VBO. The Video Navigator local cache of VOD catalog offerings, subscriber services, and rental history is automatically synchronized with the VBO database by means of a VBO interface.

Client-Facing Web Services Interface

The client-facing web services interface is the interface to the Video Navigator client application running on the STB. In Video Navigator Release 2.2, this interface provides the Video Navigator client application with a set of APIs so that the client can do the following:

- Browse the VOD catalog navigation tree
- Execute queries related to rental history and purchased services
- Search for electronic program guide (EPG) and VOD assets
- Post purchase and bookmark information

For browsing the VOD catalog, the client-facing web services interface uses XML over HTTP to provide a paging mechanism for efficient navigation.

Video Navigator Client on the STB

The Video Navigator client on the STB is the on-demand menu application, the end node that consumes and uses the metadata. The Cisco IPTV Service Delivery Server (ISDS) application provides the Video Navigator client with the address of the Video Navigator server. When the subscriber signs on and selects the VOD service, the Video Navigator client queries Video Navigator for the list of services to which the client is entitled. Video Navigator then synchronizes its subscriber database for that STB with the VBO, to make sure that the Video Navigator cache is up-to-date.

Each page of the VOD menu is an STB-resident HTML template. In response to queries from the Video Navigator client, Video Navigator returns the XML data used to fill out the template (title, genre, rating, and so forth), and sends information to create the link for each actionable button that appears on the VOD menu.

The STB software includes middleware for manipulating digital media, as well as a web browser engine optimized for user interfaces.

Cisco CDS Poster Art Server Overview

Cisco CDS Poster Art Server (preinstalled with Video Navigator on the CDE110) is an application server that stores and provides poster art to the on-demand client application running on an STB. It also provides an interface to allow third-party applications to access all posters. It implements the following:

- The Cisco Content Storage Interface (CSI), for the ingestion, removal, and management of poster data by the VBO (see the "Cisco Content Storage Interface Overview" section on page 1-5)
- The poster art client-facing API, for the retrieval of image files by the STB through a standard HTTP interface and GET requests

The following additional features are provided:

- Scaling of images in the *x* and *y* dimensions
- Resizing of JPEG image formats
- Caching of resized images to improve performance
- Removal of all resized images when original poster images are removed

For details, see Chapter 3, "Understanding and Using the Cisco CDS Poster Art Server."

Cisco Content Storage Interface Overview

The Cisco Content Storage Interface (CSI) defines the XML APIs for the interactions between a Cisco CDS component (such as Video Navigator and Poster Art Server) and the VBO for the management and ingestion of metadata or poster art. CSI supports the following:

- Getting a description or status of an asset
- Getting an inventory list of the appropriate asset type from the CDS component, including asset information such as its ID, current state, version, and size
- Adding assets
- Deleting assets
- Updating assets to synchronize them with the VBO

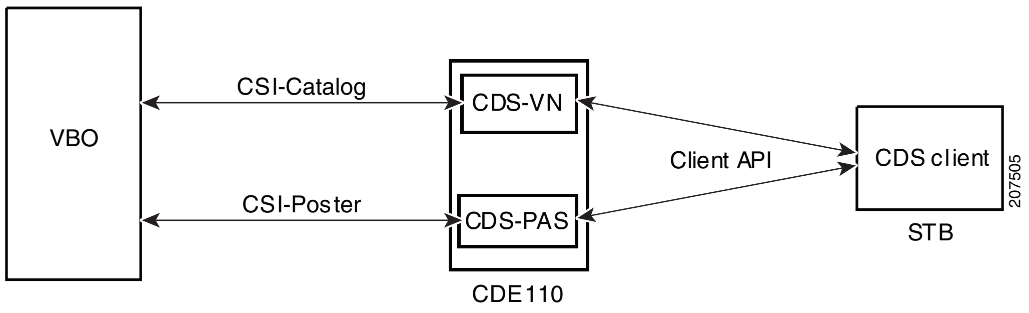
The CSI APIs use the HTTP protocol with an XML-based payload. The CDS component that interacts with the CSI acts as an HTTP server. It accepts HTTP requests from clients and returns HTTP responses to clients. The APIs employ HTTP/1.1 and a bidirectional semipersistent TCP/IP socket connection that opens and closes for each request/response pair. All requests are sent by means of an HTTP POST, as follows:

- The entity-body contains XML data.
- The entity-header includes the following:
  - content-length: *xxxxxx*, where *xxxxx* is the length of XML data
  - content-type: text/xml

The request and response headers are the same except for HTTP errors, such as a bad handler (404 error–document not found).

Figure 1-3 illustrates the role of Cisco CSI, showing the system components that use the CSI API, along with their interfaces.

Figure 1-3 Role of Cisco CSI



For more details, see Chapter 4, "Using the Cisco Content Storage Interface."

Support for Resiliency with Cisco ACE

Cisco Video Navigator and Poster Art Server both support resiliency by means of the Cisco Application Control Engine (ACE) module. See the "Configuring for Resiliency with Cisco ACE (Optional)" section on page 2-9.

For information about Cisco ACE, see "Cisco Application Control Engine Module" at the following URL:

http://www.cisco.com/en/US/prod/collateral/modules/ps2706/ps6906/product_data_sheet0900aecd8045861b.html

Also, visit http://www.cisco.com/go/ace.

CHAPTER 2

Getting Started with CDS Video Navigator

This chapter explains how to perform the initial configuration tasks needed to get Cisco Content Delivery System (CDS) Video Navigator and Cisco CDS Poster Art Server running. (Both applications are preinstalled together.) Read the following sections and perform the initial configuration tasks in this order:

1. Configuring Terminal Emulation Software, page 2-2
2. Connecting Cables to the CDE110, page 2-2
3. Configuring the Ethernet Interfaces, page 2-3
4. Configuring Video Navigator, page 2-5
5. Preparing to Configure Poster Art Server, page 2-6
6. Configuring the Apache Web Server, page 2-6
7. Starting Video Navigator and Verifying System Status, page 2-7

Where optional support for resiliency by means of the Cisco Application Control Engine (ACE) module is required, see Configuring for Resiliency with Cisco ACE (Optional), page 2-9, for basic configuration steps. In addition, Information on CDE110 Hardware, page 2-10 contains some notes on the hardware components and configuration used with Video Navigator.

This chapter assumes that the Cisco CDE110 hardware has been installed as described in the *Cisco Content Delivery Engine 110 Hardware Installation Guide*, including connecting cables and connecting power.

Note To upgrade from a previous release, contact your Cisco account team.

Configuring Terminal Emulation Software

The RJ-45 serial ports on the Cisco CDE110 front and back panels can be used for administrative access to the CDE110 through a terminal server. Terminal emulation software must be configured as follows:

- Bits per second: 9600
- Data bits: 8
- Parity: none
- Stop bits: 1
- Hardware flow control: ON

Connecting Cables to the CDE110

The following cable connections are used on the CDE110:

- For the four Ethernet interfaces on the back of the CDE110, use Category 5 UTP cables to connect the following:
  - Ethernet interface to the network that includes the set-top boxes (STBs) that Video Navigator serves
  - Ethernet interface to the network used for management and video backoffice (VBO) communication
- If a terminal server is used, the RJ-45 cable from the terminal server is connected to an RJ-45 serial port on the front or back of the CDE110.
- If a PC is connected directly to the CDE110 serial port, the cable from the PC is connected to an RJ-45 serial port on the front or back of the CDE110. The PC end of the cable connected to the CDE110 serial port varies, depending on the type of ports supported by the PC.

Note The serial port is used for the system console. Only one serial port (front or back) can be used, because it is one shared serial port. A system console is typically used—rather than a monitor, keyboard, and mouse directly attached to the CDE110.

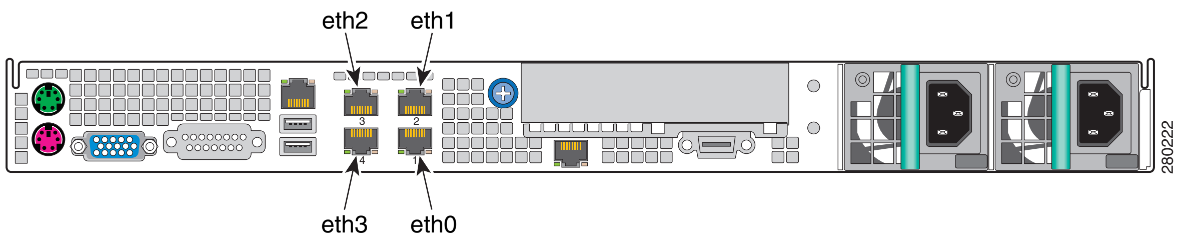
- If a monitor, keyboard, and mouse are used, the cables for the devices are connected to the appropriate connectors on the CDE110.

For the location of connectors on the CDE110 front and back panels, see the *Cisco Content Delivery Engine 110 Hardware Installation Guide*.

Configuring the Ethernet Interfaces

This section explains how to configure an Ethernet interface on the CDE110. For software configuration, the RJ-45 Ethernet ports on the CDE110 back panel are specified as eth0, eth1, eth2, and eth3, as shown in Figure 2-1.

Figure 2-1 Port Numbering for Software Configuration

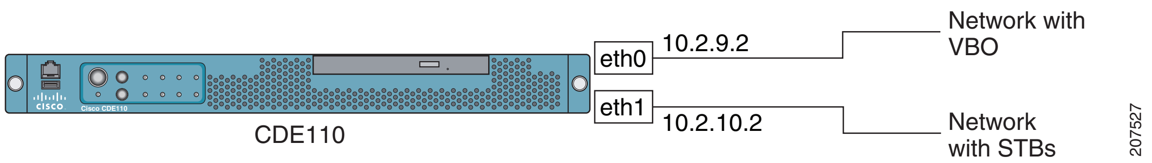


Note On the back panel, the ports labeled 1, 2, 3, and 4 are, respectively, for interfaces eth0, eth1, eth2, and eth3.

Figure 2-2 shows the example IP addresses used in the configuration examples in this section.

- Interface eth0 connects to the network used for management and VBO communication.
- Interface eth1 connects to the network containing the STBs.

Figure 2-2 Example IP Addresses for Video Navigator Configuration



To configure the CDE110 Ethernet interfaces for Video Navigator, do the following:

Step 1 Press the front panel power switch to power on the CDE110.

The operating system boots.

Step 2 Log in as **root** with the password **rootroot**.

Note To change the default password, use the **passwd** command.

Step 3 To configure the eth0 and eth1 used by Video Navigator, use a text editor to modify the two appropriate /etc/sysconfig/network-scripts/ifcfg-eth# files (where # is the number of the Ethernet interface, such as ifcfg-eth1), and do the following:

- Change ONBOOT to **yes**.
- Add the following:

```
IPADDR=ip_address_of this_system_eth#
```

- Add the following:

  ```
  NETMASK=netmask_for_eth#_network
  ```

For example, for the eth1 interface, the /etc/sysconfig/network-scripts/ifcfg-eth1 file would include the following after the modifications:

```
ONBOOT=yes
IPADDR=10.2.10.2
NETMASK=255.255.255.0
```

Step 4 To bring the Ethernet interfaces up, use the **ifup** command for eth0 and eth1 as in the following example:

```
[root@system]# ifup eth1
```

Step 5 Verify that the eth0 and eth1 interfaces are configured correctly and are up and running.

- Use the **ifconfig** *interface* command to verify that each Ethernet interface is up and running and that the IP address and netmask for each are set correctly. The following example is for eth1:

  ```
  [root@system]# ifconfig eth1

  eth1      Link encap:Ethernet  HWaddr 00:0E:0C:C6:F3:0F
            inet addr:10.2.10.2  Bcast:10.2.10.255  Mask:255.255.255.0
            inet6 addr: fe80::20e:cff:fec6:f30f/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:3 errors:0 dropped:0 overruns:0 frame:0
            TX packets:36 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:192 (192.0 b)  TX bytes:2700 (2.6 KiB)
            Base address:0x3000 Memory:b8800000-b8820000
  ```

- Use the **ip link show eth#** command (where # is the Ethernet interface number) to check that the link is up. The following example is for eth1:

  ```
  [root@system]# ip link show eth1

  eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
  link/ether 00:0e:0c:c6:e4:fe brd ff:ff:ff:ff:ff:ff
  ```

- Use the **ping** command to check that the CDE110 can reach the devices directly connected to the Ethernet interfaces (for example, a directly connected router):

  ```
  [root@system]# ping device_IP_address
  ```

Step 6 Use a text editor to modify the /etc/hosts file and add a line with the IP address for eth0 and the associated hostname, as in the following example:

```
10.2.9.2 starfire-iptv
```

Step 7 Save and close the /etc/hosts file.

Note For the host system IP address, it is recommended that you use the IP address of the Ethernet interface that points to the management network and the VBO.

Step 8 Use a text editor to modify the /etc/sysconfig/network file and change HOSTNAME to the hostname of this system, as in the following example:

```
HOSTNAME=starfire-iptv
```

Step 9 Save and close the /etc/sysconfig/network file.

Note The changes to the files /etc/hosts and /etc/sysconfig/network do not take effect until the system is rebooted in Step 10.

Step 10 To restart the system, issue the following command:

```
[root@system]# init 6
```

The operating system restarts.

Configuring Video Navigator

Video Navigator is a web application that requires minimal configuration.

Note This section deals only with the specifics of configuring Video Navigator. For the details of configuring Poster Art Server, see the "Preparing to Configure Poster Art Server" section on page 2-6.

To configure Video Navigator, do the following:

Step 1 Log in as root.

Step 2 Change the working directory as follows:

```
$ cd /home/isa/MIDAS/config
```

Step 3 Ensure that the the following lines are present in the backoffice.properties file:

```
    BackofficeVendor=EventIS

    BackofficeUrl=http://192.124.21.22/
```

Step 4 Save and close the backoffice.properties file.

Preparing to Configure Poster Art Server

Poster Art Server is configured by means of an XML file that must first be uncompressed.

Do the following to uncompress the Poster Art Server configuration file.

Step 1 Go to the directory where the compressed file resides. The following is the default directory when Poster Art Server is installed as part of Video Navigator:

```
cd /home/isa/MIDAS/webapps/
```

Step 2 Uncompress the file.

```
jar xvf paserver.war
```

Note The resulting file is web.xml, in the directory WEB-INF. For an example of this file, see the "web.xml" section on page 5-18.

Step 3 You can now edit the XML file with any text editor.

For details, see Chapter 3, "Understanding and Using the Cisco CDS Poster Art Server," and proceed to the "Configuring Poster Art Server" section on page 3-8.

Configuring the Apache Web Server

You must configure the Apache web server to work with multiple IP addresses. Typically, two CDE110 Ethernet interfaces are configured with IP addresses. One Ethernet interface is for the STB client-facing VLAN, and the other Ethernet interface is for the management- and VBO-facing VLAN.

Do the following to configure the Apache web server:

Step 1 Log in as root or use the **su** command to change to user root.

Step 2 Change the working directory.

```
# cd /usr/local/apache2/bin
```

Step 3 Stop the Apache httpd daemon.

```
# ./apachectl stop
```

Step 4 Change the working directory.

```
# cd /usr/local/apache2/conf
```

Step 5 Use a text editor to modify the httpd.conf file.

a. Search for the following section:

```
# Listen: Allows you to bind Apache to specific IP addresses and/or
# ports, instead of the default. See also the <VirtualHost>
# directive.
#
# Change this to Listen on specific IP addresses as shown below to
# prevent Apache from glomming onto all bound IP addresses.
#
Listen 80
```

b. Replace `Listen 80` with the following:

```
Listen  xxx.xx.xx.xxx:80
Listen  yyy.yy.yy.yyy:80
```

In the preceding, *xxx.xx.xx.xxx* and *yyy.yy.yy.yyy* are the IP addresses that you configured for the two CDE110 Ethernet interfaces.

c. Save and close the httpd.conf file.

Step 6 Change the working directory.

```
# cd /usr/local/apache2/bin
```

Step 7 Start the Apache httpd daemon.

```
# ./apachectl start
```

Starting Video Navigator and Verifying System Status

The following system services are started automatically each time the CDE110 is powered on:

- sshd—Secure Shell daemon
- httpd—HyperText Transfer Protocol daemon (the Apache web server)
- tomcat5—Apache Tomcat application server

This section shows you how to do the following:

1. Manually start Video Navigator when you start the application for the first time.
2. Verify that Video Navigator is running correctly.
3. Configure Video Navigator to start automatically when the CDE110 is powered on or restarted.

Note The name “midas” appears in the Video Navigator commands and directory names. This name was used for Video Navigator when the commands were created.

Do the following to start Video Navigator and verify that the needed processes are running:

Step 1 Log in as root and change to user isa by using the **su - isa** command.

```
su - isa
```

Step 2 Start Video Navigator.

```
$ start_midas

MIDAS not running ....... starting MIDAS
```

Step 3 Verify that the Video Navigator process is running.

```
$ check_midas

MIDAS (2.1.X.X) is running
```

If Video Navigator is not running, the output is as follows:

```
MIDAS is not running
```

Step 4 Test the STB client-facing web services interface.

```
$ cd /home/isa/MIDAS_IntegrationTest

$ clientinterfacetest
```

The **clientinterfacetest** script does not verify the connection from Video Navigator to the STB. It verifies that the client-facing web services interface of Video Navigator is working correctly.

If the test is *successful*, the output is as follows:

```
***************************************************
Start testing liveness of MIDAS client interface.
***************************************************
output = <html><body><h1>Welcome to MIDAS Server</h1></body></html>

Test successful
```

If the test is *not successful*, the output is as follows:

```
***************************************************
Start testing liveness of MIDAS client interface.
***************************************************
Method failed: HTTP/1.1 503 Service Temporarily Unavailable
Test unsuccessful
```

Step 5 Log in as root or use the **su** command to get root privileges.

Step 6 Verify that the sshd process is running and look for output similar to that shown below.

```
# ps -ef | grep sshd

root      2835     1  0 Jul18 ?        00:00:00 /usr/sbin/sshd
```

Step 7 Verify that the httpd process is running and look for output similar to that shown below.

```
# ps -ef | grep httpd

root      2880     1  0 Jul18 ?        00:00:00 /usr/sbin/httpd
apache    4881  2880  0 04:03 ?        00:00:00 /usr/sbin/httpd
apache    4882  2880  0 04:03 ?        00:00:00 /usr/sbin/httpd
apache    4883  2880  0 04:03 ?        00:00:00 /usr/sbin/httpd
apache    4884  2880  0 04:03 ?        00:00:00 /usr/sbin/httpd
apache    4885  2880  0 04:03 ?        00:00:00 /usr/sbin/httpd
apache    4886  2880  0 04:03 ?        00:00:00 /usr/sbin/httpd
apache    4887  2880  0 04:03 ?        00:00:00 /usr/sbin/httpd
apache    4888  2880  0 04:03 ?        00:00:00 /usr/sbin/httpd
```

Step 8 Verify the tomcat5 process is running and look for output similar to that shown below.

```
# ps -ef | grep tomcat5

root      2915     1  0 Jul18 ?        00:00:11 /usr/java/default/bin/java
-Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager
-Djava.util.logging.config.file=/usr/share/tomcat5/conf/logging.properties
-Djava.endorsed.dirs=/usr/share/tomcat5/common/endorsed -classpath
:/usr/share/tomcat5/bin/bootstrap.jar:/usr/share/tomcat5/bin/commons-logging-api.jar
-Dcatalina.base=/usr/share/tomcat5 -Dcatalina.home=/usr/share/tomcat5
-Djava.io.tmpdir=/usr/share/tomcat5/temp org.apache.catalina.startup.Bootstrap start
```

Step 9 Do one of the following after checking that the sshd, httpd, and tomcat5 processes are running:

- If these checks indicate that all processes are running, proceed to Step 10.
- If any of these checks fail, restart the processes that are not running. As an example, to restart the tomcat5 process, use the **service tomcat5 restart** command. Then verify that the processes are running and proceed to Step 10.

Step 10 Change the working directory.

```
# cd /etc
```

Step 11 To configure Video Navigator to start automatically when the CDE110 is powered on or restarts, use a text editor to modify the rc.local file.

a. Uncomment the following line by deleting the # character:

```
#su - isa -c "cd /home/isa/MIDAS; ./run_midas >& /home/isa/MIDAS/midas_log&"
```

b. Save and close the rc.local file.

Configuring for Resiliency with Cisco ACE (Optional)

Video Navigator basically takes HTTP requests from the STB. To provide resiliency, the Cisco Application Control Engine (ACE) module is used to connect to two or more Video Navigator servers. The basic configuration is as follows:

1. Add an extra VLAN for the subscriber-facing ports on the Video Navigator servers.
2. Place the outside port on the ACE module in the subscriber-facing VLAN.
3. Place the inside port of the ACE module on the extra VLAN with the Video Navigator servers.
4. Create a virtual IP address on the ACE module (in the subscriber-facing VLAN).
5. Add each Video Navigator server as a real server to the server farm (in the extra VLAN) associated with the virtual IP address.
6. Configure the ACE module to forward traffic from only port 80 or port 443 to the Video Navigator servers.
7. Monitor those ports periodically to determine whether a real server is available.

The ACE module supports two kinds of aliveness check, port aliveness and application aliveness, as follows:

- To check for port aliveness, ACE periodically pings the desired ports to see whether they are reachable. If the ping succeeds, it is assumed that the remote server is operating properly; otherwise, the server is identified as failed and is not used until the next successful ping.

- To check for application aliveness, the operator must define in ACE the contents of the desired request and response messages. ACE periodically sends the request to the Video Navigator server, and compares the response message with the configured value. If the two match, it is assumed that the remote server is operating properly; otherwise, the server is identified as failed and is not used until the next successful match.

Because subscriber-specific data is queried each time an STB browses the VoD catalog, it is possible that one Video Navigator server in the farm has subscriber-specific data that the others do not.

Caution It is important that browsing sessions from the same STB be directed to the same server in the server farm. The ACE module has a feature that can make a session from a given IP address "sticky" to a particular server. Basically, this ensures that all HTTP packets from a given IP address are directed to the same server for a configurable period of time (usually much longer than an average browsing session).

Note For supporting information, see the references in "Support for Resiliency with Cisco ACE" section on page 1-6.

Information on CDE110 Hardware

For information about the Cisco Content Delivery Engine 110, see the *Cisco Content Delivery Engine 110 Hardware Installation Guide* at the following URL:

http://www.cisco.com/en/US/docs/video/cds/cde/cde110/installation/guide/cde110_install.html

CHAPTER 3

Understanding and Using the Cisco CDS Poster Art Server

This chapter discusses the Cisco Content Delivery System (CDS) Poster Art Server and contains the following major topics:

- Basic Operation, page 3-1
- Poster Art Server Components and Functions, page 3-2
- Implementing Poster Art Server, page 3-5
- Configuring Poster Art Server, page 3-8

For a high-level overview, see "Cisco CDS Poster Art Server Overview" section on page 1-4.

Basic Operation

The basic operation of Poster Art Server is as follows:

1. The poster art client on the STB sends an HTTP GET request to the PAS, in the following example format:

```
http://<host name>:<port>/PAServer/PosterHttpServer/GetPoster?id=<poster
id>&xdim=<xdim>&ydim=<ydim>
```

   where

   - *<host name>* naturally varies
   - *<port>* is optional, and depends on the type of web server that is deployed (for example, Apache, Apache Tomcat)
   - *<poster id>* is the ID of the poster that is defined and established in the CSI request SetAssetRequest for poster ingestion (see Chapter 4, "Using the Cisco Content Storage Interface")
   - *<xdim>* and *<ydim>* are optional parameters (see the "Storing and Resizing Images" section on page 3-9)

   The following is an example request without dimensions specified:

```
http://10.66.4.14/PAServer/PosterHttpServer/GetPoster?id=2babec60-31db-46b8-80a7-231a0
8ce64ee
```

   The following is an example request with dimensions specified:

```
http://10.66.4.14/PAServer/PosterHttpServer/GetPoster?id=2babec60-31db-46b8-80a7-231a0
8ce64ee&xdim=80&ydim=120
```

2. Poster Art Server delivers the image file to the client.

Note Because the VBO synchronizes all poster data, it is possible for the STB to require a poster that is already in the asset repository, but Poster Art Server has not yet downloaded the entire file. As a result, the STB receives incorrect poster data. In this case, Poster Art Server responds with an InvalidPosterId exception. The STB then tries to download another poster.

Poster Art Server Components and Functions

This section discusses the following components and functions:

- Poster Art Server Logical Interfaces
- Role of the TaskManager Module
- Poster States

Poster Art Server Logical Interfaces

Poster Art Server is located between the VBO and the STB client. All the interfaces for Poster Art Server use XML over HTTP. Poster Art Server communicates with external components through two interfaces:

- Cisco Content Storage Interface (CSI)—CSI supports interactions for ingesting and managing poster art on Poster Art Server.
- Cisco Client API (Video Navigator client-facing API)—The client uses this interface to fetch the poster art that is needed.

For details about Cisco CSI, see Chapter 4, "Using the Cisco Content Storage Interface."

Poster Art Server uses the software modules listed in Table 3-1.

Table 3-1 Poster Art Server Software Modules

Module	Description
PosterHTTPServlet and ClientService	Receives the STB GetPoster request, and returns the requested poster data or exception accordingly.
CSIservice	Receives CSI messages from the VBO, and constructs response messages that it sends to the VBO.
ImageStore	Provides a wrapper of the file system; implements image file store and access, as well as space calculation features.
ImageProcessor	Provides image processing features, including resizing.
TaskManager	Manages image ingestion tasks.
Connection	Communicates with the poster data server in the VBO to download data files. This network-level module is responsible for setting up socket connections. downloading data, and detecting network problems.

Note Cisco CSI supports HTTP and FTP. If other protocols need to be supported in the future, a new type connection must be created and inherited from the Connection class, to implement the download method.

Role of the TaskManager Module

The TaskManager module has a task queue to ensure that ingestion commands do not overwhelm the system.

Note The number of parallel ingestion tasks is configurable. See the "Configuring the Number of Ingestion Tasks" section on page 3-10.

The TaskManager uses the following algorithm:

1. When the system starts up, the task queue is empty.
2. When a job needs to be processed, the TaskManager handles it in one of the following three ways:
   a. Use an available task. If an available task is in the task queue, the TaskManager checks it out and uses it.
   b. Create a task. If the number of tasks is less than the configured number, the TaskManager creates a task, puts it into the task queue, and executes it.
   c. Place job in job queue. The TaskManager puts the job into a job queue until an available task can be used.
3. When the task finishes downloading a poster, it actively checks in with the TaskManager for additional assignments.

The TaskManager uses two basic concepts: *task* and *connection*. Tasks are instantiated, but they do not immediately connect to the target server. When a task is implemented, it creates a connection to the target server and downloads the poster. After the download is complete, the TaskManager tears down the connection to free resources on the target server.

Poster States

A poster may be in one of four states: Pending, Transferring, Completed, or Failed. The transition among those states is as follows:

1. When the TaskManager receives an AddAsset or UpdateAsset command from the VBO, the poster object is created with a Pending state.

Note An asset is the poster art and its associated metadata.

2. When the TaskManager assigns a task to ingest the poster, it changes the state to Transferring.
3. If ingestion is successful, the state becomes Completed.
4. When a poster is transferred, a failure can occur. For example, the VBO storage server may be unreachable, or there could be a network or image resizing failure. In this case, the state is changed to Failed.

The states are detailed below.

- The following applies at the Pending state:
  - If Poster Art Server receives an AddAsset or UpdateAsset request whose version is different from an existing one, it responds with an ExistingOperationInProgress exception.
  - If Poster Art Server receives an AddAsset or UpdateAsset request whose version is the same as an existing one, it responds with a VersionAlreadyExists exception.
  - If Poster Art Server receives a DeleteAsset request, it deletes the asset.
- The following applies at the Transferring state:
  - The VBO should not send an AddAsset or UpdateAsset request. If the VBO becomes aware that the state of a poster has been Transferring for an excessive time, it must first delete the asset, and then add it.

Note The time limitation is determined by the client. Before the file is downloaded from the server, Poster Art Server sets the Transferring state with the protocol to be used (for example, FTP). Depending on the size of the file and the speed of the network, the time of this state is eventually resolved by the success or failure of the download, and the state of the process is changed to either Completed or Failed, as appropriate.

  - If Poster Art Server receives an AddAsset/UpdateAsset request for the same asset that is in Transferring state, it responds with an ExistingOperationInProgress exception.
  - If Poster Art Server receives a DeleteAsset request for the same asset that is in Transferring state, it responds with an ExistingOperationInProgress exception.
- The following applies at the Failed or Completed state:
  - If Poster Art Server receives a SetAssetAction request whose version is the same as that of an existing asset, it responds with a VersionAlreadyExists exception.
  - If Poster Art Server receives a SetAssetAction request whose version is different from that of an existing asset, it deletes the old asset and downloads a new one.
  - If Poster Art Server receives a DeleteAsset request, it deletes the asset.

Implementing Poster Art Server

The following topics are presented:

- Installing Poster Art Server
- Configuring Poster Art Server to Work with Video Navigator
- Understanding the Repository
- Designing for Resiliency
- Storing and Resizing Images
- Configuring the Number of Ingestion Tasks

Installing Poster Art Server

CDS Poster Art Server is installed as part of the process that installs CDS Video Navigator. See the "Configuring Video Navigator" section on page 2-5. .

Note For complete information on the Cisco CDE110, see the *Cisco Content Delivery Engine 110 Hardware Installation Guide*.

Configuring Poster Art Server to Work with Video Navigator

Note the following implementation issues:

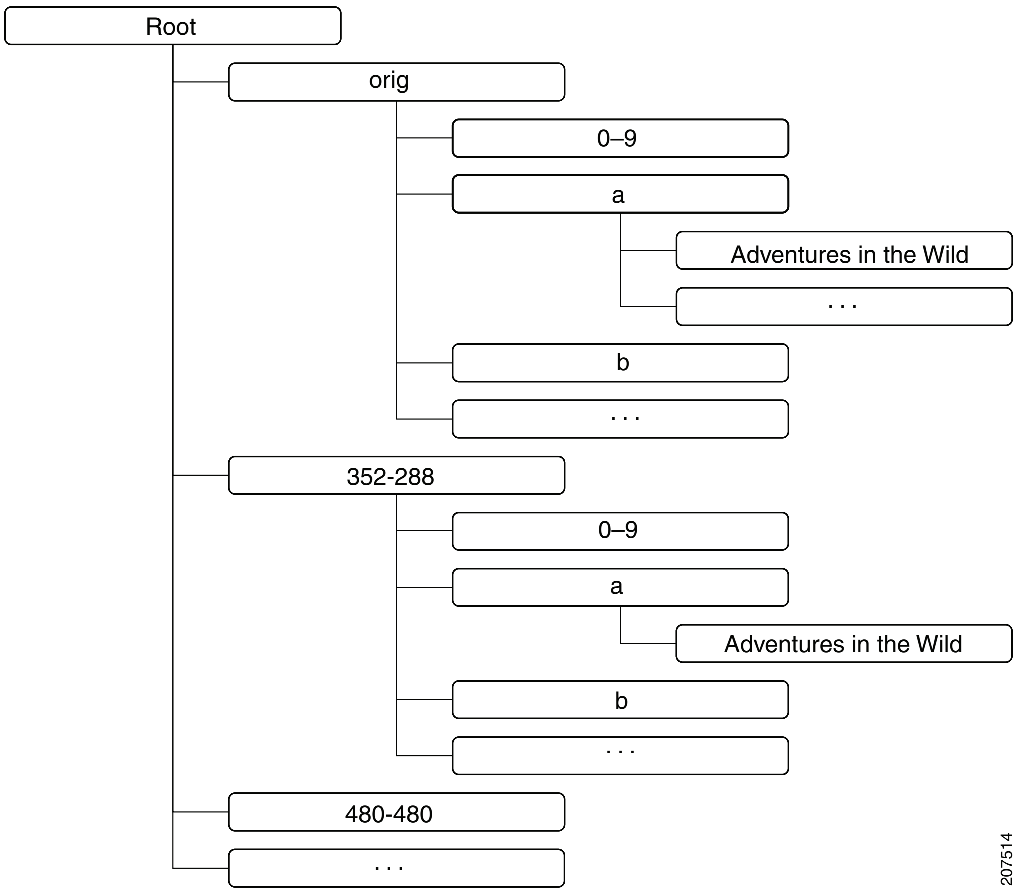
- In the current release, Poster Art Server is designed to coexist with Video Navigator on the same server.
- When requests to Poster Art Server are sent to the IP address of Video Navigator, the web server dispatches separate CSI requests to Video Navigator and to Poster Art Server.
- The interface facing the Video Navigator client can be configured to listen to different ports, so that different STB requests can be handled by different applications.
- Video Navigator stores all data in a database, avsdb, which supports both Video Navigator and Poster Art Server. However, a separate disk partition can be created to store poster data.

Note See the "Storage Database" section on page 3-7.

Understanding the Repository

To allow Poster Art Server to access stored poster files as rapidly as possible, a hierarchical design is used for the poster repository, also known as the image store. The repository is already established and cannot be altered by the operator. The repository hierarchy is shown in Figure 3-1.

Figure 3-1 Repository Hierarchy



- Poster files in the first level, /orig, are separated from resized files. All original ingested files are stored in /orig, no matter what their size is. Resized files are stored in different folders according to their resized dimension.
- The second-level directories are named from a to z and from 0 to 9.
- The poster files are stored in the third level according to the first character of the poster's unique asset ID, which is the name of the file. This prevents duplicate file names if Poster Art Server is used with multiple VBOs in a single deployment.

Note In the case of a failover, the VBO is responsible for synchronizing poster data among multiple servers (see the "VBO-Based Resiliency" section on page 3-8).

Storage Database

Poster information, such as poster ID, poster name, poster version, the number of resized posters, and the name of the resized poster file, are stored in the database. Poster data files and corresponding resized files are stored in the local file system. Table 3-2 details the parameters of the Poster Art Server storage database.

Table 3-2 Poster Art Server Database Parameters

Name	Description	Type	Value	Example
assetId	Unique ID for the poster	String(64)	Any ASCII characters	a188b1dc-376
assetName	Name of the poster (usually the original file name in VBO storage)	String(64)	Any ASCII characters	Mummy3.jpg
Version	Version of the poster	String(16)	Any ASCII characters	75.4.49
Url	URL from which the poster is downloaded	String(1024)	Any ASCII characters	ftp://ingest:ingestqa @192.168.2.22 /poster/1.jpg
size	Size of the poster, in bytes	Integer(8)	Any integer	77532
lastModifiedTime	Last time, in seconds, the poster was modified, in decimal time format	Integer(8)	Any integer	1229719854
Width	Width of the original poster, in pixels	Integer(4)	Any integer	352
Height	Height of the original poster, in pixels	Integer(4)	Any integer	288
State	State of the poster, as defined under Value	Integer(2)	Pending (0): asset transfer process waiting to start	3
			Transferring (1): asset being transferred	
			Completed (3): asset transfer completed	
			Failed (4): asset transfer failed	
Reason	Reason for current state	String(64)	Any ASCII characters	Unknown

Designing for Resiliency

Poster Art Server communicates with external components through two interfaces: a CSI interface to the VBO and one to the STB client. (See Figure 1-3 on page 1-5.) For the latter, the optional ACE-based application resiliency feature can be used to implement resiliency. For the former, VBO-based resiliency is required.

ACE-Based Application Resiliency

The basics of configuration for support through Cisco ACE are discussed in the "Configuring for Resiliency with Cisco ACE (Optional)" section on page 2-9.

VBO-Based Resiliency

When Poster Art Server is restarted or a new Poster Art Server server is added into the VoD system, a sign-on is executed (see the "Updating CDS Applications with the Latest Assets upon Sign-On and Registration" section on page 4-23), and the VBO gets the information regarding all posters, compares their versions, and decides whether to download missing poster files. This maintains data synchronization across all servers.

If there is a VBO failure for an unknown reason while a poster is being downloaded, the poster ingestion fails and the download is not resumed—even if the VBO starts up again. The VBO must re-ingest the poster, in which case the state of the poster asset is not complete. Therefore, when the VBO gets the list of all posters, it notices the incomplete download and sends an AddAsset request to Poster Art Server.

Configuring Poster Art Server

To configure Poster Art Server, the operator controls the behavior of the application by editing the web.xml file. See "Preparing to Configure Poster Art Server" section on page 2-6.

Note This file can reside anywhere. Access Video Navigator through the appropriate IP address and upload the file to Video Navigator. For an example of the web.xml file, see the "web.xml" section on page 5-18.

Table 3-3 lists and describes the available parameters for configuring how Poster Art Server operates.

Table 3-3 Configuration Parameters for Poster Art Server Operation

Parameter	Description	Values	Default
PAServerIP	IP address of Poster Art Server CSI application	String	127.0.0.1
PAServerPort	Port of Poster Art Server CSI application	8000–64000	8088
MaxRetry	Maximum number of retries if poster retrieval fails	1–10	3
RetryInterval	Interval between consecutive retries, in seconds	1–60	3
TimeOut	Timeout for VBO connection, in seconds	1–60	3
ResizeOption	Image resize options, in width *x* height	352x288, 480x480. 600x600	See values for ResizeOption in the "web.xml" section on page 5-18. You can add and edit those values as appropriate, keeping the integer format as shown.
ImageStoreRootPath	Root path of the image store (see the "Understanding the Repository" section on page 3-6)	String	/home/isa/paserver/store

In addition to the operation configurations, there is also a system configuration option for tuning Poster Art Server, as shown in Table 3-4 on page 3-9.

Table 3-4 System Configuration Option for Tuning Poster Art Server

Name	Description	Values	Default
TaskNumber	Number of parallel ingestion tasks	1–128	30

Note Before the operator and system configurations can take effect, the system must be restarted. See the "Starting Video Navigator and Verifying System Status" section on page 2-7.

Storing and Resizing Images

Poster Art Server supports the scaling and resizing of images in JPEG format (.jpg) only. (To improve performance, resized images are cached.) The result is an image with 8-bit RGB color components, corresponding to a Windows- or Solaris-style color model, with the colors red, green, and blue packed into integer pixels.

Image resizing works as follows:

- The operator uses the parameter ResizeOption (see Table 3-3 on page 3-8), and the application resizes the poster when the poster is downloaded for the first time. Poster Art Server stores the resized file in an appropriate directory (see "Understanding the Repository" section on page 3-6).
- Alternatively, in the case where ResizeOption is not used, the application does not store a resized file in the image repository.

Note Typically, Poster Art Server does not expect the STB to request a file whose scale has not been configured. To ensure high availability, the operator must use ResizeOption to meet the requirements of different customer equipment—SD TV, HD TV, PC—before poster images are ingested into Poster Art Server.

- When a poster is deleted, the application looks in the "resize" directory and deletes any resized posters.
- When the operator changes a resizing parameter, the change takes effect for new posters only; existing posters are not resized again. Alternatively, the operator updates an existing poster by creating a new version, in which case the application resizes it.

Note The number of posters that can be stored depends on the capacity of the server.

Table 3-5 presents some examples for the number of JPEG poster titles of given sizes that can be stored. Additional storage is needed for each cached image.

Table 3-5 Example Poster Storage Sizes

	Size, KB		
Example	**Low**	**High**	**Average**
320 x 240 JPEG	3.1	26	14.6
80 x 60 JPEG	1.2	7.7	4.5
	Size, MB		
Number of titles	**Low**	**High**	**Average**
1000	5	33	19
5000	22	165	93
10000	43	330	186
25000	106	823	464

Configuring the Number of Ingestion Tasks

You can configure the number of ingestion tasks by changing the value for <*param-value*> in the TaskNum component of the file web.xml. The default is currently 30. The following example changes it to 35.

```
<context-param>
        <param-name>TaskNum</param-name>
        <param-value>35</param-value>
    </context-param>
```

CHAPTER 4

Using the Cisco Content Storage Interface

This chapter discusses the Cisco Content Storage Interface (CSI), and presents the following major topics:

Cisco CSI Overview

The basic functionality of Cisco CSI is introduced in the "Cisco Content Storage Interface Overview" section on page 1-5. The CSI uses detailed metadata for the VOD catalog in conformance with the CableLabs Video-On-Demand Content Specification, Version 1.1.

Cisco CSI Data Types

This section describes the following data types used with the Cisco CSI, presents their schemas, and defines their elements and attributes:

AssetId

AssetId represents the ID of an asset and is unique for each AssetType.

Schema

```
<xs:simpleType name="AssetId">
     <xs:restriction base="xs:string">
<xs:minLength value="1" />
<xs:maxLength value="128" />
<xs:pattern value="[a-zA-Z0-9-_]+" />
     </xs:restriction>
</xs:simpleType>
```

Element	Type	Description
AssetId	xs:string	String from 1 to 128 characters

AssetInfoType

AssetInfoType provides a detailed description of an asset.

Schema

```
<xs:complexType name="AssetInfoType">
    <xs:attribute name="id"         type="tns:AssetId"use="required" />
    <xs:attribute name="state"     type="xs:int"use="required" />
      <xs:attribute name="reason" use="optional">
         <xs:simpleType>
           <xs:restriction base="xs:string">
              <xs:minLength value="1" />
              <xs:maxLength value="64" />
           </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
    <xs:attribute name="version" type="xs:string"use="optional" />
    <xs:attribute name="size"      type="xs:long"use="optional" />
</xs:complexType>
```

Attribute	Type	Description
id	AssetId	Unique ID or name of the asset (for example, funny.jpg) within a specific AssetType
state	xs:int	Possible states: • Pending (0)—Asset transfer process has not started. • Transferring (1)—Asset is being transferred. • Transferring-Playable (2)—Asset is being transferred but a PLAY request can start. Applicable only to AssetType:content. • Completed (3)—Asset transfer has completed. • Failed (4)—Asset transfer has failed. • NonExist (5)—Asset does not exist in the system.
reason	xs:string	Reason why asset is in current state. Not used if state = NonExist. String from 1 to 64 characters.
version	xs:string	Not used if state = NonExist.
size	xs:long	Size of asset in Kbytes. Not used if state = NonExist.

AssetStatusType

AssetStatusType represents the state of an asset.

Schema

```
<xs:complexType name="AssetStatusType">
    <xs:attribute name="id"        type="tns:AssetId" use="required" />
    <xs:attribute name="state"     type="xs:int"      use="required" />
      <xs:attribute name="reason" use="optional">
         <xs:simpleType>
           <xs:restriction base="xs:string">
              <xs:minLength value="1" />
              <xs:maxLength value="64" />
           </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
</xs:complexType>
```

Attribute	Type	Description
id	AssetId	Unique ID or name of the item within a specific AssetType
state	xs:int	Possible states: • Pending (0)—Asset transfer process has not been started. • Transferring (1)—Asset is being transferred. • Transferring-Playable (2)—Asset is being transferred but a PLAY request can start. Applicable only to AssetType:content. • Completed (3)—Asset transfer has completed. • Failed (4)—Asset transfer has failed. • NonExist (5)—Asset does not exist in the system.
reason	xs:string	Reason why the assset is in the current state. Not used if state = NonExist. String from 1 to 64 characters.

AssetType

AssetType represents the type of asset.

Schema

```
<xs:simpleType name="AssetType">
     <xs:restriction base="xs:string">
<xs:enumeration value="catalog" />
<xs:enumeration value="poster" />
<xs:enumeration value="content" />
     </xs:restriction>
</xs:simpleType>
```

Element	Type	Description
AssetType	xs:string	Type of the asset. Possible values: • catalog • poster • content

CallbackAddressType

CallbackAddressType contains information about where the CDS should send the callback message.

Schema

```
<xs:complexType name="CallbackAddressType">
    <xs:attribute name="ipAddress"   type="xs:string"           use="required" />
    <xs:attribute name="port"        type="xs:unsignedShort "   use="required" />
    <xs:attribute name="path"        type="xs:string"           use="required" />
</xs:complexType>
```

Attribute	Type	Description
ipAddress	xs:string	IP address specifying the video backoffice (VBO) to which the CDS should send the callback message. Address can be in dotted decimal notation (IPv4) or colon-separated format (IPv6).
port	xs:unsignedShort	Number of the VBO port to which the CDS should send the callback message
path	xs:string	Full pathname specifying where to send the callback message

CallbackMessageType

CallbackMessageType describes the callback message made to the specified VBO location (CallbackAddressType) after the requested operation is finished. Each callback message contains status for only one asset.

Schema

```
<xs:complexType name="CallbackMessageType">
    <xs:attribute name="type"    type="tns:AssetType"   use="required" />
    <xs:attribute name="id"      type="tns:AssetId"     use="required" />
    <xs:attribute name="status"  type="xs:string"       use="required" />
    <xs:attribute name="reason"  type="xs:string"       use="required" />
</xs:complexType>
```

Attribute	Type	Description
type	AssetId	Type of asset
id	AssetType	Unique ID or name of the item within a specific AssetType
status	xs:string	Result of the operation. Possible values: • succeed • failed
reason	xs:string	Reason for the result

ErrorResponseType

ErrorResponseType describes the error message that is returned if there is problem servicing a request.

Schema

```
<xs:complexType name="ErrorResponseType">
       <xs:attribute name="code" type="xs:string" />
       <xs:attribute name="reason" type="xs:string" />
</xs:complexType>
```

Attribute	Type	Description
code	xs:string	Error code (see Table 4-1)
reason	xs:string	Reason for the error (see Table 4-1)

Table 4-1 lists error codes and associated reasons.

Table 4-1 CSI Error Codes and Reasons

Error Code	Reason
InvalidAssetType	Asset type specified is invalid
InvalidId	ID is in invalid format
VersionAlreadyExists	Version of this asset already exists in the system
VersionDoesNotExist	Version specified does not exist in the system
InvalidTransferProtocol	Invalid transfer protocol specified
InvalidAddressType	Invalid callback address specified
InternalError	Internal processing error
VBORepositoryError	VBO repository error, cannot access data
ExistingOperationInProgress	Operation on the asset is currently in progress
InvalidXMLFormat	XML data in the message is incorrectly formed
InvalidContentFormat	Content (poster and movie) format is not valid

TransferProtocolType

TransferProtocolType describes the protocol to be used for fetching assets.

Schema

```
<xs:complexType name="TransferProtocolType">
    <xs:attribute name="protocol"     type="xs:string"          use="required" />
    <xs:attribute name="ipAddress"    type="xs:string"          use="required" />
    <xs:attribute name="port"         type="xs:unsignedShort"   use="required" />
    <xs:attribute name="userName"     type="xs:string"          use="optional" />
    <xs:attribute name="password"     type="xs:string"          use="optional" />
    <xs:attribute name="path"         type="xs:string"          use="required" />
 </xs:complexType>
```

Attribute	Type	Description
protocol	xs:string	Possible values: • FTP • HTTP GET
ipAddress	xs:string	IP address in dotted decimal notation (IPv4) or colon-separated format (IPv6)
port	xs:unsignedShort	Port number
username	xs:string	Used for certain protocols (FTP)
password	xs:string	Used for certain protocols (FTP)
path	xs:string	Full pathname to the asset, for example, ~/posters/example_1.jpg

Cisco CSI Requests and Responses

This section describes the following requests and responses for Cisco CSI:

Note The namespace for the above requests and responses is com.cisco.csi.

DeleteAsset

DeleteAsset requests the deletion of an asset.

Note The request is carried out asynchronously. A requester wanting to be notified when the operation is completed should register for a callback notification by using the callback element.

Message Summary

Message	XML Element
Request	DeleteAssetRequest
Response	DeleteAssetResponse

Request

Element: DeleteAssetRequest

Schema:

```
<xs:element name="DeleteAssetRequest">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="type" type="tns:AssetType" />
        <xs:element name="id" type="tns:AssetId" />
        <xs:element name="version" type="xs:string" />
        <xs:element minOccurs="0" name="callback" type="tns:CallbackAddressType" />
      </xs:sequence>
    </xs:complexType>
</xs:element>
```

Element	Type	Description
type	AssetType	Type of asset
id	AssetId	Unique ID of the asset. Maximum length is set to 128.
version	xs:string	Version of the asset
callback	CallbackAddressType	Callback address. Notification is made to the address provided after the delete asset process is finished. This parameter is optional. Without it, no notification is made. **Note** The requester can request only one callback location.

Example Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DeleteAssetRequest xmlns="com.cisco.csi">
    <type>poster</type>
    <id>1234567</id>
    <version>1.1</version>
</DeleteAssetRequest>
```

Response

Element: DeleteAssetResponse

Schema:

```
<xs:element name="DeleteAssetResponse">
     <xs:complexType>
      <xs:sequence>
        <xs:element name="status" type="tns:AssetStatusType" />
       </xs:sequence>
     </xs:complexType>
</xs:element>
```

Element	Type	Description
status	AssetStatusType	Status of an asset. Possible values: • will get state—pending • exists—false with this operation

Example Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DeleteAssetResponse xmlns="com.cisco.csi">
    <status reason="Pending" state="0" exists="false" id="1234567"/>
</DeleteAssetResponse>
```

Error Codes

- InvalidAssetType
- InvalidXMLFormat
- InvalidContentFormat
- InvalidId
- VersionDoesNotExist
- InvalidTransferProtocol
- InvalidCallbackAddress
- InternalError

Note For definitions of the above error codes, see Table 4-1 on page 4-7.

GetAssetCount

GetAssetCount returns the number of the requested asset type that exists in the system.

Message Summary

Message	XML Element
Request	GetAssetCountRequest
Response	GetAssetCountResponse

Request

Element: GetAssetCountRequest

Schema:

```
<xs:element name="GetAssetCountRequest">
     <xs:complexType>
          <xs:sequence>
              <xs:element name="type" type="tns:AssetType" />
          </xs:sequence>
     </xs:complexType>
</xs:element>
```

Element	Type	Description
type	AssetType	Type of asset

Example Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<GetAssetCountRequest xmlns="com.cisco.csi">
    <type>poster</type>
</GetAssetCountRequest>
```

Response

Element: GetAssetCountResponse

Schema:

```
<xs:element name="GetAssetCountResponse">
      <xs:complexType>
      <xs:sequence>
        <xs:element name="count" type="xs:int" />
      </xs:sequence>
      </xs:complexType>
</xs:element>
```

Element	Type	Description
count	xs:int	Number of assets

Example Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<GetAssetCountResponse xmlns="com.cisco.csi">
    <count>2</count>
</GetAssetCountResponse>
```

Error Codes

- InvalidAssetType
- Invalid XMLFormat
- InternalError

Note For definitions of the above error codes, see Table 4-1 on page 4-7.

GetAssetInventoryList

GetAssetInventoryList returns an inventory list for the requested asset type.

Message Summary

Message	XML Element
Request	GetAssetInventoryListRequest
Response	GetAssetInventoryListResponse

Request

Element: GetAssetInventoryListRequest

Schema:

```
<xs:element name="GetAssetInventoryListRequest">
      <xs:complexType>
      <xs:sequence>
        <xs:element name="type" type="tns:AssetType" />
      </xs:sequence>
      </xs:complexType>
</xs:element>
```

Element	Type	Description
type	AssetType	Type of asset

Example Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<GetAssetInventoryListRequest xmlns="com.cisco.csi">
    <type>poster</type>
</GetAssetInventoryListRequest>
```

Response

Element: GetAssetInventoryListResponse

Schema:

```
<xs:element name="GetAssetInventoryListResponse">
    <xs:complexType>
      <xs:sequence>
            <xs:element minOccurs="0" maxOccurs="unbounded" name="info"
type="tns:AssetInfoType" />
      </xs:sequence>
    </xs:complexType>
</xs:element>
```

Element	Type	Description
type	AssetInfoType	Information about the type of asset

Example Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<GetAssetInventoryListResponse xmlns="com.cisco.csi">
    <info size="84000" version="1.1" reason="Pending" state="0" id="1234567"/>
    <info size="44000" version="1.1" reason="Completed" state="3" id="7654321"/>
</GetAssetInventoryListResponse>
```

Error Codes

- InvalidAssetType
- InvalidXMLFormat
- InternalError

Note For definitions of the above error codes, see Table 4-1 on page 4-7.

GetAssetsInfo

GetAssetsInfo returns the description of the requested asset.

Message Summary

Message	XML Element
Request	GetAssetsInfoRequest
Response	GetAssetsInfoResponse

Request

Element: GetAssetsInfoRequest

Schema:

```
<xs:element name="GetAssetsInfoRequest">
      <xs:complexType>
      <xs:sequence>
        <xs:element name="type" type="tns:AssetType" />
        <xs:element maxOccurs="unbounded" name="id" type="tns:AssetId" />
      </xs:sequence>
      </xs:complexType>
</xs:element>
```

Element	Type	Description
type	AssetType	Type of asset
id	AssetId	Unique ID of the asset. Maximum length is set to 128.

Example Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<GetAssetsInfoRequest xmlns="com.cisco.csi">
    <type>poster</type>
    <id>1234567</id>
    <id>7654321</id>
</GetAssetsInfoRequest>
```

Response

Element: GetAssetsInfoResponse

Schema:

```
<xs:element name="GetAssetsInfoResponse">
      <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="info" type="tns:AssetInfoType" />
      </xs:sequence>
      </xs:complexType>
</xs:element>
```

Element	Type	Description
info	AssetInfoType	Information about the asset

Example Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<GetAssetsInfoResponse xmlns="com.cisco.csi">
    <info size="84000" version="1.1" reason="Pending" state="0" id="1234567"/>
    <info size="44000" version="1.1" reason="Completed" state="3" id="7654321"/>
</GetAssetsInfoResponse>
```

Error Codes

- InvalidAssetType
- InvalidXMLFormat
- InvalidId
- InternalError

Note For definitions of the above error codes, see Table 4-1 on page 4-7.

GetAssetsStatus

GetAssetsStatus returns the state of the requested asset.

Message Summary

Message	XML Element
Request	GetAssetsStatusRequest
Response	GetAssetsStatusResponse

Request

Element: GetAssetsStatusRequest

Schema:

```
<xs:element name="GetAssetsStatusRequest">
      <xs:complexType>
      <xs:sequence>
        <xs:element name="type" type="tns:AssetType" />
        <xs:element maxOccurs="unbounded" name="id" type="tns:AssetId" />
      </xs:sequence>
      </xs:complexType>
</xs:element>
```

Element	Type	Description
type	AssetType	The type of asset
id	AssetId	Unique ID of the asset. Maximum length is set to 128.

Example Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<GetAssetsStatusRequest xmlns="com.cisco.csi">
    <type>poster</type>
    <id>1234567</id>
    <id>7654321</id>
</GetAssetsStatusRequest>
```

Response

Element: GetAssetsStatusResponse

Schema:

```
<xs:element name="GetAssetsStatusResponse">
     <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="status" type="tns:AssetStatusType" />
      </xs:sequence>
     </xs:complexType>
</xs:element>
```

Element	Type	Description
status	AssetStatusType	Status of an asset. Possible values: • If exists = false, the asset is NOT in the CDS system. • If exists = true, the asset is in the CDS system.

Example Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<GetAssetsStatusResponse xmlns="com.cisco.csi">
    <status reason="Pending" state="0" exists="true" id="1234567"/>
    <status reason="Completed" state="3" exists="false" id="7654321"/>
</GetAssetsStatusResponse>
```

Error Codes

- InvalidAssetType
- InvalidXMLFormat
- InvalidId
- InternalError

Note For definitions of the above error codes, see Table 4-1 on page 4-7.

GetAvailableSpace

GetAvailableSpace returns the quantity of space available for an asset type.

Message Summary

Message	XML Element
Request	GetAvailableSpaceRequest
Response	GetAvailableSpaceResponse

Request

Element: GetSpaceAvailableRequest

Schema:

```
<xs:element name="GetAvailableSpaceRequest">
      <xs:complexType>
      <xs:sequence>
        <xs:element name="type" type="tns:AssetType" />
      </xs:sequence>
      </xs:complexType>
</xs:element>
```

Element	Type	Description
type	AssetType	Type of asset

Example Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<GetAvailableSpaceRequest xmlns="com.cisco.csi">
    <type>poster</type>
</GetAvailableSpaceRequest>
```

Response

Element: GetAvailableSpaceResponse

Schema:

```
<xs:element name="GetAvailableSpaceResponse">
    <xs:complexType>
         <xs:sequence>
             <xs:element name="totalSpace" type="xs:long" />
           <xs:element name="freeSpace" type="xs:long" />
         </xs:sequence>
    </xs:complexType>
</xs:element>
```

Element	Type	Description
totalSpace	xs:long	Total space in kilobytes
freeSpace	xs:long	Free space in kilobytes

Example Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<GetAvailableSpaceResponse xmlns="com.cisco.csi">
    <totalSpace>200000</totalSpace>
    <freeSpace>100000</freeSpace>
</GetAvailableSpaceResponse>
```

Error Codes

- InvalidAssetType
- InvalidXMLFormat
- InternalError

Note For definitions of the above error codes, see Table 4-1 on page 4-7.

SetAssetAction

SetAssetAction requests the ingestion or update of an asset. The CDS component uses the protocol provided to fetch the asset.

Note The request is carried out asynchronously. A requester wanting to be notified when the operation is completed should register for a callback notification by using the callback element.

Message Summary

Message	XML Element
Request	SetAssetRequest
Response	SetAssetResponse

Request

Element: SetAssetRequest

Schema:

```
<xs:element name="SetAssetRequest">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="type"    type="tns:AssetType" />
        <xs:element name="id"        type="tns:AssetId" />
        <xs:element name="version"  type="xs:string" />
        <xs:element name="protocol" type="tns:TransferProtocolType" />
        <xs:element minOccurs="0" name="callback"  type="tns:CallbackAddressType" />
      </xs:sequence>
    </xs:complexType>
</xs:element>
```

Element	Type	Description
type	AssetType	Type of asset
id	AssetId	Unique ID of the asset. Maximum length is set to 128.
version	xs:string	Version of the asset
protocol	TransferProtocolType	Transfer protocol for getting the image
callback	CallbackAddressType	Callback address. Notification is made to the address provided after the add asset process is finished. This parameter is optional. Without it, no notification is made. **Note** The requester can request only one callback location.

Example Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SetAssetRequest xmlns="com.cisco.csi">
    <type>poster</type>
    <id>1234567</id>
    <version>1.1</version>
    <protocol path="/poster" password="ingestqa" userName="ingest" port="21"
ipAddress="192.168.1.88" protocol="ftp"/>
    <callback path="/callback" port="1234" ipAddress="192.168.1.100"/>
</SetAssetRequest>
```

Response

Element: SetAssetResponse

Schema:

```
<xs:element name="SetAssetResponse">
     <xs:complexType>
      <xs:sequence>
        <xs:element name="status"  type="tns:AssetStatusType" />
      </xs:sequence>
     </xs:complexType>
</xs:element>
```

Element	Type	Description
status	AssetStatusType	Status of an asset. Possible values: • will get state—pending • exists—true with this operation

Example Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SetAssetResponse xmlns="com.cisco.csi">
    <status reason="Pending" state="0" exists="false" id="1234567"/>
</SetAssetResponse>
```

Error Codes

- InvalidAssetType
- InvalidXMLFormat
- InvalidContentFormat
- InvalidId
- VersionAlreadyExist
- InvalidTransferProtocol
- InvalidCallbackAddress
- ExistingOperationInProgress
- VBORepositoryError
- InternalError

Note For definitions of the above error codes, see Table 4-1 on page 4-7.

Using the Cisco CSI to Update CDS Applications

This section provides the following detailed use cases for the Cisco Content Storage Interface:

- Updating CDS Applications with the Latest Assets upon Sign-On and Registration, page 4-23
- Updating CDS Applications with the Catalog and Poster Art, page 4-24

Updating CDS Applications with the Latest Assets upon Sign-On and Registration

This section describes a typical interaction used for updating Cisco CDS applications (Video Navigator or Poster Art Server) when the application first signs on or registers with the VBO. Requests from the VBO for the CDS application to add or delete an asset are performed asynchronously. If the VBO requires notification that the add or delete operation has completed, the VBO must provide callback information in the add or delete request.

The following is a sequence of typical CSI API interactions between the VBO and the CDS application when the application registers (signs on) with the VBO.

1. When the application starts up, it sends a **SignOn** request to the VBO.
2. The VBO uses **GetAssetInventoryList** to get an inventory list of the appropriate asset type (for example, catalog or poster) from the application. The response from the application includes asset information such as ID, current state, version, and size.
3. In a loop operation, the VBO uses **DeleteAsset** to request that the application delete assets that are not part of the internal list of the VBO. Using the asset ID specified in the request, the application starts processing the delete request and then sends a **DeleteAssetResponse** to the VBO.
4. Optionally, as part of the loop operation, the application performs a callback to notify the VBO that the delete operation is completed. If a callback is used, the details of how to perform the callback are provided by the VBO in a **DeleteAssetRequest**.
5. In a loop operation, the VBO uses **SetAsset** to request that the application add missing assets. The application starts processing the request and then sends a **SetAssetResponse** to the VBO.
6. The application fetches the asset by using information provided in the VBO **SetAssetRequest**, such as the protocol, IP address, and port number.
7. Optionally, as part of the loop operation, the application performs a callback to notify the VBO that the add operation is completed. If a callback is used, the details of how to perform the callback are provided by the VBO in the **SetAssetRequest**.
8. In a loop operation, the VBO uses **SetAsset** to request that the application update assets that are out of date. The application starts processing the update request and then sends a **SetAssetResponse** to the VBO. (The update process is very similar to the add process, but includes deleting each out-of-date asset.)
9. The application fetches the updated asset by using information provided in the **SetAssetRequest**.
10. Optionally, as part of the loop operation, the component performs a callback to notify the VBO that the update operation is completed. If a callback is used, the details of how to perform the callback are provided by the VBO in the **SetAssetRequest**.

Updating CDS Applications with the Catalog and Poster Art

This section describes a typical interaction used in updating the catalog in Video Navigator and updating the poster art in Poster Art Server. The following conditions must be met before an update can take place:

- Filename of the asset detail data is *AssetID*.xml, where *AssetID* is a unique asset ID.
- Filename of the product detail data is *ProductID*.xml, where *ProductID* is a unique product ID.
- Filename of the poster is defined in *AssetID*.xml.
- When the asset detail information is updated in the asset detail file, the asset version is updated in the catalog file.
- When the product detail information of the asset is updated in the product detail file, the asset version is updated in the catalog file.
- Asset detail files, product detail files, and posters specified by one catalog file are located in the same directory as that catalog file.
- Transport protocol of the asset detail files, the product detail files, and the posters is the same as the transport protocol for the catalog.
- In a poster update, because the poster's version is embedded in the response of the GetAssetInventoryList request, the VBO can determine whether posters need to be updated. It is assumed that Poster Art Server always downloads the poster when it receives an AddAsset request.

The following is a sequence of typical interactions between the VBO and the CDS application during the catalog and poster art updates:

1. The VBO uses SetAssetAction to request that Video Navigator add the catalog. As part of this request, the VBO provides the asset type, ID, and version, as well as the protocol to use for the transfer. If a callback is used, the details for the callback are provided by the VBO in the SetAssetAction request.
2. Video Navigator downloads the catalog through the transport protocol defined in the SetAssetAction request.
3. Video Navigator parses the catalog file, and then compares the asset version with the version that currently exists in the Video Navigator system, as follows:
   a. If the asset version in the catalog is newer than the one in the Video Navigator system, Video Navigator downloads the asset. If the asset version is not newer than the one already in the system, Video Navigator processes the next asset.
   b. If the asset version in the catalog is newer than the one in the system, Video Navigator downloads *AssetID*.xml first, parses the metadata, and then stores the metadata in the database for future retrieval.
   c. If the asset version in the catalog is newer than the one in the system, Video Navigator also downloads, parses, and stores *ProductID*.xml.
4. After processing all of the assets, Video Navigator uses a callback to notify the VBO that ingestion is completed. The use of a callback is optional.

The sequence for updating a poster art asset on Poster Art Server is as follows:

Note To ensure that the most recent poster art is available, the VBO checks the poster art version before using SetAssetAction.

1. The VBO uses SetAssetAction to request that Poster Art Server add a poster. As part of SetAssetAction, the VBO provides the asset type, ID, and version, as well as the protocol to use for the transfer. If a callback is used, the details for the callback are provided by the VBO in the SetAssetAction request.
2. Poster Art Server downloads the poster through the transport protocol defined in the SetAssetAction request.
3. After the poster is downloaded and resized, Poster Art Server uses a callback to notify the VBO that ingestion is completed. The use of a callback is optional.

VBO Registration APIs

This section documents a reference registration interface provided by VBO vendors. Cisco CDS applications such as Video Navigator and Poster Art Server use this interface to register and deregister themselves with the VBO. The following registration APIs are presented:

- SignOn, page 4-25
- SignOff, page 4-27
- XSD for SignOn and SignOff Request and Response Pair, page 4-28

Note The namespace for the VBO registration API is defined by the VBO.

SignOn

Message Summary

Message	XML Element
Request	SignOnRequest
Response	SignOnResponse

Request

Element: SignOnRequest

Schema:

```
<element name="SignOnRequest">
        <attribute name="ipAddress"  type="xs:string"  use="required"/>
        <attribute name="port"  type="xs:string"  use="required"/>
        <attribute name="name"  type="xs:string"  use="optional"/>
        <attribute name="type"  type="AssetType"  use="required"/>
</element>
```

Element	Type	Description
ipAddress	xs:string	Dotted-decimal notation (IPv4) or colon-separated (IPv6) format
port	xs:int	Port number
name	xs:string	Name of the CDS application
type	AssetType	Asset type of interest to the CDS application

Example Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SignOnRequest xmlns="com.cisco.csi" type="poster" name="poster art server" port="6666"
ipAddress="192.168.1.66"/>
```

Response

Element: SignOnResponse

Schema:

```
<element name="SignOnResponse" />
   <complexType>
      <sequence>
            <element name="status" type="xs:string" minOccurs="1" maxOccurs="1">
<simpleType>
                  <restriction base="xs:string" />
                            <enumeration  value="Success" />
                            <enumeration  value="Failed" />
                            <enumeration  value="InvaidParameters" />
                   </restriction />
          </simpleType>
</element>
      </sequence>
   </complexType>
</element>
```

Element	Type	Description
status	xs:string	Status of the response to the SignOn request. Possible responses: • Success • Failed • Invalid parameters

Example Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SignOnResponse xmlns="com.cisco.csi">
    <status>Success</status>
</SignOnResponse>
```

SignOff

Message Summary

Message	XML Element
Request	SignOffRequest
Response	SignOffResponse

Request

Element: SignOffRequest

Schema:

```
<element name="SignOffRequest">
        <attribute name="ipAddress"  type="xs:string"  use="required"/>
        <attribute name="port"  type="xs:int"  use="required"/>
        <attribute name="name"  type="xs:string"  use="optional"/>
</element>
```

Element	Type	Description
ipAddress	xs:string	Dotted-decimal notation (IPv4) or colon-separated (IPv6) format
port	xs:int	Port number
name	xs:string	Name of the CDS application

Example Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SignOffRequest xmlns="com.cisco.csi" name="poster art server" port="6666"
ipAddress="192.168.1.66"/>
```

Response

Element: SignOffResponse

Schema:

```
<element name="SignOffResponse" />
   <complexType>
      <sequence>
            <element name="status" type="xs:string" minOccurs="1" maxOccurs="1">
<simpleType>
                  <restriction base="xs:string" />
                            <enumeration  value="Success" />
                            <enumeration  value="Failed" />
                            <enumeration  value="InvaidParameters" />
                   </restriction />
          </simpleType>
</element>
      </sequence>
   </complexType>
</element>
```

Element	Type	Description
status	xs:string	Status of the response to the SignOff request. Possible responses: • Success • Failed • Invalid parameters

Example Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SignOffResponse xmlns="com.cisco.csi">
    <status>Success</status>
</SignOffResponse>
```

XSD for SignOn and SignOff Request and Response Pair

The following schema is a sample XSD for the SignOn and SignOff Request and Response pair.

Schema

```
<xs:element name="SignOnRequest">
    <xs:complexType>
      <xs:attribute name="ipAddress" type="xs:string" />
      <xs:attribute name="port" type="xs:int" />
      <xs:attribute name="name" type="xs:string" use="optional" />
      <xs:attribute name="type" type="tns:AssetType" />
    </xs:complexType>
  </xs:element>
  <xs:element name="SignOnResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="status" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="SignOffRequest">
    <xs:complexType>
      <xs:attribute name="ipAddress" type="xs:string" />
      <xs:attribute name="port" type="xs:int" />
      <xs:attribute name="name" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <xs:element name="SignOffResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="status" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
</xs:element>
```

Required ADI Metadata in Catalog

This section provides information related to the required Asset Distribution Interface (ADI) metadata in the catalog. The ADI is the means by which assets, both content and metadata describing the content, are transported from the provider to the asset management system (AMS). The ADI specification defines the logical organization of metadata structures (Group Asset, Metadata Asset, Content Asset), as well as the operations that put content assets in context with one or more service offerings.

Note See the following CableLabs documents:

CableLabs Asset Distribution Interface Specification Version 1.1

CableLabs Video-On-Demand Content 1.0 Specification

Table 4-2 shows the specific ADI metadata in the catalog, with abbreviations and definitions as listed in Table 4-3 on page 4-30.

Table 4-2 Required ADI Metadata

Name	Type	Specification	Required by ADI	Required by VN
Actors	Title	MOD/SVOD	O	Y
Actors_Display	Title	MOD/SVOD	O	Y
Advisories	Title	MOD/SVOD	O	N
Audience	Title	MOD/SVOD	O	Y
Audio_Type	Movie	MOD/SVOD	R	Y
Audio_Type	Preview	MOD/SVOD	R	N
Chapter	Title	MOD/SVOD	O	N
Country_of_Origin	Title	MOD/SVOD	O	N
Director	Title	MOD/SVOD	O	Y
Display_Run_Time	Title	MOD/SVOD	R	N
Dubbed_Langauges	Movie	MOD/SVOD	O	N
Dubbed_Langauges	Preview	MOD/SVOD	O	N
Episode_ID	Title	MOD/SVOD	O	Y
Episode_Name	Title	MOD/SVOD	O	Y
Genre	Title	MOD/SVOD	O	Y
HDContent	Movie	MOD/SVOD	R	Y
HDContent	Preview	MOD/SVOD	O	N
ISAN	Title	MOD/SVOD	O	N
Languages	Movie	MOD/SVOD	O	N
Languages	Preview	MOD/SVOD	O	N
MSORating	Title	MOD/SVOD	O	N
Producers	Title	MOD/SVOD	O	Y

Table 4-2 Required ADI Metadata (continued)

Name	Type	Specification	Required by ADI	Required by VN
Provider	Movie	AMS	R	Y
Rating	Title	MOD/SVOD	R	Y
Rating	Preview	MOD/SVOD	R	N
Run_Time	Title	MOD/SVOD	R	Y
Run_Time	Preview	MOD/SVOD	R	Y
Screen_Format	Movie	MOD/SVOD	O	Y
Screen_Format	Preview	MOD/SVOD	O	N
Season_Finale	Title	MOD/SVOD	O	Y
Season_Premiere	Title	MOD/SVOD	O	N
Studio	Title	MOD/SVOD	O	Y
Subtitle_Languages	Movie	MOD/SVOD	O	N
Subtitle_Languages	Preview	MOD/SVOD	O	N
Summary_Long	Title	MOD/SVOD	O	N
Summary_Medium	Title	MOD/SVOD	O	N
Summary_Short	Title	MOD/SVOD	R	Y
Title	Title	MOD/SVOD	R	Y
Title_Brief	Title	MOD/SVOD	R	Y
Title_Sort_Name	Title	MOD/SVOD	O	N
Writer_Display	Title	MOD/SVOD	O	Y
Year	Title	MOD/SVOD	O	Y
Year	Preview	MOD/SVOD	O	Y

Table 4-3 Abbreviations and Definitions for ADI Metadata

Abbreviation	Definition
MOD	Movie on Demand
SVOD	Subscriber Video on Demand
AMS	Asset Management System
ADI	Asset Distribution Interface
VN	Cisco CDS Video Navigator
O	Optional
R	Required
Y	Yes
N	No

Cisco CSI XSD

This section provides the complete Cisco CSI schema. XML is used for data exchange, and object representation is documented in “XML Schema Definition (XSD).”

Schema

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:tns="com.cisco.csi" elementFormDefault="qualified"
targetNamespace="com.cisco.csi" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="AssetType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="catalog" />
      <xs:enumeration value="poster" />
      <xs:enumeration value="content" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="AssetId">
    <xs:restriction base="xs:string">
      <xs:maxLength value="128" />
      <xs:minLength value="1" />
      <xs:pattern value="[a-zA-Z0-9-_]+" />
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="AssetInfoType">
    <xs:attribute name="id" type="tns:AssetId" use="required" />
    <xs:attribute name="state" type="xs:int" use="required" />
    <xs:attribute name="reason" use="optional">
       <xs:simpleType>
          <xs:restriction base="xs:string">
             <xs:minLength value="1" />
             <xs:maxLength value="64" />
          </xs:restriction>
       </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="version" type="xs:string" use="optional" />
    <xs:attribute name="size" type="xs:long" use="optional" />
  </xs:complexType>
  <xs:complexType name="AssetStatusType">
    <xs:attribute name="id" type="tns:AssetId" use="required" />
    <xs:attribute name="state" type="xs:int" use="required" />
    <xs:attribute name="reason" use="optional">
       <xs:simpleType>
          <xs:restriction base="xs:string">
             <xs:minLength value="1" />
             <xs:maxLength value="64" />
          </xs:restriction>
       </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="TransferProtocolType">
    <xs:attribute name="protocol" type="xs:string" use="required" />
    <xs:attribute name="ipAddress" type="xs:string" use="required" />
    <xs:attribute name="port" type="xs:unsignedShort" use="required" />
    <xs:attribute name="userName" type="xs:string" use="optional" />
    <xs:attribute name="password" type="xs:string" use="optional" />
    <xs:attribute name="path" type="xs:string" use="required" />
  </xs:complexType>
  <xs:complexType name="CallbackAddressType">
    <xs:attribute name="ipAddress" type="xs:string" use="required" />
    <xs:attribute name="port" type="xs:unsignedShort" use="required" />
    <xs:attribute name="path" type="xs:string" use="required" />
    </xs:complexType>
    <xs:complexType name="CallbackMessageType">
      <xs:attribute name="type" type="tns:AssetType" use="required" />
      <xs:attribute name="id" type="tns:AssetId" use="required" />
      <xs:attribute name="status" type="xs:string" use="required" />
      <xs:attribute name="reason" type="xs:string" use="required" />
    </xs:complexType>
    <xs:complexType name="ErrorResponseType">
      <xs:attribute name="code" type="xs:string" />
      <xs:attribute name="reason" type="xs:string" />
    </xs:complexType>
    <xs:element name="GetAssetCountRequest">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="type" type="tns:AssetType" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="GetAssetCountResponse">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="count" type="xs:int" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="GetAssetInventoryListRequest">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="type" type="tns:AssetType" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="GetAssetInventoryListResponse">
      <xs:complexType>
        <xs:sequence>
          <xs:element minOccurs="0" maxOccurs="unbounded" name="info"
  type="tns:AssetInfoType" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="GetAvailableSpaceRequest">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="type" type="tns:AssetType" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="GetAvailableSpaceResponse">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="totalSpace" type="xs:long" />
          <xs:element name="freeSpace" type="xs:long" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="SetAssetRequest">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="type" type="tns:AssetType" />
          <xs:element name="id" type="tns:AssetId" />
          <xs:element name="version" type="xs:string" />
          <xs:element name="protocol" type="tns:TransferProtocolType" />
          <xs:element minOccurs="0" name="callback" type="tns:CallbackAddressType" />
        </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="SetAssetResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="status" type="tns:AssetStatusType" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="DeleteAssetRequest">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="type" type="tns:AssetType" />
        <xs:element name="id" type="tns:AssetId" />
        <xs:element name="version" type="xs:string" />
        <xs:element minOccurs="0" name="callback" type="tns:CallbackAddressType" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="DeleteAssetResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="status" type="tns:AssetStatusType" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="GetAssetsStatusRequest">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="type" type="tns:AssetType" />
        <xs:element maxOccurs="unbounded" name="id" type="tns:AssetId" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="GetAssetsStatusResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="status" type="tns:AssetStatusType" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="GetAssetsInfoRequest">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="type" type="tns:AssetType" />
        <xs:element maxOccurs="unbounded" name="id" type="tns:AssetId" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="GetAssetsInfoResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="info" type="tns:AssetInfoType" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

CHAPTER 5

Troubleshooting CDS Video Navigator and Poster Art Server

This chapter contains information that is useful for identifying and remedying problems related to Cisco CDS Video Navigator and Poster Art Server. This chapter presents the following major topics:

- Troubleshooting Video Navigator, page 5-1
- Troubleshooting Poster Art Server, page 5-8

Troubleshooting Video Navigator

The following tasks are presented:

- Using Video Navigator Logging and Log Files
- Understanding the Video Navigator Directory Structure
- Using Video Navigator Scripts
- Troubleshooting Specific Problems in Video Navigator

Using Video Navigator Logging and Log Files

Video Navigator log file entries can provide useful information for troubleshooting. Video Navigator uses the Apache Tomcat application server, which uses the log4j logging system.

The Video Navigator log file is named midas.log and resides in the /home/isa/MIDAS/logs directory. The log file is automatically rotated when it reaches 20 MB. Up to ten archived log files are stored in the /home/isa/MIDAS/logs directory, as in the following example:

```
-rw-rw-r-- 1 isa isa 342473 Jan 23 20:40 midas.log
-rw-rw-r-- 1 isa isa 20965936 Jan 19 17:45 midas.log.1
-rw-rw-r-- 1 isa isa 20960394 Jan 18 14:00 midas.log.2
-rw-rw-r-- 1 isa isa 20966090 Jan 17 20:52 midas.log.3
-rw-rw-r-- 1 isa isa 20968401 Jan 16 20:52 midas.log.4
-rw-rw-r-- 1 isa isa 20969543 Jan 15 20:51 midas.log.5
-rw-rw-r-- 1 isa isa 20969613 Jan 14 20:51 midas.log.6
-rw-rw-r-- 1 isa isa 20968092 Jan 13 20:51 midas.log.7
-rw-rw-r-- 1 isa isa 20966422 Jan 12 20:51 midas.log.8
-rw-rw-r-- 1 isa isa 20969681 Jan 11 20:51 midas.log.9
```

Logging Levels

The Video Navigator logging levels are FATAL, ERROR, WARN, INFO, and DEBUG. Logging levels go from least verbose to most verbose. The FATAL level generates the smallest number of messages, and the DEBUG level generates the greatest number of messages. The default logging level is WARN.

In a lab environment, consider keeping the logging level at DEBUG. However, using the DEBUG level affects performance. For load testing or a production environment, the logging level should be set to INFO or, for better performance, to WARN.

The Video Navigator logging configuration file is named midasLog.properties and resides in the /home/isa/MIDAS/config directory.

Do the following to change the logging level:

Step 1 If needed, log in as root on the CDE110 that hosts Video Navigator.

Step 2 Change the working directory as follows:

```
[root@system]# cd /home/isa/MIDAS/config
```

Step 3 Use a text editor to modify the midasLog.properties file.

a. Change the logging level in the property log4j.logger.Midas Logger to the applicable value. Allowed values are FATAL, ERROR, WARN, INFO, and DEBUG. For example, the following line changes the log level to DEBUG:

```
log4j.logger.MidasLogger=DEBUG, MidasLogFile
```

b. Save and close the midasLog.properties file.

Note Changes to the midasLog.properties file do not take effect until Video Navigator is restarted in Step 4.

Step 4 To stop and restart Video Navigator, issue the following commands:

```
[root@system]# stop_midas

MIDAS Running ......shutting down
Using CATALINA_BASE: /usr/local/tomcat
Using CATALINA_HOME: /usr/local/tomcat
Using CATALINA_TMPDIR: /usr/local/tomcat/temp
Using JRE_HOME: /usr/local/java
Killing: 17895
MIDAS stopped

[root@system]# start_midas

MIDAS not running ....... starting MIDAS
```

Log Entry Format

Each Video Navigator log entry has the following format:

[*Date Time*] [*Logging level*] [*Logging class*] - [*Log message*]

Some sample log entries are as follows:

```
[2008-02-04 18:49:49,626][ERROR] [CatalogHttpServer] - Error fetching catalog from
Tandberg java.io.IOException: Method failed: HTTP/1.1 503 Service Temporarily Unavailable

[2008-02-04 18:45:29,322][WARN] [BrowseCatalogAction] - device: 132 - Invalid page size -1
[2008-02-04 18:45:58,290][WARN] [BrowseCatalogAction] - device: 132 - Invalid page size -1
[2008-02-04 18:46:04,971][WARN] [BrowseCatalogAction] - device: 132 - Catalog 100 doesn't
exist

[2008-02-04 18:49:49,627][INFO] [CatalogHttpServer] - Fetch local catalog
[2008-02-04 18:49:49,825][INFO] [CatalogHttpServer] - Last local catalog
[2008-02-04 18:49:49,826][INFO] [CatalogHttpServer] - CR catalogId = 1
[2008-02-04 18:49:49,827][INFO] [CatalogHttpServer] - CR downloadTime = 1201219710722
[2008-02-04 18:49:49,828][INFO] [CatalogHttpServer] - CR fileName = catalog.113.xml
[2008-02-04 18:49:49,828][INFO] [CatalogHttpServer] - Going to load local catalog file
/home/isa/MIDAS/catalog/catalog.113.xml

[2008-02-04 18:49:50,133][DEBUG] [AssetIndexer] - Built index for asset 327
[2008-02-04 18:49:50,133][DEBUG] [AssetIndexer] - Add title = Wedding Crashers
[2008-02-04 18:49:50,134][DEBUG] [AssetIndexer] - Add summary = Two divorce mediators who
crash weddings to pick up women engage in comical misadventures at the nuptials of a
politician's daughter. Ends 09/02
[2008-02-04 18:49:50,134][DEBUG] [AssetIndexer] - Add content = Wedding Crashers Two
divorce mediators who crash weddings to pick up women engage in comical misadventures at
the nuptials of a politician's daughter. Ends 09/02 Owen Wilson Vince Vaughn Christopher
Walken Rachel McAdams
[2008-02-04 18:49:50,142][DEBUG] [AssetIndexer] - Built index for asset 328
[2008-02-04 18:49:50,142][DEBUG] [AssetIndexer] - Add title = The Rock
[2008-02-04 18:49:50,143][DEBUG] [AssetIndexer] - Add summary = Sean Connery and Nicolas
Cage team up to stop a devious U.S. general's threat to chemically attack San Francisco
from Alcatraz Island. Ends 09/09
[2008-02-04 18:49:50,160][DEBUG] [AssetIndexer] - Add content = The Rock Sean Connery and
Nicolas Cage team up to stop a devious U.S. general's threat to chemically attack San
Francisco from Alcatraz Island. Ends 09/09 Sean Connery Nicolas Cage Ed Harris Michael
Biehn William Forsythe
```

Understanding the Video Navigator Directory Structure

Video Navigator uses the directory structure shown in Table 5-1.

Table 5-1 Video Navigator Directory Structure

Directory	Contents
/home/isa/MIDAS/catalog	Latest catalog files fetched from the VBO. A maximum of five latest catalog files are kept in this directory. **Caution** *Do not modify the contents of this directory.*
/home/isa/MIDAS/config	Files for configuring Video Navigator
/home/isa/MIDAS/docs	Release notes or some user help files
/home/isa/MIDAS/epg/data	EPG data and a mapping file
/home/isa/MIDAS/epg/index	EPG search indexes
/home/isa/MIDAS/logs	Video Navigator log files. For more information, see the "Using Video Navigator Logging and Log Files" section on page 5-2.
/home/isa/MIDAS/scripts	Scripts for starting, stopping, and checking whether Video Navigator is running
/home/isa/MIDAS/test	Sample test files
/home/isa/MIDAS/webapps	Video Navigator web applications.
/usr/local/apache2	Apache web server binary executable
/usr/local/tomcat	Files related to the Tomcat application server

Using Video Navigator Scripts

The Video Navigator software includes the scripts shown in Table 5-2. These scripts may be needed to troubleshoot and resolve some Video Navigator problems.

Table 5-2 Video Navigator Scripts

Directory and Script	Contents
/home/isa/MIDAS/scripts/start_midas	Starts Video Navigator and the Apache Tomcat application server (tomcat5).
/home/isa/MIDAS/scripts/stop_midas	Stops Video Navigator.
/home/isa/MIDAS/scripts/check_midas	Checks whether Video Navigator is running and displays the release number of the installed Video Navigator software.
/home/isa/MIDAS_IntegrationTest/clientinterfacetest	Checks whether the client-facing web services interface is working correctly. The **clientinterfacetest** command does not verify the connection from Video Navigator to the STB.
/home/isa/MIDAS_IntegrationTest/searchTest	Checks whether EPG and VoD search are working.

Note The Video Navigator software installation script adds /home/isa/MIDAS/scripts to the PATH variable but does not add /home/isa/MIDAS_IntegrationTest to the PATH. Therefore, your current working directory must be /home/isa/MIDAS_IntegrationTest when the clientinterfacetest script is executed.

Troubleshooting Specific Problems in Video Navigator

This section provides information on troubleshooting.

Problems with VOD Catalog Not Displaying on the STB

If the VOD catalog is not available on an STB, set the Video Navigator logging level to DEBUG. For information on how to set the logging level, see the "Logging Levels" section on page 5-2.

Observe the entries in the log file for SignOn and BrowseCatalog requests. When the STB user selects the VOD service, the Video Navigator client on the STB makes a SignOn request to Video Navigator. The Video Navigator client queries Video Navigator by device ID (MAC address of the STB) for the list of VoD services to which it is entitled. With the logging level set to DEBUG, you should be able to see the SignOn request in the Video Navigator log file, as in the following example:

```
[2008-01-31 11:10:38,125][DEBUG] [SignOnAction] - SignOn request
[2008-01-31 11:10:38,125][DEBUG] [SignOnAction] - deviceId = null
[2008-01-31 11:10:38,128][DEBUG] [SignOnAction] - deviceId 00:14:F8:E3:0A:DF- SignOn:
processServices
[2008-01-31 11:10:38,152][DEBUG] [SignOnAction] - Process service done
[2008-01-31 11:10:38,152][DEBUG] [SignOnAction] - deviceId 00:14:F8:E3:0A:DF- SignOn:
processRentals
[2008-01-31 11:10:38,221][DEBUG] [SignOnAction] - deviceId 00:14:F8:E3:0A:DF has 0 rental
records on Tandberg system
[2008-01-31 11:10:38,221][DEBUG] [SignOnAction] - No of rental records in MIDAS database =
0
[2008-01-31 11:10:38,222][DEBUG] [SignOnAction] - Process rental done
[2008-01-31 11:10:38,222][DEBUG] [ActionSkeleton] - HTTP Response
[2008-01-31 11:10:38,222][DEBUG] [ActionSkeleton] - <?xml version="1.0" encoding="UTF-8"?>
```

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"><env:Body
xmlns="com.cisco.midas.client"><SignOnResponse /></env:Body></env:Envelope>^
```

When the STB user starts browsing the VOD catalog, the Video Navigator client on the STB makes a BrowseCatalog request to Video Navigator. The following log entries show a BrowseCatalog request-and-response example:

```
[2008-01-31 11:10:38,334][DEBUG] [BrowseCatalogAction] - BrowseCatalog request
[2008-01-31 11:10:38,335][DEBUG] [BrowseCatalogAction] - deviceId = 00:14:F8:E3:0A:DF
[2008-01-31 11:10:38,335][DEBUG] [BrowseCatalogAction] - catalogId = 1
[2008-01-31 11:10:38,335][DEBUG] [BrowseCatalogAction] - categoryId = 0
[2008-01-31 11:10:38,338][DEBUG] [BrowseCatalogAction] - pageNo = 1
[2008-01-31 11:10:38,338][DEBUG] [BrowseCatalogAction] - pageSize = 7
[2008-01-31 11:10:38,338][DEBUG] [BrowseCatalogAction] - Construct JDOM response
[2008-01-31 11:10:38,339][DEBUG] [BrowseCatalogAction] - deviceId 00:14:F8:E3:0A:DF - From
= 0 to = 5
[2008-01-31 11:10:38,339][DEBUG] [BrowseCatalogAction] - deviceId 00:14:F8:E3:0A:DF -
Output list size = 5
[2008-01-31 11:10:38,339][DEBUG] [BrowseCatalogAction] - hasCategory
[2008-01-31 11:10:38,339][DEBUG] [BrowseCatalogAction] - hasServiceCategory
[2008-01-31 11:10:38,339][DEBUG] [ActionSkeleton] - HTTP Response
[2008-01-31 11:10:38,340][DEBUG] [ActionSkeleton] - <?xml version="1.0"
encoding="UTF-8"?>^M
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"><env:Body
xmlns="com.cisco.midas.client"><BrowseCatalogResponse><category id="2" name="All Movies"
parentId="0" desc="" /><category id="4" name="Cat1" parentId="0" desc="" /><category
id="12" name="HBO" parentId="0" desc="" /><serviceCategory id="15" name="MAX" parentId="0"
desc=""><service id="203" name="MAX" desc="Cinemax On Demand" price="$2.99"
purchased="false"><backoffice name="vendorName" value="Tandberg" /><backoffice
name="vendorVersion" value="3.3" /><backoffice name="serviceOfferingId" value="203"
/><backoffice name="serviceName" value="MAX"
/></service></serviceCategory><serviceCategory id="20" name="Showtime" parentId="0"
desc=""><service id="255" name="SOD" desc="Showtime On Demand" price="$5.99"
purchased="false"><backoffice name="vendorName" value="Tandberg" /><backoffice
name="vendorVersion" value="3.3" /><backoffice name="serviceOfferingId" value="255"
/><backoffice name="serviceName" value="SOD"
/></service></serviceCategory><pageNo>1</pageNo><pageSize>7</pageSize><totalItems>5</total
Items><hasMore>false</hasMore><catalogVersion>1</catalogVersion></BrowseCatalogResponse></
env:Body></env:Envelope>
```

It is possible that the SignOn and BrowseCatalog requests and responses appear in the Video Navigator log entries, but no VoD catalog is displayed on the STB. This set of symptoms may indicate that there is a network problem or (less likely) that the STB cannot process the returned data.

If there is no log-file entry for the SignOn request or other activity in the Video Navigator log file, follow these steps:

Step 1 Log in as root.

Step 2 If you know the IP address of the STB, use the **ping** command and the IP address to verify that the Video Navigator server (CDE110) can reach the STB.

Step 3 To check that Video Navigator is running, issue the following command:

```
[root@system]# check_midas
```

If Video Navigator is running, "MIDAS is running" is returned. If Video Navigator is not running, "MIDAS is not running" is returned.

Do one of the following:

- If Video Navigator is not running, go to Step 5.

- If Video Navigator is running, go to Step 4.

Step 4 If Video Navigatorr is running, verify that the Apache HTTP server service (httpd) is running by issuing the following command:

```
[root@system]# ps -ef | grep httpd
```

Do one of the following depending on whether httpd is running:

- If httpd is running, the output is similar to the following. Go to Step 6.

```
root      2880     1  0 Jul18 ?        00:00:00 /usr/sbin/httpd
apache    4881  2880  0 04:03 ?        00:00:00 /usr/sbin/httpd
apache    4882  2880  0 04:03 ?        00:00:00 /usr/sbin/httpd
apache    4883  2880  0 04:03 ?        00:00:00 /usr/sbin/httpd
apache    4884  2880  0 04:03 ?        00:00:00 /usr/sbin/httpd
apache    4885  2880  0 04:03 ?        00:00:00 /usr/sbin/httpd
apache    4886  2880  0 04:03 ?        00:00:00 /usr/sbin/httpd
apache    4887  2880  0 04:03 ?        00:00:00 /usr/sbin/httpd
apache    4888  2880  0 04:03 ?        00:00:00 /usr/sbin/httpd
```

- If httpd is not running, the preceding output will not be present. Restart the httpd daemon by issuing the following command:

```
[root@system]# service httpd restart
```

  Go to Step 6.

Step 5 If Video Navigator is not running, restart it and verify that it is running by issuing the following commands:

```
[root@system]# start_midas

MIDAS not running ....... starting MIDAS

[root@system]# check_midas
```

Note Because CD-VN is a web application, there is no Video Navigator process that you can check to see if it is running. Instead, use the **check_midas** command.

```
MIDAS (10.1.X.X) is running
```

Step 6 On the STB, select or have the user select the VoD service.

Step 7 Check the log entries to verify that a SignOn request from the STB is sent to Video Navigator.

Troubleshooting Poster Art Server

This section covers issues that are specific to Poster Art Server.

The following tasks are presented:

- Using Poster Art Server Logging and Log Files
- Understanding the Poster Art Server Directory Structure
- Troubleshooting Poster Art Server Sequence Flows

Using Poster Art Server Logging and Log Files

After Poster Art Server is installed, it uses the default Apache log4j configuration file, log4j.properties:

```
<webapps_path>/PAServer/WEB-INF/classes/log4j.properties
```

Table 5-3 lists the components of the log4j.properties file.

Table 5-3 Components of log4j.properties

Component	Description/Example
log4j.appender.PasLogFile	org.apache.log4j.RollingFileAppender
log4j.appender.PasLogFile.File	/home/isa/MIDAS/logs/PosterServer.lo g
log4j.appender.PasLogFile.MaxFileSize	40 MB
log4j.appender.PasLogFile.MaxBackupIndex	5
log4j.appender.PasLogFile.layout	org.apache.log4j.PatternLayout
log4j.appender.PasLogFile.layout.ConversionPattern	[%d][%p] [%C{1}] - %m%n
log4j.logger.com.cisco	debug, PasLogFile

Each log message is in the following format:

<local time>: <severity>: <operation>: <error code>: <error message string>

The following are example log messages:

```
[2010-02-25 23:50:32,110][DEBUG] [PAServerInit] - Complete to init DBConnection
[2010-02-25 23:50:32,110][DEBUG] [PAServerInit] - Init task manager
[2010-02-25 23:50:32,118][DEBUG] [PAServerInit] - Init image store
[2010-02-25 23:50:32,123][DEBUG] [PAServerInit] - Init CSI interface
```

Understanding the Poster Art Server Directory Structure

Poster Art Server uses the directory structure shown in Table 5-4.

Table 5-4 Poster Art Server Directory Structure

Directory	Description/Contents
/home/isa/MIDAS/webapps/PAServer	Root of Poster Art Server application
/home/isa/MIDAS/webapps/PAServer/WEB-INF/web.xml	Poster Art Server configuration files
/home/isa/MIDAS/config/paslog.properties	Poster Art Server log configuration file. If this file does not exist, the log file is /home/isa/MIDAS/webapps/PAServer/WEB-INF/classes/log4j.properties

Troubleshooting Poster Art Server Sequence Flows

This section presents some detailed sequence flows and explains how to troubleshoot them.The sequence flow for each use case lists a set of error conditions that are logged to the application log file, pas.log. (See Using Poster Art Server Logging and Log Files, page 5-8).

Note Syslog logging is not supported in this release of Poster Art Server.

Table 5-5 describes the Poster Art Server log message formats and provides examples.

Table 5-5 Poster Art Server Log Message Formats

Element	Description	Value	Example
local time	Local time code	Any ASCII string	Tue Dec 23 02:30:06 2008
Error code	Unique code for each error See Poster Art Server Error Messages, page 5-17.	1xx: Internal System Error	100
		2xx: Network Communication Error	200
		3xx: Communication Message Error	300
		Fatal	Warn

Table 5-5 Poster Art Server Log Message Formats (continued)

Element	Description	Value	Example
Severity	Degree of impact to the system	Fatal	Warn
		Error	
		Warn	
		Info	
		Debug	
		Trace	
Operation	Brief description of the operation being attempted when the error occurred	SystemStartup	
		DBconnect	DBconnect
		GetAvailableSpace	
		GetAssetCount	
		GetAssetsInfo	
		GetAssetsStatus	
		GetAssetInventoryList	
		AddAsset	
		UpdateAsset	
		DeleteAsset	
		GetPoster	

The following is an example log message:

```
Tue Dec 23 02:30:06 2008 : Critical : 300 : SystemStartup : Could not find configuration
file config.xml
```

Note The format of the above message illustrates the default. The operator can customize this format by modifying pasLog.properties.

For the list of all error codes and error strings, see the "Poster Art Server Error Messages" section on page 5-17.

Sequence Flows

This section presents a variety of practical sequence flows that can be used in troubleshooting:

- SystemStartup
- GetAvailableSpace
- GetAssetInventoryList
- AddAsset
- UpdateAsset
- DeleteAsset
- GetPoster
- SystemStop

SystemStartup

The following components are involved in this sequence:

VBO <> CSIService <> ImageStore <> Database

1. ImageStore sets up a persistent connection with the database.
2. CSIService sends a SignOn request to the VBO.
3. The VBO responds to the message. If the response message reports an error, CSIService keeps retrying until it gets a successful response.
4. CSIService starts listening at the configured port.

Note To synchronize with the poster data, and ensure system resilience, the VBO must execute a sign-on. For more information, see the "SignOn" section on page 4-25.

The system does not load poster information from the database at start up, but rather waits until it receives a sign-on request.

Table 5-6 lists and describes the error codes for SystemStartup.

Table 5-6 Error Codes and Descriptions for SystemStartup

Error code	Description
100	Load configuration file error. Poster Art Server uses the default configuration, and logs a warning message in the log file.
101	Read configuration parameter error. Poster Art Server uses the default parameter, and logs a warning message in the log file.
102	ImageStore connection to database error. Poster Art Server keeps retrying, and logs a fatal error message in log file.
103	CSIService fails to listen at the configured port. Poster Art Server quits, and logs a fatal error message in the log file. Troubleshoot to resolve the problem.
201	VBO server is down, and sign on fails. Poster Art Server keeps retrying, and logs a fatal error message in the log file.
202	VBO server responds that sign-on has failed. Poster Art Server keeps retrying, and logs a fatal error message in the log file.

GetAvailableSpace

The following components are involved in this sequence:

VBO <> CSIService <> ImageStore <> FileSystem

1. The VBO sends a GetAvailableSpace request to the listening address. CSIService gets the request and parses it.
2. CSIService calls the ImageStore by using the GetAvailableSpace method to get the size of the available space.
3. ImageStore makes a request to the native file system to get the size of the available space.
4. CSIService constructs an HTTP response message.
5. CSIService sends the response message back to the VBO.

Table 5-7 lists and describes the error codes for GetAvailableSpace.

Table 5-7 Error Codes and Descriptios for GetAvailableSpace

Error code	Description
301	Invalid asset type in request. Poster Art Server responds with an InvalidAssetException to the VBO and logs the error.
104	Native file system returns an error message when ImageStore issues a GetAvailableSpace request. Poster Art Server responds to the VBO with an InternalErrorException and logs the error.

GetAssetInventoryList

The following components are involved in this sequence:

VBO <> CSIService <> ImageStore <> Database

1. The VBO sends a GetAssetInventoryList request to the listening address. CSIService gets the request and parses it.
2. CSIService calls the ImageStore by using the GetAssetList method to get all asset information.
3. ImageStore sends a GetAll request to the database.
4. The database returns the asset information list.
5. ImageStore passes the asset information list to CSIService.
6. CSIService constructs an HTTP response message.
7. CSIService sends the response message back to the VBO.

Table 5-8 lists and describes the error codes for GetAssetInventoryList.

Table 5-8 Error Codes and Descriptions for GetAssetInventoryList

Error code	Description
301	Invalid asset type in request. Poster Art Server responds to the VBO with an InvalidAssetException and logs the error.
105	Database transaction error. Poster Art Server responds to the VBO with an InternalErrorException and logs the error.

AddAsset

The following components are involved in this sequence:

VBO <> CSIService <> TaskManager <> Task <> ImageStore <> Connection <> ImageProcessor <> Database <> VBO Storage

1. The VBO sends an AddAsset request to the listening address. CSIService gets the request and parses it.
2. CSIService adds the request to the TaskManager asynchronous task queue, and a check is made to see if the asset is already in the system.
3. TaskManager creates the poster information entry in the database, and sets the state to Pending.
4. CSIService constructs an HTTP response message and sends it to the VBO.
5. When it is time for TaskManager to process a new request, it checks out a new task from the Task Queue.
6. If TaskManager needs a task, it creates one. For the algorithm used, see the "GetAssetInventoryList" section on page 5-13.
7. Task creates a Connection object.
8. TaskManager starts the task to download the poster.
9. Task sets the state of the poster to Transferring.
10. Task asks the Connection object to connect to the target server.
11. Connection makes a connection to the target server.

12. Task calls the AddImage method of ImageStore to add the poster file.
13. ImageStore calls the Connection object to start the download.
14. The Connection object communicates with the target server to download the data.
15. After the download is complete, ImageStore processes the image.
16. ImageStore works with ImageProcessor to implement some image processing features, such as resizing.
17. ImageStore stores the poster file in the corresponding local directory.
18. Task asks Connection to disconnect from the target server.
19. Connection disconnects from the target server.
20. Task sets the state of the poster to Completed if the operation is successful, or to Failed if a failure occurs in any of the preceding steps.
21. Task constructs a callback message and sends it to the VBO.
22. Task communicates with TaskManager to check in and terminate itself.

Table 5-9 lists error codes and descriptions for AddAsset.

Table 5-9 Error Codes and Descriptions for AddAsset

Error code	Description
301	Invalid asset type in the request. Poster Art Server responds to the VBO with InvalidAssetType and logs the error.
302	Asset exists in the system with Completed status and the same version that the VBO has for it. Poster Art Server responds to the VBO with VersionAlreadyExists and logs the error.
303	Transport protocol in the request message has the wrong format. Poster Art Server responds to the VBO with InvalidTransferProtocol, and logs the error.
304	Callback address in the request message is in the wrong format. Poster Art Server responds to the VBO with InvalidCallbackAddress, and logs the error.
105	Database transaction error. Poster Art Server responds to the VBO with InternalErrorException and logs the error. The state of the poster is set to Failed and the VBO must ingest the poster again.
203	Problem with VBO repository server. Poster Art Server sends VBORepositoryErrorException to the VBO in the callback message and logs the error. The state of the poster is set to Failed and the VBO must ingest the poster again.
106	Problem with resizing. Poster Art Server responds to the VBO with InternalErrorException and logs the error. The state of the poster is set to Failed and the VBO must ingest the poster again.
107	File system error. Poster Art Server responds to the VBO with InternalErrorException, and logs the error. The state of the poster is set to Failed and the VBO must ingest the poster again.

Table 5-9 Error Codes and Descriptions for AddAsset (continued)

Error code	Description
305	Asset in the AddAsset request exists and is in Transferring state. Poster Art Server responds to the VBO with InvalidId exception.
204	Callback server is unreachable after maximum number of retries. Poster Art Server only logs the error, because there is no way to contact the VBO.
109	Available space is used up. Poster Art Server responds to the VBO with InternalErrorException, and logs the error. The state of the poster is Pending and the VBO requires troubleshooting.

UpdateAsset

The following components are involved in this sequence:

VBO <> CSIService <> TaskManager <> ImageStore <> Connection <> ImageProcessor <> Database
<> VBO Storage

This is similar to the "AddAsset" section on page 5-13, except that the following steps are inserted between Step 12 and Step 13:

1. Call the UpdateImage method of an ImageStore instance, instead of calling AddImage.
2. ImageStore gets the poster information from the database.
    a. If the poster version is different from the existing version, Poster Art Server deletes the old one, including resized poster files, from ImageStore.
    b. If the poster version is the same as the requested one, Poster Art Server responds with a VersionAlreadyExists exception.

Error codes are identical to those in Table 5-9 on page 5-14, except that AddAsset is replaced with UpdateAsset.

DeleteAsset

The following components are involved in this sequence:

VBO <> CSIService <> TaskManager <> Task <> ImageStore <> Database

1. The VBO sends a DeleteAsset request to the listening address. CSIService gets the request and parses it.
2. CSIService adds the request into the TaskManager asynchronous task queue.
3. CSIService constructs an HTTP response message and sends it to the VBO.
4. When it is time for TaskManager to process a new request, it checks out a new task from the task queue. Any available task can process this job.
5. TaskManager places the callback information into the Task object.
6. If the state of the asset is not Transferring, TaskManager starts the task to delete the poster.
7. Task calls the DeleteImage method of ImageStore to delete the poster file.
8. ImageStore deletes the poster file, including resized ones, from the corresponding local directory.
9. ImageStore deletes the poster information from the database.

10. Task constructs a callback message and sends it back to the VBO.
11. Task checks in itself into TaskManager and terminates.

Table 5-10 lists error codes and descriptions for DeleteAsset.

Table 5-10 Error Codes and Descriptions for DeleteAsset

Error code	Description
301	Invalided asset type in request. Poster Art Server responds to the VBO with InvalidAssetException, and logs the error.
306	Version requested is different from the one in the system. Poster Art Server responds to the VBO with VersionDoesNotExist, and logs the error.
105	Database transaction error. Poster Art Server responds to the VBO with InternalErrorException, and logs the error. The VBO must delete the poster again.
107	File system error. Poster Art Server responds to the VBO with InternalErrorException, and logs the error. The VBO must delete the poster again.

GetPoster

The following components are involved in this sequence:

STB <> PosterHttpServer <> ClientService <> ImageStore <> Image

1. The STB sends a GetPoster request to the Poster Art Server.
2. PosterHTTPServer gets and parses the request, then calls the ClientService GetPoster method.
3. ClientService calls the GetImage method of ImageStore, with the input arguments of xdim and ydim. ImageStore creates an Image object, assigns the correct local path to it, and returns the object to the STB.
4. ClientService calls the GetImageData method of the Image object to get binary data.
5. CSIService constructs a response message and returns it to PostHTTPServer.
6. PostHTTPServer constructs an HTTP response message and sends it back to the STB.

Table 5-11 lists error codes and descriptions for GetPoster.

Table 5-11 Error Codes and Descriptions for GetPoster

Error code	Description
105	Database transaction error. Poster Art Server responds to the the STB with a MIDASInternalSystemError exception and logs the error. An exception is returned either as a SOAP message (as a soapenv:Fault element within the SOAP envelope), or as just a plain XML message.

Table 5-11 Error Codes and Descriptions for GetPoster (continued)

Error code	Description
107	File system error. Poster Art Server responds to the the STB with a MIDASInternalSystemError exception and logs the error.
108	Poster does not exist. Poster Art Server responds to the the STB with a MIDASInvalidPosterId exception and logs the error.

SystemStop

When Poster Art Server service is stopped, CSIService sends a SignOff request to the VBO. However, if any error occurs it does not retry, and a log is written in the log file.

Table 5-12 lists error codes and descriptions for SystemStop.

Table 5-12 Error Codes and Descriptions for SystemStop

Error code	Description
201	VBO server is down, sign off failed. Poster Art Server logs a fatal error message in the log file.
202	VBO server response sign on failed. Poster Art Server logs a fatal error message in the log file.

Poster Art Server Error Messages

Table 5-13 lists all of the error codes and strings that are used for Poster Art Server.

Table 5-13 Poster Art Server Error Codes and Strings

Error code	Error string
100	The system was unable to load the configuration file (By default, the configuration file name is web.xml. The log property file name is pasLog.properties.)
101	Failed to read configuration parameter
102	Connections database error
103	Failed to listen at configured port
104	Get disk info error
105	Database transaction error
106	Failed to resize
107	IO error
108	Poster does not exist
109	Available space is used up
201	Failed to connect remote server
202	Remote server sign on failure
203	VBO repository server has problem

Table 5-13 Poster Art Server Error Codes and Strings (continued)

Error code	Error string
204	Call back server has problem
301	Invalid asset type
302	Asset version exists in system
303	Invalid transport protocol
304	Invalid callback address
305	Invalid asset ID
306	The asset version in request is different from the one in system
307	Wrong format or version
308	Asset is in processing progress

web.xml

The following is an example of the configuration file web.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <display-name>PosterArtServer</display-name>

    <servlet>
        <servlet-name>PAServerInit</servlet-name>
        <servlet-class>com.cisco.paserver.PAServerInit</servlet-class>

        <init-param>
           <param-name>log4j-init-file</param-name>
           <param-value>/home/isa/MIDAS/config/paslog.properties</param-value>
        </init-param>

        <!-- If set SignOnOffMode to be 1, poster art server will sign on vbo -->
        <init-param>
          <param-name>SignOnOffMode</param-name>
          <param-value>0</param-value>
        </init-param>

    <!-- PAServerIp, PAServerPort and PAServerName are used to construct vbo sign-on
request-->
        <init-param>
           <param-name>PAServerIp</param-name>
           <param-value>127.0.0.1</param-value>
        </init-param>

        <init-param>
           <param-name>PAServerPort</param-name>
           <param-value>8088</param-value>
        </init-param>

        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet>
    <description>
    </description>
    <display-name> CSIHttpServer</display-name>
    <servlet-name>CSIHttpServer</servlet-name>
    <servlet-class> com.cisco.paserver.csi.CSIHttpServer</servlet-class>
</servlet>
<servlet>
    <description>MIDAS Poster Server</description>
    <display-name>PosterHttpServer</display-name>
    <servlet-name>PosterHttpServer</servlet-name>
    <servlet-class>com.cisco.midas.client.PosterServer</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>CSIHttpServer</servlet-name>
    <url-pattern>/CSI/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>PosterHttpServer</servlet-name>
    <url-pattern>/PosterHttpServer/*</url-pattern>
</servlet-mapping>

<welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
</welcome-file-list>

<context-param>
    <param-name>MaxRetry</param-name>
    <param-value>3</param-value>
</context-param>

<context-param>
    <param-name>RetryInterval</param-name>
    <param-value>3</param-value>
</context-param>

<context-param>
    <param-name>TimeOut</param-name>
    <param-value>3</param-value>
</context-param>

<context-param>
    <param-name>ResizeOption</param-name>
    <param-value>352x288:480x480:600x600</param-value>
</context-param>

<context-param>
    <param-name>DBHost</param-name>
    <param-value>127.0.0.1</param-value>
</context-param>

<context-param>
    <param-name>DBPort</param-name>
    <param-value>9999</param-value>
</context-param>

<context-param>
    <param-name>MinDBConnection</param-name>
    <param-value>3</param-value>
</context-param>
```

```
    <context-param>
        <param-name>MaxDBConnection</param-name>
        <param-value>8</param-value>
    </context-param>

    <context-param>
        <param-name>ImageStoreRootPath</param-name>
        <param-value>/tmp/pasroot</param-value>
    </context-param>

    <context-param>
        <param-name>VboUrl</param-name>
        <param-value>http://127.0.0.1:8000/sign</param-value>
    </context-param>

</web-app>
```