Introduction to Cisco VDS-TV Software APIs

Cisco TV Videoscape Distribution Suite (VDS) software provides three sets of application program interfaces (APIs):

- Monitoring
- Maintenance
- Real Time Streaming Protocol (RTSP) Stream Diagnostics
- D5 Interface
- Package Details
- Replication Group
- PVR Scheduler Interface
- Video Backoffice (VBO)
- TV Playout
- Recorder Configuration

The Monitoring APIs use Hypertext Transfer Protocol (HTTP) GET message format for sending control messages between any HTTP client and the Cisco VDS.

The Maintenance APIs use Hypertext Transfer Protocol (HTTP) GET message format for restarting services such as AIM.

The RTSP Stream Diagnostics APIs use eXtensible Markup Language (XML) over HTTP for sending control messages between any HTTP client and the Cisco VDS.

The D5 Interface APIs use HTTP GET/POST message format for exchanging server configuration and server statistics between On Demand Resource Manager (ODRM) and Streaming Server component of Next Generation On Demand (NGOD) system.

The Package APIs use HTTP GET/POST message format to provide the package status details, VOD Market details, Content Distribution Rules details and upload VOD/Content Distribution Rules configuration to the Cisco VDS.

The Replication Group APIs use HTTP GET/POST message format to add/delete replication groups, assign servers to a replication group, un-assign servers from a replication group and retrieve the replication groups configured in the system.

The PVR Scheduler Interface APIs uses HTTP POST message format to interact with the PVR Server.

The TV Playout APIs use the **curl** utility to retrieve content from the Cisco VDS using HTTP.

Recorder Configuration APIs use a REST client to upload and download nDVR Recorder configuration settings.

The APIs can be used to monitor the states of specified VDS functions, report on VDS configurations, or create VDS entities. Table 1-1 describes these APIs.

Table 1-1 Cisco TV VDS APIs

API	Description
Monitoring	Returns information on content states, stream history, trick-mode history, as well as other information about the devices.
Maintenance	Provides a way to restart services such as AIM.
RTSP stream diagnostics	Returns information about streams and trick modes for a specified session ID in an RTSP environment, as well as stream information about a specified smart card. A smart card is a unique ID that represents a set-top box (STB).
D5 Interface	This interface is used by Streaming Server to notify the ODRM of streaming server configuration and streaming server statistics/current resource usage and is also used by ODRM to query the streaming server status and current resource usage.
Package Details	Returns information on package status, package history, package stats and content distribution rules. It also provides details on SOAP Exports and SOAP Imports functions. Provides a way to upload VOD Market configurations and Content distribution rules configuration.
Replication Group	Returns information on replication group configuration list, unassigned replication servers list. Provide a way to add a replication group, delete a replication group, assign serves to a replication group and un-assign servers from a replication group.
VBO APIs	Returns information on content list and session play history.
TV Playout	Returns information on TV Playout schedules, content and barker streams currently playing, configured TV Playout channels, content and barker streams that were playing during a specified time interval, and all barker streams. Provides a way to import TV Playout schedules, create barker streams and playlists, and start and stop barker streams.
Recorder Configuration	Provides a way to upload and download nDVR Recorder configuration settings.

This chapter contains the following sections:

Monitoring and RTSP Stream Diagnostic Interfaces

This section describes the Monitoring and the RTSP Stream Diagnostic APIs.

Connections

Any HTTP client that can send a request to the Content Delivery System Manager (CDSM) in the proper format can be used to send the API messages.

Connections can be semi-persistent or persistent; that is, the connection can be used for a single request-response pair or multiple request-response pairs.

A standard set of HTTP headers is used for all HTTP requests and responses. These headers include content type, content length, and a sequence number.

The HTTP request transmitter is considered the client and the HTTP request receiver is considered the server. The client always initiates the connection. The client must either receive a response or time out a request before sending another request on the same connection. The same connection can be used for multiple requests. However, when the client sends a new request, if there is already a connection established for a previous request for which the client is expecting a response from the server, then the client must open a new connection for the new request.

The API messages require a bidirectional socket connection for sending an HTTP request and receiving an HTTP response. The persistent protocol maintains a connection between requests until either the client or server indicates that the socket should close. This is typical of HTTP/1.0.

Persistent connections are handled using the traditional mechanisms specified in HTTP. In HTTP/1.0, connections are persistent by default. The HTTP response must include the header “Connection: close” in order to indicate that the connection will be closed at the end of transmission. During periods when no messages are being exchanged, the client or server may close the connection to conserve resources. The recommended connection approach is to use the default HTTP/1.0 behavior and to use the same connection for all requests.

HTTP Headers

This section covers the HTTP header formats for the request messages and the response messages.

Request Messages

There are two different request message formats:

- Monitoring and Trick-mode Event Reconciliation requests use an HTTP GET message format
- RTSP Stream Diagnostics requests use an HTTP POST message format

Monitoring Requests

The Monitoring APIs use an HTTP GET message format. All parameters are included in the HTTP GET message.

Table 1-2 provides an overview of the expected input parameters for each request message.

Table 1-2 Monitoring Request Parameters

Parameter	Required or Optional	Description
messageType	Required	Always required.
fromDate	Required for time-based messages	Required for StreamHistory, TrickModeHistory, PlayServerHistory, Errors, and CapacityPlanning. Length of time between fromDate and toDate must not exceed one hour.
toDate	Required for time-based messages	Required for StreamHistory, TrickModeHistory, PlayServerHistory, Errors, and CapacityPlanning.
maxRows	Optional	Specifies the maximum number of rows to return for this result set. Available for all messages apart from Trick-mode Event Reconciliation.
fromOffset	Optional	Specifies the row offset to start returning for this result set. A zero-based offset. Available for all messages apart from Trick-mode Event Reconciliation.
dateFormat	Optional	Specifies the formatting of the fromDate and toDate parameters to be seconds or milliseconds since epoch. The default is seconds. This parameter does not affect the output formatting of dates in the return message. The options are **sec** or **ms**.
Session	Optional	Specifies what type of error messages to retrieve from the VDS. The options are: • **1**—Retrieve only session-related error messages. • **2**—Retrieve only error messages not session-related (no session ID). • **3**—Retrieve all error messages. The default is 3.
StreamID	Mandatory for Trick-mode Event Reconciliation	ID of the stream for which a list of trick-modes events is being requested.
Action	Optional	Specifies the range of Streamers to include in the report. Available for the CapacityPlanning message. The options are: • OVERALL—Present capacity data for all Streamers and ISVs[1] (also known as SSVs). • SERVICEGROUP—Filter the capacity data by the specified Service Group. • SERVER—Filter the capacity data by the specified Streamer or ISV.
serviceGroup	Mandatory if action is set to SERVICEGROUP	Service Group identifier.
serverID	Mandatory if action is set to SERVER	Streamer or ISV identifier.

Table 1-2 Monitoring Request Parameters (continued)

timeType	Optional	Valid values for the timeType parameter are as follows: • HOUR—Provide peak bandwidth and stream count per hour for the specified date range. • DAY—Provide peak bandwidth and stream count per day for the specified date range. • WEEK—Provide peak bandwidth and stream count per week for the specified date range. Incomplete weeks are not returned. The start date determines the first day of the week. For example, if you specify Tuesday, the 2nd of November 2010 as the start date, the first week is calculated as spanning from Tuesday, the 2nd of November 2010 to Monday, the 8th of November 2010. • MONTH—Provide peak bandwidth and stream count per month. Incomplete months are not returned. The start date determines the first day of the month. For example, if you specify Tuesday, the 2nd of November 2010 as the start date, the first month is calculated as spanning from Tuesday, the 2nd of November 2010 to Tuesday, the 30th of November 2010. • DAYHOURVIEW—Provide peak bandwidth and stream count per hour for each standard week within the specified date range.[2] • DAYHALFHOURVIEW—Provide peak bandwidth and stream count per thirty minute intervals for each standard week within the specified date range. • DAYFIFTEENVIEW—Provide peak bandwidth and stream count per fifteen minute intervals for each standard week within the specified date range. • DAYMINUTEVIEW—Provide peak bandwidth and stream count per minute for each standard week within the specified date range. • DAYFIVEMINVIEW—Provide peak bandwidth and stream count per five minute intervals for each standard week within the specified date range. Available for the CapacityPlanning message.
asmConfig	Mandatory if the messageType is set to one of the below: • StreamHistory • StreamListHistory • TrickModeHistory • TrickModeListHistory • PlayServerHistory • ContentState • ContentRange	Specifies the installation type. 1 denotes ISA and 2 denotes RTSP. The default is set to 1.

1. ISV = Integrated Streamer-Vault.
2. A standard week is from Sunday to Saturday.

RTSP Stream Diagnostics Requests

For RTSP Stream Diagnostic requests, regardless of the request type, the same HTTP request header format is used. Basic request requirements include the following:

- All requests are sent by means of POST.
- Uniform Resource Identifier () specifies the root XML tag.
- HTTP version is 1.0.
- Entity bodies are used to convey XML data.

Required entity headers are the following:

- content-length: Bytes (length of XML data)
- content-type: text/xml
- cseq: Unique numeric ID

Other entity headers are optional. For example, the HTTP header may include a date header. However, non-required entity headers may be ignored by the server.

The specifies the root service for the message followed by the query string. The query string has the syntax of action=<xml root tag>. The always begins with the service name. For example, if the service name is `PlayoutDetails` and the XML root tag is `<GetCurrentlyPlayingDetails>`, the is `/apis/PlayoutDetails?action=GetCurrentlyPlayingDetails`.

The following example is the complete HTTP POST message:

```
POST /apis/PlayoutDetails?action=GetCurrentlyPlayingDetails HTTP/1.0
    User-Agent: HTTPTool/1.01
    Content-Type: text/xml
    Content-Length: 60
    CSeq: 123

<?xml version="1.0" encoding="utf-8"?> <GetCurrentlyPlayingDetails />
```

Response Messages

There are two different response message formats:

- Monitoring responses
- RTSP Stream Diagnostics responses

Monitoring Response

All response messages for the Monitoring APIs return an XML document.

RTSP Stream Diagnostics

The same HTTP response header is used regardless of the response type. The status code and status text in the response indicate whether the server received and processed the request. Some HTTP response messages consist of only the HTTP header, while others consist of both the HTTP header and XML message body.

The only required entity header is cseq, which is a unique numeric ID.

The only required entity header when doing a POST with the XML message body is the content-type, text/xml.

Other entity headers are optional. For example, the HTTP header may include a date header. However, non-required entity headers may be ignored by the client when processing the response.

The sequence number specified in the HTTP-response must match the sequence number in the HTTP request.

The status codes and status text are specific to the HTTP-response. Chapter 4, "RTSP APIs" provides a list of appropriate status codes for each RTSP message.

The following example is an RTSP Stream Diagnostics HTTP response with no XML body:

```
HTTP/1.0 200 OK
Date: Mon, 02 Jun 2008 22:50:45 GMT
CSeq: 123
```

The following example is an RTSP Stream Diagnostics HTTP response with the XML body (body not shown):

```
HTTP/1.0 200 OK
Date: Mon, 02 Jun 2008 22:50:45 GMT
Server: Apache/1.3.33 (UNIX) PHP/4.4.8
X-Powered-By: PHP/4.4.8
Connection: close
Content-Type: text/xml
CSeq: 123
```

Message Flow for RTSP Stream Diagnostics APIs

This section covers the HTTP message flow and possible causes for incomplete message transactions.

Request and Response

The client connects to the server, sends an HTTP request, and waits for an HTTP response. During this time, the client must not send any other requests using this connection. The client can establish a separate connection to send another request in parallel.

The server processes the request and sends an HTTP response to the client on the same connection on which it received the request. In the case of RTSP messages, the response must use the same sequence number that was specified in the request. When the client receives the response, it validates the sequence number before processing the response.

Based on the HTTP headers, the client or server may close the connection or the connection may be left open for subsequent requests from the client. The client can have several open connections to the server at any given time.

Request Timeout

The client sets a timer when it sends an HTTP request. This timer represents the maximum amount of time the client waits for a response from the server. For an RTSP request, a typical timeout period is five seconds.

If the client fails to receive an HTTP response before the timer expires, the client must consider the request as failed. The client can discard the request, immediately retry the request, or retry the request at a later time. The method of handling the failure is implementation-specific, and may vary by importance of the request type.

Invalid Sequence Number

When the client receives an HTTP response with a sequence number that differs from the request sequence number, the client discards the message and continues to wait for a valid response until the timeout threshold has been reached.

Connection Lost

During the HTTP transaction, if the connection to the server is lost at any time prior to receiving a valid HTTP response message, the client must consider the request as having failed. The client should immediately retry the request. If the retry fails, the client can discard the request or retry the request at a later time. The method of handling the failure is implementation-specific, and may vary by the importance of the request type.

In most cases, the server has started processing the request prior to the connection loss. The server should finish processing the request. The server must not establish a connection with the client in order to return the response. It is up to the client to reconnect and retransmit the request.

TV Playout Interface

The TV Playout APIs use the **curl** utility to retrieve content from the Cisco VDS using HTTP. The basic **curl** syntax is as follows:

```
curl [options] "url"
```

In the following example, **curl** is used to return a list of TV Playout channels from the specified URL and to send this output to the file reply_1_5.xml:

```
curl -o reply_1_5.xml "http://209.165.201.1/api/services/configure/system/outputchannels"
```

An HTTP POST method is used to provide the VDS with information. Using **curl**, the syntax of the POST message is as follows:

```
curl -o filename -F "fileupload=@xml_filename" "url"
```

The data is specified in the XML file. In the following example, the file barker.xml, which contains the configuration for creating a barker stream, is posted to the VDS:

```
curl -o reply_1_2.xml -F "fileupload=@xml_barker.xml"
"http://209.165.201.1/api/services/configure/array/barkerstream"
```

The barker.xml file contains the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<List xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:ws="http://www.cisco.com/schemas/VCPBU/CDS-TV/R0/ciscowebsvcs"
      Type="Barker"
      Name="Barker1"
      Channel="NBC">
      <Content Name="BBX_00_102000004.mpg" Loops="2"/>
      <Content Name="BBX_00_102000005.mpg" Loops="4"/>
</List>
```

All response messages for the TV Playout APIs return an XML document.

For more information on the **curl** utility, see the manual (man) pages located at the following URL:

http://curl.haxx.se/docs/manpage.html

The TV Playout interface uses method overloading to allow some parameters to be specified in the request message. To overload an HTTP method, you specify the intended method using the key "_method=" appended to the URL. The following is an example of overloading an HTTP POST method with an HTTP GET method:

```
curl -o reply_1_3.xml -F "fileupload=@request.xml" "http://209.165.201.1
/api/services/report/system/streams/playout/\(_method=GET\)"
```

In this example, the client requests a report of TV Playout streams for a period of time specified in the file last_modified.xml. The file includes the following elements:

```
<param name="FromDate" value="2010-5-30"/>
<param name="ToDate" value="2010-5-30"/>
```

The "TV Playout Errors" section on page 10-17 provides an overview of status codes and status text for TV Playout APIs.